

**GigaDevice Semiconductor Inc.**

**GD32E50x**

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

**Firmware Library  
User Guide**

Revision 1.6

(Dec. 2023)

# Table of Contents

<b>Table of Contents .....</b>	<b>2</b>
<b>List of Figures .....</b>	<b>5</b>
<b>List of Tables .....</b>	<b>6</b>
<b>1. Introduction.....</b>	<b>31</b>
<b>1.1. Rules of User Manual and Firmware Library .....</b>	<b>31</b>
1.1.1. Peripherals.....	31
1.1.2. Naming rules.....	32
<b>2. Firmware Library Overview .....</b>	<b>34</b>
<b>2.1. File Structure of Firmware Library .....</b>	<b>34</b>
2.1.1. Examples Folder .....	35
2.1.2. Firmware Folder.....	35
2.1.3. Template Folder .....	36
2.1.4. Utilities Folder .....	38
<b>2.2. File descriptions of Firmware Library .....</b>	<b>39</b>
<b>3. Firmware Library of Standard Peripherals .....</b>	<b>40</b>
<b>3.1. Overview of Firmware Library of Standard Peripherals.....</b>	<b>40</b>
<b>3.2. ADC .....</b>	<b>40</b>
3.2.1. Descriptions of Peripheral registers.....	40
3.2.2. Descriptions of Peripheral functions .....	41
<b>3.3. BKP.....</b>	<b>74</b>
3.3.1. Descriptions of Peripheral registers.....	75
3.3.2. Descriptions of Peripheral functions .....	75
<b>3.4. CAN .....</b>	<b>87</b>
3.4.1. Descriptions of Peripheral registers.....	87
3.4.2. Descriptions of Peripheral functions .....	89
<b>3.5. CRC .....</b>	<b>115</b>
3.5.1. Descriptions of Peripheral registers.....	115
3.5.2. Descriptions of Peripheral functions .....	116
<b>3.6. CTC.....</b>	<b>123</b>
3.6.1. Descriptions of Peripheral registers.....	123
3.6.2. Descriptions of Peripheral functions .....	124
<b>3.7. CMP .....</b>	<b>136</b>
3.7.1. Descriptions of Peripheral registers.....	136
3.7.2. Descriptions of Peripheral functions .....	137

<b>3.8. DAC .....</b>	<b>142</b>
3.8.1. Descriptions of Peripheral registers .....	143
3.8.2. Descriptions of Peripheral functions .....	143
<b>3.9. DBG .....</b>	<b>164</b>
3.9.1. Descriptions of Peripheral registers .....	164
3.9.2. Descriptions of Peripheral functions .....	164
<b>3.10. DMA.....</b>	<b>170</b>
3.10.1. Descriptions of Peripheral registers .....	170
3.10.2. Descriptions of Peripheral functions .....	170
<b>3.11. ENET .....</b>	<b>190</b>
3.11.1. Descriptions of Peripheral registers .....	190
3.11.2. Descriptions of Peripheral functions .....	193
<b>3.12. EXMC .....</b>	<b>281</b>
3.12.1. Descriptions of Peripheral registers .....	281
3.12.2. Descriptions of Peripheral functions .....	281
<b>3.13. EXTI.....</b>	<b>300</b>
3.13.1. Descriptions of Peripheral registers .....	301
3.13.2. Descriptions of Peripheral functions .....	301
<b>3.14. FMC .....</b>	<b>308</b>
3.14.1. Descriptions of Peripheral registers .....	308
3.14.2. Descriptions of Peripheral functions .....	309
<b>3.15. FWDGT.....</b>	<b>327</b>
3.15.1. Descriptions of Peripheral registers .....	327
3.15.2. Descriptions of Peripheral functions .....	327
<b>3.16. GPIO.....</b>	<b>332</b>
3.16.1. Descriptions of Peripheral registers .....	332
3.16.2. Descriptions of Peripheral functions .....	333
<b>3.17. SHRTIMER .....</b>	<b>351</b>
3.17.1. Descriptions of Peripheral registers .....	352
3.17.2. Descriptions of Peripheral functions .....	354
<b>3.18. I2C .....</b>	<b>420</b>
3.18.1. Descriptions of Peripheral registers .....	421
3.18.2. Descriptions of Peripheral functions .....	421
<b>3.19. MISC.....</b>	<b>484</b>
3.19.1. Descriptions of Peripheral registers .....	484
3.19.2. Descriptions of Peripheral functions .....	484
<b>3.20. PMU.....</b>	<b>492</b>
3.20.1. Descriptions of Peripheral registers .....	492
3.20.2. Descriptions of Peripheral functions .....	492

<b>3.21.</b>	<b>RCU .....</b>	<b>505</b>
3.21.1.	Descriptions of Peripheral registers .....	505
3.21.2.	Descriptions of Peripheral functions .....	506
<b>3.22.</b>	<b>RTC .....</b>	<b>544</b>
3.22.1.	Descriptions of Peripheral registers .....	544
3.22.2.	Descriptions of Peripheral functions .....	545
<b>3.23.</b>	<b>SDIO .....</b>	<b>552</b>
3.23.1.	Descriptions of Peripheral registers .....	552
3.23.2.	Descriptions of Peripheral functions .....	553
<b>3.24.</b>	<b>SPI .....</b>	<b>583</b>
3.24.1.	Descriptions of Peripheral registers .....	583
3.24.2.	Descriptions of Peripheral functions .....	584
<b>3.25.</b>	<b>SQPI .....</b>	<b>611</b>
3.25.1.	Descriptions of Peripheral registers .....	611
3.25.2.	Descriptions of Peripheral functions .....	611
<b>3.26.</b>	<b>TIMER .....</b>	<b>617</b>
3.26.1.	Descriptions of Peripheral registers .....	618
3.26.2.	Descriptions of Peripheral functions .....	618
<b>3.27.</b>	<b>TMU .....</b>	<b>674</b>
3.27.1.	Descriptions of Peripheral registers .....	674
3.27.2.	Descriptions of Peripheral functions .....	674
<b>3.28.</b>	<b>USART .....</b>	<b>680</b>
3.28.1.	Descriptions of Peripheral registers .....	680
3.28.2.	Descriptions of Peripheral functions .....	681
<b>3.29.</b>	<b>WWDGT .....</b>	<b>735</b>
3.29.1.	Descriptions of Peripheral registers .....	736
3.29.2.	Descriptions of Peripheral functions .....	736
<b>4.</b>	<b>Revision history .....</b>	<b>741</b>



## List of Figures

Figure 2-1. File structure of firmware library of GD32E50x .....	34
Figure 2-2. Select peripheral example files .....	36
Figure 2-3. Copy the peripheral example files .....	37
Figure 2-4. Open the project file .....	37
Figure 2-5. Configure project files .....	38
Figure 2-6. Compile-debug-download .....	38

# List of Tables

Table 1-1. Peripherals .....	31
Table 2-1. Function descriptions of Firmware Library .....	39
Table 3-1. Peripheral function format of Firmware Library .....	40
Table 3-2. ADC Registers .....	40
Table 3-3. ADC firmware function .....	41
Table 3-4. Function adc_deinit .....	42
Table 3-5. Function adc_enable .....	43
Table 3-6. Function adc_disable .....	43
Table 3-7. Function adc_calibration_enable .....	44
Table 3-8. Function adc_calibration_number .....	44
Table 3-9. Function adc_dma_mode_enable .....	45
Table 3-10. Function adc_dma_mode_disable .....	46
Table 3-11. Function adc_tempsensor_vrefint_enable .....	46
Table 3-12. Function adc_tempsensor_vrefint_disable .....	47
Table 3-13. Function adc_discontinuous_mode_config .....	47
Table 3-14. Function adc_mode_config .....	48
Table 3-15. Function adc_special_function_config .....	49
Table 3-16. Function adc_data_alignment_config .....	50
Table 3-17. Function adc_channel_length_config .....	50
Table 3-18. Function adc_regular_channel_config .....	51
Table 3-19. Function adc_inserted_channel_config .....	52
Table 3-20. Function adc_inserted_channel_offset_config .....	53
Table 3-21. Function adc_channel_differential_mode_config .....	54
Table 3-22. Function adc_external_trigger_config .....	55
Table 3-23. Function adc_external_trigger_source_config .....	56
Table 3-24. Function adc_software_trigger_enable .....	58
Table 3-25. Function adc_regular_data_read .....	59
Table 3-26. Function adc_inserted_data_read .....	59
Table 3-27. Function adc_sync_mode_convert_value_read .....	60
Table 3-28. Function adc_watchdog0_single_channel_enable .....	60
Table 3-29. Function adc_watchdog0_group_channel_enable .....	61
Table 3-30. Function adc_watchdog0_disable .....	62
Table 3-31. Function adc_watchdog1_channel_config .....	62
Table 3-32. Function adc_watchdog2_channel_config .....	63
Table 3-33. Function adc_watchdog1_disable .....	64
Table 3-34. Function adc_watchdog2_disable .....	64
Table 3-35. Function adc_watchdog0_threshold_config .....	65
Table 3-36. Function adc_watchdog1_threshold_config .....	65
Table 3-37. Function adc_watchdog2_threshold_config .....	66
Table 3-38. Function adc_resolution_config .....	67

Table 3-39. Function <code>adc_oversample_mode_config</code> .....	67
Table 3-40. Function <code>adc_oversample_mode_enable</code> .....	69
Table 3-41. Function <code>adc_oversample_mode_disable</code> .....	70
Table 3-42. Function <code>adc_flag_get</code> .....	70
Table 3-43. Function <code>adc_flag_clear</code> .....	71
Table 3-44. Function <code>adc_interrupt_enable</code> .....	72
Table 3-45. Function <code>adc_interrupt_disable</code> .....	72
Table 3-46. Function <code>adc_interrupt_flag_get</code> .....	73
Table 3-47. Function <code>adc_interrupt_flag_clear</code> .....	74
Table 3-48. BKP Registers .....	75
Table 3-49. BKP firmware function.....	75
Table 3-50. Enum <code>bkp_data_register_enum</code> .....	76
Table 3-51. Function <code>bkp_deinit</code> .....	77
Table 3-52. Function <code>bkp_write_data</code> .....	77
Table 3-53. Function <code>bkp_data_read</code> .....	78
Table 3-54. Function <code>bkp_rtc_calibration_output_enable</code> .....	78
Table 3-55. Function <code>bkp_rtc_calibration_output_disable</code> .....	79
Table 3-56. Function <code>bkp_rtc_signal_output_enable</code> .....	79
Table 3-57. Function <code>bkp_rtc_signal_output_disable</code> .....	80
Table 3-58. Function <code>bkp_rtc_output_select</code> .....	80
Table 3-59. Function <code>bkp_rtc_clock_output_select</code> .....	81
Table 3-60. Function <code>bkp_rtc_clock_calibration_direction</code> .....	81
Table 3-61. Function <code>bkp_rtc_calibration_value_set</code> .....	82
Table 3-62. Function <code>bkp_tamper_detection_enable</code> .....	83
Table 3-63. Function <code>bkp_tamper_detection_disable</code> .....	83
Table 3-64. Function <code>bkp_tamper_active_level_set</code> .....	83
Table 3-65. Function <code>bkp_tamper_interrupt_enable</code> .....	84
Table 3-66. Function <code>bkp_tamper_interrupt_disable</code> .....	85
Table 3-67. Function <code>bkp_flag_get</code> .....	85
Table 3-68. Function <code>bkp_flag_clear</code> .....	86
Table 3-69. Function <code>bkp_interrupt_flag_get</code> .....	86
Table 3-70. Function <code>bkp_interrupt_flag_clear</code> .....	87
Table 3-71. CAN Registers .....	87
Table 3-72. CAN-FD Registers .....	88
Table 3-73. CAN firmware function .....	89
Table 3-74. <code>can_parameter_struct</code> .....	90
Table 3-75. <code>can_transmit_message_struct</code> .....	90
Table 3-76. <code>can_transmit_message_struct</code> (for CAN-FD).....	90
Table 3-77. <code>can_receive_message_struct</code> .....	91
Table 3-78. <code>can_receive_message_struct</code> (for CAN-FD).....	91
Table 3-79. <code>can_fd_tdc_struct</code> (for CAN-FD).....	91
Table 3-80. <code>can_fdframe_struct</code> (for CAN-FD) .....	91
Table 3-81. <code>can_filter_parameter_struct</code> .....	92
Table 3-82. Enum <code>can_interrupt_flag_enum</code> .....	92

Table 3-83. Enum can_flag_enum .....	93
Table 3-84. Enum can_error_enum .....	94
Table 3-85. Enum can_transmit_state_enum .....	94
Table 3-86. Enum can_format_fifo_enum .....	94
Table 3-87. Enum can_struct_type_enum .....	94
Table 3-88. Function can_deinit .....	95
Table 3-89. Function can_struct_para_init .....	95
Table 3-90. Function can_init .....	96
Table 3-91. Function can_filter_init .....	96
Table 3-92. Function can_filter_mask_mode_init .....	97
Table 3-94. Function can_monitor_mode_set .....	98
Table 3-95. Function can_fd_init .....	98
Table 3-97. Function can_fd_function_enable .....	99
Table 3-98. Function can_fd_function_disable .....	100
Table 3-99. Function can1_filter_start_bank .....	100
Table 3-100. Function can_debug_freeze_enable .....	101
Table 3-101. Function can_debug_freeze_disable .....	101
Table 3-102. Function can_time_trigger_mode_enable .....	102
Table 3-103. Function can_time_trigger_mode_disable .....	102
Table 3-104. Function can_message_transmit .....	103
Table 3-105. Function can_transmit_states .....	104
Table 3-106. Function can_transmission_stop .....	104
Table 3-107. Function can_message_receive .....	105
Table 3-108. Function can_fifo_release .....	106
Table 3-109. Function can_receive_message_length_get .....	106
Table 3-110. Function can_working_mode_set .....	107
Table 3-111. Function can_wakeup .....	107
Table 3-112. Function can_error_get .....	108
Table 3-113. Function can_receive_error_number_get .....	108
Table 3-114. Function can_transmit_error_number_get .....	109
Table 3-115. Function can_interrupt_enable .....	109
Table 3-116. Function can_interrupt_disable .....	110
Table 3-117. Function can_flag_get .....	111
Table 3-118. Function can_flag_clear .....	113
Table 3-119. Function can_interrupt_flag_get .....	113
Table 3-120. Function can_interrupt_flag_clear .....	115
Table 3-121. CRC Registers .....	116
Table 3-122. CRC firmware function .....	116
Table 3-123. Function crc_deinit .....	116
Table 3-124. Function crc_data_register_reset .....	117
Table 3-125. Function crc_reverse_output_data_enable .....	117
Table 3-126. Function crc_reverse_output_data_disable .....	118
Table 3-127. Function crc_input_data_reverse_config .....	118
Table 3-128. Function crc_data_register_read .....	119

Table 3-129. Function <code>crc_free_data_register_read</code> .....	119
Table 3-130. Function <code>crc_free_data_register_write</code> .....	120
Table 3-131. Function <code>crc_init_data_register_write</code> .....	120
Table 3-132. Function <code>crc_polynomial_size_set</code> .....	121
Table 3-133. Function <code>crc_polynomial_set</code> .....	121
Table 3-134. Function <code>crc_single_data_calculate</code> .....	122
Table 3-135. Function <code>crc_block_data_calculate</code> .....	122
Table 3-136. CTC Registers.....	123
Table 3-137. CTC firmware function.....	124
Table 3-138. Function <code>ctc_deinit</code> .....	124
Table 3-139. Function <code>ctc_counter_enable</code> .....	125
Table 3-140. Function <code>ctc_counter_disable</code> .....	125
Table 3-141. Function <code>ctc_irc48m_trim_value_config</code> .....	126
Table 3-142. Function <code>ctc_software_refsource_pulse_generate</code> .....	126
Table 3-143. Function <code>ctc_hardware_trim_mode_config</code> .....	127
Table 3-144. Function <code>ctc_refsource_polarity_config</code> .....	127
Table 3-145. Function <code>ctc_refsource_signal_select</code> .....	128
Table 3-146. Function <code>ctc_refsource_prescaler_config</code> .....	128
Table 3-147. Function <code>ctc_clock_limit_value_config</code> .....	129
Table 3-148. Function <code>ctc_counter_reload_value_config</code> .....	130
Table 3-149. Function <code>ctc_counter_capture_value_read</code> .....	130
Table 3-150. Function <code>ctc_counter_direction_read</code> .....	131
Table 3-151. Function <code>ctc_counter_reload_value_read</code> .....	131
Table 3-152. Function <code>ctc_irc48m_trim_value_read</code> .....	132
Table 3-153. Function <code>ctc_flag_get</code> .....	132
Table 3-154. Function <code>ctc_flag_clear</code> .....	133
Table 3-155. Function <code>ctc_interrupt_enable</code> .....	134
Table 3-156. Function <code>ctc_interrupt_disable</code> .....	134
Table 3-157. Function <code>ctc_interrupt_flag_get</code> .....	135
Table 3-158. Function <code>ctc_interrupt_flag_clear</code> .....	136
Table 3-157. CMP Registers.....	137
Table 3-158. CMP firmware function.....	137
Table 3-159. Enum <code>cmp_enum</code> .....	137
Table 3-160. Function <code>cmp_deinit</code> .....	137
Table 3-161. Function <code>cmp_mode_init</code> .....	138
Table 3-162. Function <code>cmp_output_init</code> .....	139
Table 3-163. Function <code>cmp_blanking_init</code> .....	140
Table 3-164. Function <code>cmp_enable</code> .....	140
Table 3-165. Function <code>cmp_disable</code> .....	141
Table 3-166. Function <code>cmp_lock_enable</code> .....	141
Table 3-167. Function <code>cmp_output_level_get</code> .....	142
Table 3-171. DAC Registers.....	143
Table 3-172. DAC firmware functions.....	143
Table 3-173. Function <code>dac_deinit</code> .....	144

Table 3-174. Function <code>dac_enable</code> .....	145
Table 3-175. Function <code>dac_disable</code> .....	145
Table 3-176. Function <code>dac_dma_enable</code> .....	146
Table 3-177. Function <code>dac_dma_disable</code> .....	146
Table 3-178. Function <code>dac_output_buffer_enable</code> .....	147
Table 3-179. Function <code>dac_output_buffer_disable</code> .....	147
Table 3-180. Function <code>dac_output_value_get</code> .....	148
Table 3-181. Function <code>dac_data_set</code> .....	149
Table 3-182. Function <code>dac_trigger_enable</code> .....	149
Table 3-183. Function <code>dac_trigger_disable</code> .....	150
Table 3-184. Function <code>dac_trigger_source_config</code> .....	151
Table 3-185. Function <code>dac_software_trigger_enable</code> .....	152
Table 3-186. Function <code>dac_wave_mode_config</code> .....	152
Table 3-187. Function <code>dac_lfsr_noise_config</code> .....	153
Table 3-188. Function <code>dac_triangle_noise_config</code> .....	154
Table 3-189. Function <code>dac_concurrent_enable</code> .....	155
Table 3-190. Function <code>dac_concurrent_disable</code> .....	155
Table 3-191. Function <code>dac_concurrent_software_trigger_enable</code> .....	156
Table 3-192. Function <code>dac_concurrent_output_buffer_enable</code> .....	156
Table 3-193. Function <code>dac_concurrent_output_buffer_disable</code> .....	157
Table 3-194. Function <code>dac_concurrent_data_set</code> .....	157
Table 3-195. Function <code>dac_output_fifo_enable</code> .....	158
Table 3-196. Function <code>dac_output_fifo_disable</code> .....	158
Table 3-197. Function <code>dac_output_fifo_number_get</code> .....	159
Table 3-198. Function <code>dac_flag_get</code> .....	160
Table 3-199. Function <code>dac_flag_clear</code> .....	160
Table 3-200. Function <code>dac_interrupt_enable</code> .....	161
Table 3-201. Function <code>dac_interrupt_disable</code> .....	162
Table 3-202. Function <code>dac_interrupt_flag_get</code> .....	162
Table 3-203. Function <code>dac_interrupt_flag_clear</code> .....	163
Table 3-209. DBG Registers.....	164
Table 3-210. DBG firmware function .....	164
Table 3-211. Enum <code>dbg_periph_enum</code> .....	165
Table 3-212. Function <code>dbg_deinit</code> .....	165
Table 3-213. Function <code>dbg_id_get</code> .....	166
Table 3-214. Function <code>dbg_low_power_enable</code> .....	166
Table 3-215. Function <code>dbg_low_power_disable</code> .....	167
Table 3-216. Function <code>dbg_periph_enable</code> .....	168
Table 3-217. Function <code>dbg_periph_disable</code> .....	168
Table 3-218. Function <code>dbg_trace_pin_enable</code> .....	169
Table 3-219. Function <code>dbg_trace_pin_disable</code> .....	169
Table 3-220. DMA Registers.....	170
Table 3-221. DMA firmware function .....	171
Table 3-222. Enum <code>dma_channel_enum</code> .....	171

Table 3-223. Structure dma_parameter_struct.....	172
Table 3-224. Function dma_deinit .....	172
Table 3-225. Function dma_struct_para_init .....	173
Table 3-226. Function dma_init.....	173
Table 3-227. Function dma_circulation_enable .....	174
Table 3-228. Function dma_circulation_disable .....	175
Table 3-229. Function dma_memory_to_memory_enable .....	175
Table 3-230. Function dma_memory_to_memory_disable .....	176
Table 3-231. Function dma_channel_enable .....	176
Table 3-232. Function dma_channel_disable .....	177
Table 3-233. Function dma_periph_address_config .....	178
Table 3-234. Function dma_memory_address_config .....	178
Table 3-235. Function dma_transfer_number_config .....	179
Table 3-236. Function dma_transfer_number_get.....	180
Table 3-237. Function dma_priority_config .....	180
Table 3-238. Function dma_memory_width_config .....	181
Table 3-239. Function dma_periph_width_config .....	182
Table 3-240. Function dma_memory_increase_enable .....	183
Table 3-241. Function dma_memory_increase_disable .....	183
Table 3-242. Function dma_periph_increase_enable .....	184
Table 3-243. Function dma_periph_increase_disable .....	184
Table 3-244. Function dma_transfer_direction_config .....	185
Table 3-245. Function dma_flag_get .....	186
Table 3-246. Function dma_flag_clear .....	186
Table 3-247. Function dma_interrupt_enable .....	187
Table 3-248. Function dma_interrupt_disable .....	188
Table 3-249. Function dma_interrupt_flag_get .....	189
Table 3-250. Function dma_interrupt_flag_clear .....	189
Table 3-251. ENET Registers .....	190
Table 3-252. ENET firmware function .....	193
Table 3-253. Structure enet_initpara_struct .....	196
Table 3-254. Structure enet_descriptors_struct .....	196
Table 3-255. Structure enet_ptp_systime_struct .....	197
Table 3-256. Enum enet_flag_enum .....	197
Table 3-257. Enum enet_flag_clear_enum .....	199
Table 3-258. Enum enet_int_enum .....	200
Table 3-259. Enum enet_int_flag_enum .....	201
Table 3-260. Enum enet_int_flag_clear_enum .....	202
Table 3-261. Enum enet_desc_reg_enum .....	203
Table 3-262. Enum enet_msc_counter_enum .....	203
Table 3-263. Enum enet_option_enum .....	204
Table 3-264. Enum enet_mediamode_enum .....	204
Table 3-265. Enum enet_chksumconf_enum .....	205
Table 3-266. Enum enet_frmrecept_enum .....	205

Table 3-267. Enum enet_registers_type_enum .....	205
Table 3-268. Enum enet_dmadirection_enum .....	206
Table 3-269. Enum enet_phydirection_enum .....	206
Table 3-270. Enum enet_regdirection_enum .....	206
Table 3-271. Enum enet_macaddress_enum .....	206
Table 3-272. Enum enet_descstate_enum .....	206
Table 3-273. Enum enet_msc_preset_enum .....	207
Table 3-274. Function enet_deinit .....	207
Table 3-275. Function enet_initpara_config .....	208
Table 3-276. Function enet_init .....	211
Table 3-277. Function enet_software_reset .....	212
Table 3-278. Function enet_rxframe_size_get .....	213
Table 3-279. Function enet_descriptors_chain_init .....	213
Table 3-280. Function enet_descriptors_ring_init .....	214
Table 3-281. Function enet_frame_receive .....	215
Table 3-282. Function enet_frame_transmit .....	215
Table 3-283. Function enet_transmit_checksum_config .....	216
Table 3-284. Function enet_enable .....	217
Table 3-285. Function enet_disable .....	217
Table 3-286. Function enet_mac_address_set .....	218
Table 3-287. Function enet_mac_address_get .....	219
Table 3-288. Function enet_flag_get .....	219
Table 3-289. Function enet_flag_clear .....	222
Table 3-290. Function enet_interrupt_enable .....	223
Table 3-291. Function enet_interrupt_disable .....	224
Table 3-292. Function enet_interrupt_flag_get .....	226
Table 3-293. Function enet_interrupt_flag_clear .....	227
Table 3-294. Function enet_tx_enable .....	229
Table 3-295. Function enet_tx_disable .....	229
Table 3-296. Function enet_rx_enable .....	230
Table 3-297. Function enet_rx_disable .....	230
Table 3-298. Function enet_registers_get .....	231
Table 3-299. Function enet_debug_status_get .....	231
Table 3-300. Function enet_address_filter_enable .....	232
Table 3-301. Function enet_address_filter_disable .....	233
Table 3-302. Function enet_address_filter_config .....	234
Table 3-303. Function enet_phy_config .....	235
Table 3-304. Function enet_phy_write_read .....	236
Table 3-305. Function enet_phyloopback_enable .....	236
Table 3-306. Function enet_phyloopback_disable .....	237
Table 3-307. Function enet_forward_feature_enable .....	237
Table 3-308. Function enet_forward_feature_disable .....	238
Table 3-309. Function enet_fliter_feature_enable .....	239
Table 3-310. Function enet_fliter_feature_disable .....	240



Table 3-311. Function enet_pauseframe_generate.....	240
Table 3-312. Function enet_pauseframe_detect_config .....	241
Table 3-313. Function enet_pauseframe_config.....	242
Table 3-314. Function enet_flowcontrol_threshold_config .....	242
Table 3-315. Function enet_flowcontrol_feature_enable .....	244
Table 3-316. Function enet_flowcontrol_feature_disable .....	244
Table 3-317. Function enet_dmaprocess_state_get .....	245
Table 3-318. Function enet_dmaprocess_resume.....	246
Table 3-319. Function enet_rxprocess_check_recovery.....	247
Table 3-320. Function enet_txfifo_flush .....	247
Table 3-321. Function enet_current_desc_address_get .....	248
Table 3-322. Function enet_desc_information_get .....	248
Table 3-323. Function enet_missed_frame_counter_get .....	249
Table 3-324. Function enet_desc_flag_get .....	250
Table 3-325. Function enet_desc_flag_set .....	252
Table 3-326. Function enet_desc_flag_clear .....	253
Table 3-327. Function enet_rx_desc_immediate_receive_complete_interrupt.....	253
Table 3-328. Function enet_rx_desc_delay_receive_complete_interrupt.....	254
Table 3-329. Function enet_rxframe_drop .....	255
Table 3-330. Function enet_dma_feature_enable.....	255
Table 3-331. Function enet_dma_feature_disable.....	256
Table 3-332. Function enet_rx_desc_enhanced_status_get .....	256
Table 3-333. Function enet_desc_select_enhanced_mode .....	257
Table 3-334. Function enet_ptp_enhanced_descriptors_chain_init .....	258
Table 3-335. Function enet_ptp_enhanced_descriptors_ring_init .....	258
Table 3-336. Function enet_ptpframe_receive_enhanced_mode .....	259
Table 3-337. Function enet_ptpframe_transmit_enhanced_mode .....	260
Table 3-338. Function enet_desc_select_normal_mode .....	260
Table 3-339. Function enet_ptp_normal_descriptors_chain_init .....	261
Table 3-340. Function enet_ptp_normal_descriptors_ring_init .....	262
Table 3-341. Function enet_ptpframe_receive_normal_mode .....	262
Table 3-342. Function enet_ptpframe_transmit_normal_mode .....	263
Table 3-343. Function enet_wum_filter_register_pointer_reset .....	264
Table 3-344. Function enet_wum_filter_config .....	264
Table 3-345. Function enet_wum_feature_enable .....	265
Table 3-346. Function enet_wum_feature_disable .....	266
Table 3-347. Function enet_msc_counters_reset.....	266
Table 3-348. Function enet_msc_feature_enable .....	267
Table 3-349. Function enet_msc_feature_disable .....	267
Table 3-350. Function enet_msc_counters_preset_config.....	268
Table 3-351. Function enet_msc_counters_get.....	269
Table 3-352. Function enet_ptp_subsecond_2_nanosecond.....	269
Table 3-353. Function enet_ptp_nanosecond_2_subsecond.....	270
Table 3-354. Function enet_ptp_feature_enable.....	270

Table 3-355. Function enet_ptp_feature_disable .....	271
Table 3-356. Function enet_ptp_timestamp_function_config .....	272
Table 3-357. Function enet_ptp_subsecond_increment_config .....	274
Table 3-358. Function enet_ptp_timestamp_addend_config .....	274
Table 3-359. Function enet_ptp_timestamp_update_config .....	275
Table 3-360. Function enet_ptp_expected_time_config .....	275
Table 3-361. Function enet_ptp_system_time_get .....	276
Table 3-362. enet_ptp_pps_output_frequency_config .....	276
Table 3-363. enet_ptp_start .....	277
Table 3-364. enet_ptp_finecorrection_adjfreq .....	278
Table 3-365. enet_ptp_coarsecorrection_systime_update .....	279
Table 3-366. enet_ptp_finecorrection_settime .....	279
Table 3-367. enet_ptp_flag_get .....	280
Table 3-368. Function enet_initpara_reset .....	281
Table 3-369. EXMC Registers .....	281
Table 3-370. EXMC firmware function .....	282
Table 3-371. Structure exmc_norsram_timing_parameter_struct .....	282
Table 3-372. Structure exmc_norsram_parameter_struct .....	283
Table 3-373. Structure exmc_nand_pccard_timing_parameter_struct .....	283
Table 3-374. Structure exmc_nand_parameter_struct .....	283
Table 3-375. Structure exmc_pccard_parameter_struct .....	284
Table 3-376. Function exmc_norsram_deinit .....	284
Table 3-377. Function exmc_norsram_struct_para_init .....	285
Table 3-378. Function exmc_norsram_init .....	285
Table 3-379. Function exmc_norsram_enable .....	287
Table 3-380. Function exmc_norsram_disable .....	287
Table 3-381. Function exmc_norsram_page_size_config .....	288
Table 3-382. Function exmc_nand_deinit .....	289
Table 3-383. Function exmc_nand_struct_para_init .....	289
Table 3-384. Function exmc_nand_init .....	290
Table 3-385. Function exmc_nand_enable .....	291
Table 3-386. Function exmc_nand_disable .....	291
Table 3-387. Function exmc_nand_ecc_config .....	292
Table 3-388. Function exmc_ecc_get .....	292
Table 3-389. Function exmc_pccard_deinit .....	293
Table 3-390. Function exmc_pccard_struct_para_init .....	293
Table 3-391. Function exmc_pccard_init .....	294
Table 3-392. Function exmc_pccard_enable .....	295
Table 3-393. Function exmc_pccard_disable .....	295
Table 3-394. Function exmc_interrupt_enable .....	296
Table 3-395. Function exmc_interrupt_disable .....	296
Table 3-396. Function exmc_flag_get .....	297
Table 3-397. Function exmc_flag_clear .....	298
Table 3-398. Function exmc_interrupt_flag_get .....	299

Table 3-399. Function <code>exmc_interrupt_flag_clear</code> .....	300
Table 3-400. EXTI Registers.....	301
Table 3-401. EXTI firmware function .....	301
Table 3-402. Enum <code>exti_line_enum</code> .....	301
Table 3-403. Enum <code>exti_mode_enum</code> .....	302
Table 3-404. Enum <code>exti_trig_type_enum</code> .....	302
Table 3-405. Function <code>exti_deinit</code> .....	302
Table 3-406. Function <code>exti_init</code> .....	303
Table 3-407. Function <code>exti_interrupt_enable</code> .....	303
Table 3-408. Function <code>exti_interrupt_disable</code> .....	304
Table 3-409. Function <code>exti_event_enable</code> .....	304
Table 3-410. Function <code>exti_event_disable</code> .....	305
Table 3-411. Function <code>exti_software_interrupt_enable</code> .....	305
Table 3-412. Function <code>exti_software_interrupt_disable</code> .....	306
Table 3-413. Function <code>exti_flag_get</code> .....	306
Table 3-414. Function <code>exti_flag_clear</code> .....	307
Table 3-415. Function <code>exti_interrupt_flag_get</code> .....	307
Table 3-416. Function <code>exti_interrupt_flag_clear</code> .....	308
Table 3-417. FMC Registers .....	308
Table 3-418. FMC firmware function .....	309
Table 3-419. <code>fmc_state_enum</code> .....	310
Table 3-420. Function <code>fmc_unlock</code> .....	310
Table 3-421. Function <code>fmc_lock</code> .....	310
Table 3-422. Function <code>fmc_wscnt_set</code> .....	311
Table 3-423. Function <code>fmc_prefetch_enable</code> .....	311
Table 3-424. Function <code>fmc_prefetch_disable</code> .....	312
Table 3-425. Function <code>fmc_ibus_enable</code> .....	312
Table 3-426. Function <code>fmc_ibus_disable</code> .....	313
Table 3-427. Function <code>fmc_ibus_reset</code> .....	313
Table 3-428. Function <code>fmc_dbus_enable</code> .....	314
Table 3-429. Function <code>fmc_dbus_disable</code> .....	314
Table 3-430. Function <code>fmc_dbus_reset</code> .....	315
Table 3-431. Function <code>fmc_page_erase</code> .....	315
Table 3-432. Function <code>fmc_mass_erase</code> .....	316
Table 3-433. Function <code>fmc_word_program</code> .....	316
Table 3-434. Function <code>ob_unlock</code> .....	317
Table 3-435. Function <code>ob_lock</code> .....	317
Table 3-436. Function <code>ob_erase</code> .....	318
Table 3-437. Function <code>ob_write_protection_enable</code> .....	318
Table 3-438. Function <code>ob_security_protection_config</code> .....	319
Table 3-439. Function <code>ob_user_write</code> .....	320
Table 3-440. Function <code>ob_data_program</code> .....	321
Table 3-441. Function <code>ob_user_get</code> .....	321
Table 3-442. Function <code>ob_data_program</code> .....	322

Table 3-443. Function ob_write_protection_get .....	322
Table 3-444. Function ob_security_protection_flag_get .....	323
Table 3-445. Function fmc_flag_get .....	323
Table 3-446. Function fmc_flag_clear .....	324
Table 3-447. Function fmc_interrupt_enable .....	325
Table 3-448. Function fmc_interrupt_disable .....	325
Table 3-449. Function fmc_interrupt_flag_get .....	326
Table 3-450. Function fmc_interrupt_flag_clear .....	326
Table 3-451. FWDGT Registers .....	327
Table 3-452. FWDGT firmware function .....	327
Table 3-453. Function fwdgt_write_enable .....	328
Table 3-454. Function fwdgt_write_disable .....	328
Table 3-455. Function fwdgt_enable .....	329
Table 3-456. Function fwdgt_prescaler_value_config .....	329
Table 3-457. Function fwdgt_reload_value_config .....	330
Table 3-458. Function fwdgt_config .....	330
Table 3-459. Function fwdgt_counter_reload .....	331
Table 3-460. Function fwdgt_flag_get .....	332
Table 3-461. GPIO Registers .....	332
Table 3-462. GPIO firmware function .....	333
Table 3-463. Function gpio_deinit .....	334
Table 3-464. Function gpio_afio_deinit .....	334
Table 3-465. Function gpio_init .....	335
Table 3-466. Function gpio_bit_set .....	336
Table 3-467. Function gpio_bit_reset .....	337
Table 3-468. Function gpio_bit_write .....	337
Table 3-469. Function gpio_port_write .....	338
Table 3-470. Function gpio_input_bit_get .....	338
Table 3-471. Function gpio_input_port_get .....	339
Table 3-472. Function gpio_output_bit_get .....	340
Table 3-473. Function gpio_output_port_get .....	340
Table 3-474. Function gpio_pin_remap_config .....	341
Table 3-475. Function gpio_afio_port_config .....	343
Table 3-476. Function gpio_ethernet_phy_select .....	347
Table 3-477. Function gpio_exti_source_select .....	348
Table 3-478. Function gpio_event_output_config .....	348
Table 3-479. Function gpio_event_output_enable .....	349
Table 3-480. Function gpio_event_output_disable .....	349
Table 3-481. Function gpio_pin_lock .....	350
Table 3-482. Function gpio_compensation_config .....	350
Table 3-483. Function gpio_compensation_flag_get .....	351
Table 3-484. SHRTIMER Register .....	352
Table 3-485. SHRTIMER firmware function .....	354
Table 3-486. Structure shrtimer_baseinit_parameter_struct .....	356

Table 3-487. Structure shrtimer_timerinit_parameter_struct .....	357
Table 3-488. Structure shrtimer_timercfg_parameter_struct .....	357
Table 3-489. Structure shrtimer_comparecfg_parameter_struct .....	358
Table 3-490. Structure shrtimer_exevcfg_parameter_struct .....	358
Table 3-491. Structure shrtimer_deadtimecfg_parameter_struct .....	358
Table 3-492. Structure shrtimer_carriersignalcfg_parameter_struct .....	358
Table 3-493. Structure shrtimer_synccfg_parameter_struct .....	359
Table 3-494. Structure shrtimer_bunchmode_parameter_struct .....	359
Table 3-495. Structure shrtimer_exeventcfg_parameter_struct .....	359
Table 3-496. Structure shrtimer_faultcfg_parameter_struct .....	360
Table 3-497. Structure shrtimer_adctrigcfg_parameter_struct .....	360
Table 3-498. Structure shrtimer_channel_outputcfg_parameter_struct .....	360
Table 3-499. Function shrtimer_deinit .....	360
Table 3-500. Function shrtimer_dll_calibration_start .....	361
Table 3-501. Function shrtimer_baseinit_struct_para_init .....	362
Table 3-502. Function shrtimer_timers_base_init .....	362
Table 3-503. Function shrtimer_timers_counter_enable .....	363
Table 3-504. Function shrtimer_timers_counter_enable .....	364
Table 3-505. Function shrtimer_timers_update_event_enable .....	365
Table 3-506. Function shrtimer_timers_update_event_enable .....	365
Table 3-507. Function shrtimer_software_update .....	366
Table 3-508. Function shrtimer_software_counter_reset .....	367
Table 3-509. Function shrtimer_timerinit_struct_para_init .....	367
Table 3-510. Function shrtimer_timers_waveform_init .....	368
Table 3-511. Function shrtimer_timercfg_struct_para_init .....	369
Table 3-512. Function shrtimer_slavetimer_waveform_config .....	370
Table 3-513. Function shrtimer_comparecfg_struct_para_init .....	371
Table 3-514. Function shrtimer_slavetimer_waveform_compare_config .....	371
Table 3-515. Function shrtimer_channel_outputcfg_struct_para_init .....	372
Table 3-516. Function shrtimer_slavetimer_waveform_channel_config .....	373
Table 3-517. Function shrtimer_slavetimer_waveform_channel_software_request .....	374
Table 3-518. Function shrtimer_slavetimer_waveform_channel_output_level_get .....	375
Table 3-519. Function shrtimer_slavetimer_waveform_channel_state_get .....	376
Table 3-520. Function shrtimer_channel_outputcfg_struct_para_init .....	377
Table 3-521. Function shrtimer_slavetimer_waveform_channel_software_request .....	377
Table 3-522. Function shrtimer_channel_outputcfg_struct_para_init .....	379
Table 3-523. Function shrtimer_slavetimer_carriersignal_config .....	379
Table 3-524. Function shrtimer_output_channel_enable .....	380
Table 3-525. Function shrtimer_output_channel_disable .....	381
Table 3-526. Function shrtimer_slavetimer_waveform_compare_config .....	381
Table 3-527. Function shrtimer_slavetimer_compare_value_get .....	382
Table 3-528. Function shrtimer_mastertimer_compare_value_config .....	383
Table 3-529. Function shrtimer_slavetimer_compare_value_get .....	384
Table 3-530. Function shrtimer_timers_counter_value_config .....	384

Table 3-531. Function <code>shrtimer_timers_counter_value_get</code> .....	385
Table 3-532. Function <code>shrtimer_timers_autoreload_value_config</code> .....	386
Table 3-533. Function <code>shrtimer_timers_autoreload_value_get</code> .....	386
Table 3-534. Function <code>shrtimer_timers_repetition_value_config</code> .....	387
Table 3-535. Function <code>shrtimer_timers_repetition_value_get</code> .....	388
Table 3-536. Function <code>shrtimer_exevfiter_struct_para_init</code> .....	389
Table 3-537. Function <code>shrtimer_slavetimer_exeevent_filtering_config</code> .....	389
Table 3-538. Function <code>shrtimer_exeeventcfg_struct_para_init</code> .....	390
Table 3-539. Function <code>shrtimer_exeevent_config</code> .....	391
Table 3-540. Function <code>shrtimer_exeevent_prescaler</code> .....	392
Table 3-541. Function <code>shrtimer_synccfg_struct_para_init</code> .....	392
Table 3-542. Function <code>shrtimer_synchronization_config</code> .....	393
Table 3-543. Function <code>shrtimer_faultcfg_struct_para_init</code> .....	394
Table 3-544. Function <code>shrtimer_fault_config</code> .....	394
Table 3-545. Function <code>shrtimer_fault_prescaler_config</code> .....	395
Table 3-546. Function <code>shrtimer_fault_input_enable</code> .....	396
Table 3-547. Function <code>shrtimer_fault_input_disable</code> .....	396
Table 3-548. Function <code>shrtimer_timers_dma_enable</code> .....	397
Table 3-549. Function <code>shrtimer_timers_dma_enable</code> .....	398
Table 3-550. Function <code>shrtimer_dmamode_config</code> .....	399
Table 3-551. Function <code>shrtimer_bunchmode_struct_para_init</code> .....	401
Table 3-552. Function <code>shrtimer_bunchmode_config</code> .....	401
Table 3-553. Function <code>shrtimer_bunchmode_enable</code> .....	402
Table 3-554. Function <code>shrtimer_bunchmode_disable</code> .....	403
Table 3-555. Function <code>shrtimer_bunchmode_flag_get</code> .....	403
Table 3-556. Function <code>shrtimer_bunchmode_software_start</code> .....	404
Table 3-557. Function <code>shrtimer_slavetimer_capture_config</code> .....	404
Table 3-558. Function <code>shrtimer_slavetimer_capture_software</code> .....	405
Table 3-559. Function <code>shrtimer_slavetimer_capture_value_read</code> .....	406
Table 3-560. Function <code>shrtimer_adctrigcfg_struct_para_init</code> .....	407
Table 3-561. Function <code>shrtimer_adc_trigger_config</code> .....	408
Table 3-562. Function <code>shrtimer_timers_flag_get</code> .....	408
Table 3-563. Function <code>shrtimer_timers_flag_clear</code> .....	410
Table 3-564. Function <code>shrtimer_common_flag_get</code> .....	411
Table 3-565. Function <code>shrtimer_common_flag_clear</code> .....	412
Table 3-566. Function <code>shrtimer_timers_interrupt_enable</code> .....	413
Table 3-567. Function <code>shrtimer_timers_interrupt_disable</code> .....	414
Table 3-568. Function <code>shrtimer_timers_interrupt_flag_get</code> .....	415
Table 3-569. Function <code>shrtimer_timers_interrupt_flag_clear</code> .....	416
Table 3-570. Function <code>shrtimer_common_interrupt_enable</code> .....	417
Table 3-571. Function <code>shrtimer_common_interrupt_disable</code> .....	418
Table 3-572. Function <code>shrtimer_common_interrupt_flag_get</code> .....	419
Table 3-573. Function <code>shrtimer_common_interrupt_flag_clear</code> .....	420
Table 3-574. I2C Registers .....	421



Table 3-575. I2C firmware function.....	421
Table 3-576. i2c_flag_enum .....	424
Table 3-577. i2c_interrupt_enum .....	425
Table 3-578. i2c_interrupt_flag_enum .....	425
Table 3-579. i2c2_interrupt_flag_enum .....	426
Table 3-580. Function i2c_deinit.....	426
Table 3-581. Function i2c_enable .....	427
Table 3-582. Function i2c_disable .....	427
Table 3-583. Function i2c_start_on_bus .....	428
Table 3-584. Function i2c_stop_on_bus .....	428
Table 3-585. Function i2c_slave_response_to_gcall_enable .....	429
Table 3-586. Function i2c_slave_response_to_gcall_disable .....	429
Table 3-587. Function i2c_stretch_scl_low_enable .....	430
Table 3-588. Function i2c_stretch_scl_low_disable .....	430
Table 3-589. Function i2c_data_transmit .....	431
Table 3-590. Function i2c_data_receive .....	431
Table 3-591. Function i2c_pec_transfer .....	432
Table 3-592. Function i2c_pec_enable .....	432
Table 3-593. Function i2c_pec_disable .....	433
Table 3-594. Function i2c_pec_value_get.....	433
Table 3-595. Function i2c_clock_config.....	434
Table 3-596. Function i2c_mode_addr_config .....	435
Table 3-597. Function i2c_smbus_type_config .....	436
Table 3-598. Function i2c_ack_config .....	436
Table 3-599. Function i2c_ackpos_config .....	437
Table 3-600. Function i2c_master_addressing .....	437
Table 3-601. Function i2c_dualaddr_enable .....	438
Table 3-602. Function i2c_dualaddr_disable .....	439
Table 3-603. Function i2c_dma_config .....	439
Table 3-604. Function i2c_dma_last_transfer_config .....	440
Table 3-605. Function i2c_software_reset_config .....	440
Table 3-606. Function i2c_smbus_alert_config .....	441
Table 3-607. Function i2c_smbus_arp_config .....	442
Table 3-608. Function i2c_sam_enable .....	442
Table 3-609. Function i2c_sam_disable .....	443
Table 3-610. Function i2c_sam_timeout_enable.....	443
Table 3-611. Function i2c_sam_timeout_disable .....	444
Table 3-612. Function i2c_start_early_termination_mode_config .....	444
Table 3-613. Function i2c_timeout_calculation_enable .....	445
Table 3-614. Function i2c_timeout_calculation_disable .....	445
Table 3-615. Function i2c_record_received_slave_address_enable .....	446
Table 3-616. Function i2c_record_received_slave_address_disable .....	446
Table 3-617. Function i2c_address_bit_compare_config .....	447
Table 3-618. Function i2c_status_clear_enable.....	448

Table 3-619. Function i2c_status_clear_disable .....	448
Table 3-620. Function i2c_start_early_termination_mode_config .....	449
Table 3-621. Function i2c_flag_get .....	449
Table 3-622. Function i2c_flag_clear .....	451
Table 3-623. Function i2c_interrupt_enable .....	452
Table 3-624. Function i2c_interrupt_disable .....	452
Table 3-625. Function i2c_interrupt_flag_get.....	453
Table 3-626. Function i2c_interrupt_flag_clear.....	455
Table 3-627. Function i2c_timing_config .....	456
Table 3-628. Function i2c_digital_noise_filter_config .....	456
Table 3-629. Function i2c_analog_noise_filter_enable .....	457
Table 3-630. Function i2c_analog_noise_filter_disable .....	458
Table 3-631. Function i2c_wakeup_from_deepsleep_enable.....	458
Table 3-632. Function i2c_wakeup_from_deepsleep_disable.....	459
Table 3-633. Function i2c_master_clock_config .....	459
Table 3-634. Function i2c2_master_transfer_direction_config.....	460
Table 3-635. Function i2c_address10_header_enable.....	461
Table 3-636. Function i2c_address10_header_disable.....	461
Table 3-637. Function i2c_address10_enable .....	462
Table 3-638. Function i2c_address10_disable .....	462
Table 3-639. Function i2c_automatic_end_enable .....	463
Table 3-640. Function i2c_automatic_end_disable .....	463
Table 3-641. Function i2c_address_config .....	464
Table 3-642. Function i2c_address_disable .....	464
Table 3-643. Function i2c_second_address_config.....	465
Table 3-644. Function i2c_second_address_disable .....	466
Table 3-645. Function i2c_receved_address_get .....	466
Table 3-646. Function i2c_slave_byte_control_enable.....	467
Table 3-647. Function i2c_slave_byte_control_disable.....	467
Table 3-648. Function i2c_nack_enable .....	468
Table 3-649. Function i2c_nack_disable .....	468
Table 3-650. Function i2c_reload_enable.....	469
Table 3-651. Function i2c_reload_disable .....	469
Table 3-652. Function i2c_transfer_byte_number_config.....	470
Table 3-653. Function i2c2_dma_enable .....	470
Table 3-654. Function i2c2_dma_disable .....	471
Table 3-655. Function i2c_smbus_alert_enable.....	472
Table 3-656. Function i2c_smbus_alert_disable.....	472
Table 3-657. Function i2c_smbus_default_addr_enable.....	473
Table 3-658. Function i2c_smbus_default_addr_disable.....	473
Table 3-659. Function i2c_smbus_host_addr_enable .....	474
Table 3-660. Function i2c_smbus_host_addr_disable .....	474
Table 3-661. Function i2c_extented_clock_timeout_enable.....	475
Table 3-662. Function i2c_extented_clock_timeout_disable.....	475



Table 3-663. Function i2c_clock_timeout_enable .....	476
Table 3-664. Function i2c_clock_timeout_disable .....	476
Table 3-665. Function i2c_bus_timeout_b_config.....	477
Table 3-666. Function i2c_bus_timeout_a_config.....	477
Table 3-667. Function i2c_idle_clock_timeout_config .....	478
Table 3-668. Function i2c2_flag_get .....	478
Table 3-669. Function i2c2_flag_clear .....	479
Table 3-670. Function i2c2_interrupt_enable .....	480
Table 3-671. Function i2c2_interrupt_disable .....	481
Table 3-672. Function i2c2_interrupt_flag_get.....	482
Table 3-673. Function i2c2_interrupt_flag_clear.....	483
Table 3-674. NVIC Registers .....	484
Table 3-675. SysTick Registers .....	484
Table 3-676. MISC firmware function .....	485
Table 3-677. IRQn_Type.....	485
Table 3-678. Function nvic_priority_group_set .....	487
Table 3-679. Function nvic_irq_enable.....	488
Table 3-680. Function nvic_irq_disable.....	488
Table 3-681. Function nvic_system_reset .....	489
Table 3-682. Function nvic_vector_table_set.....	489
Table 3-683. Function system_lowpower_set .....	490
Table 3-684. Function system_lowpower_reset.....	491
Table 3-685. Function systick_clksource_set .....	491
Table 3-686. PMU Registers.....	492
Table 3-687. PMU firmware function .....	492
Table 3-688. Function pmu_deinit .....	493
Table 3-689. Function pmu_lvd_select .....	493
Table 3-690. Function pmu_lvd_disable.....	494
Table 3-691. Function pmu_highdriver_mode_enable .....	495
Table 3-692. Function pmu_highdriver_mode_disable .....	495
Table 3-693. Function pmu_highdriver_switch_select.....	495
Table 3-694. Function pmu_lowdriver_mode_enable .....	496
Table 3-695. Function pmu_lowdriver_mode_disable .....	497
Table 3-696. Function pmu_lowpower_driver_config.....	497
Table 3-697. Function pmu_normalpower_driver_config .....	498
Table 3-698. Function pmu_to_sleepmode.....	498
Table 3-699. Function pmu_to_deepsleepmode .....	499
Table 3-700. Function pmu_to_deepsleepmode_1.....	499
Table 3-701. Function pmu_to_deepsleepmode_2.....	500
Table 3-702. Function pmu_to_standbymode .....	501
Table 3-703. Function pmu_backup_write_enable .....	502
Table 3-704. Function pmu_backup_write_disable .....	502
Table 3-705. Function pmu_wakeup_pin_enable .....	503
Table 3-706. Function pmu_wakeup_pin_disable .....	503

Table 3-707. Function pmu_flag_clear.....	504
Table 3-708. Function pmu_flag_get.....	505
Table 3-709. RCU Registers.....	506
Table 3-710. RCU firmware function .....	506
Table 3-711. Enum rcu_periph_enum .....	508
Table 3-712. Enum rcu_periph_sleep_enum .....	509
Table 3-713. Enum rcu_periph_reset_enum.....	509
Table 3-714. Enum rcu_flag_enum.....	511
Table 3-715. Enum rcu_int_flag_enum .....	512
Table 3-716. Enum rcu_int_flag_clear_enum .....	512
Table 3-717. Enum rcu_int_enum.....	513
Table 3-718. Enum rcu_osci_type_enum .....	513
Table 3-719. Enum rcu_clock_freq_enum.....	513
Table 3-720. Function rcu_deinit .....	514
Table 3-721. Function rcu_periph_clock_enable .....	514
Table 3-722. Function rcu_periph_clock_disable.....	515
Table 3-723. Function rcu_periph_clock_sleep_enable .....	515
Table 3-724. Function rcu_periph_clock_sleep_disable .....	516
Table 3-725. Function rcu_periph_reset_enable.....	516
Table 3-726. Function rcu_periph_reset_disable .....	517
Table 3-727. Function rcu_bkp_reset_enable .....	517
Table 3-728. Function rcu_bkp_reset_disable .....	518
Table 3-729. Function rcu_system_clock_source_config.....	518
Table 3-730. Function rcu_system_clock_source_get .....	519
Table 3-731. Function rcu_ahb_clock_config .....	519
Table 3-732. Function rcu_apb1_clock_config .....	520
Table 3-733. Function rcu_apb2_clock_config .....	520
Table 3-734. Function rcu_ckout0_config.....	521
Table 3-735. Function rcu_pll_config .....	522
Table 3-736. Function rcu_pllpresel_config .....	523
Table 3-737. Function rcu_predv0_config .....	523
Table 3-738. Function rcu_predv0_config .....	524
Table 3-739. Function rcu_predv1_config .....	525
Table 3-740. Function rcu_pll1_config .....	525
Table 3-741. Function rcu_pll2_config .....	526
Table 3-742. Function rcu_pllusbpresel_config.....	526
Table 3-743. Function rcu_pllusbpredv_config .....	527
Table 3-744. Function rcu_pllusb_config.....	527
Table 3-745. Function rcu_adc_clock_config.....	528
Table 3-746. Function rcu_usb_clock_config .....	529
Table 3-747. Function rcu_rtc_clock_config .....	530
Table 3-748. Function rcu_shrtimer_clock_config.....	530
Table 3-749. Function rcu_usart5_clock_config.....	531
Table 3-750. Function rcu_i2c2_clock_config.....	531

Table 3-751. Function rcu_ck48m_clock_config .....	532
Table 3-752. Function rcu_i2s1_clock_config.....	533
Table 3-753. Function rcu_i2s2_clock_config.....	533
Table 3-754. Function rcu_usbhsel_config .....	534
Table 3-755. Function rcu_usbdv_config.....	534
Table 3-756. Function rcu_lxtal_drive_capability_config.....	535
Table 3-757. Function rcu_osci_stab_wait .....	536
Table 3-758. Function rcu_osci_on .....	536
Table 3-759. Function rcu_osci_off.....	537
Table 3-760. Function rcu_osci_bypass_mode_enable .....	537
Table 3-761. Function rcu_osci_bypass_mode_disable .....	538
Table 3-762. Function rcu_irc8m_adjust_value_set.....	538
Table 3-763. Function rcu_hxtal_clock_monitor_enable .....	539
Table 3-764. Function rcu_hxtal_clock_monitor_disable .....	539
Table 3-765. Function rcu_deepsleep_voltage_set.....	540
Table 3-766. Function rcu_clock_freq_get.....	540
Table 3-767. Function rcu_flag_get.....	541
Table 3-768. Function rcu_all_reset_flag_clear .....	542
Table 3-769. Function rcu_interrupt_flag_get .....	542
Table 3-770. Function rcu_interrupt_flag_clear .....	543
Table 3-771. Function rcu_interrupt_enable.....	543
Table 3-772. Function rcu_interrupt_disable.....	544
Table 3-773. RTC Registers .....	544
Table 3-774. RTC firmware function.....	545
Table 3-775. Function rtc_configuration_mode_enter.....	545
Table 3-776. Function rtc_configuration_mode_exit .....	546
Table 3-777. Function rtc_lwoff_wait .....	546
Table 3-778. Function rtc_register_sync_wait .....	547
Table 3-779. Function rtc_counter_get.....	547
Table 3-780. Function rtc_counter_set.....	548
Table 3-781. Function rtc_prescaler_set.....	548
Table 3-782. Function rtc_alarm_config.....	549
Table 3-783. Function rtc_divider_get .....	549
Table 3-784. Function rtc_interrupt_enable.....	550
Table 3-785. Function rtc_interrupt_disable.....	550
Table 3-786. Function rtc_flag_get.....	551
Table 3-787. Function rtc_flag_clear.....	552
Table 3-788. SDIO Registers.....	553
Table 3-789. SDIO firmware function .....	553
Table 3-790. Function sdio_deinit .....	554
Table 3-791. Function sdio_clock_config .....	555
Table 3-792. Function sdio_hardware_clock_enable.....	556
Table 3-793. Function sdio_hardware_clock_disable.....	556
Table 3-794. Function sdio_bus_mode_set.....	557

Table 3-795. Function <code>sdio_power_state_set</code> .....	557
Table 3-796. Function <code>sdio_power_state_get</code> .....	558
Table 3-797. Function <code>sdio_clock_enable</code> .....	558
Table 3-798. Function <code>sdio_clock_disable</code> .....	559
Table 3-799. Function <code>sdio_command_response_config</code> .....	559
Table 3-800. Function <code>sdio_wait_type_set</code> .....	560
Table 3-801. Function <code>sdio_csm_enable</code> .....	561
Table 3-802. Function <code>sdio_csm_disable</code> .....	561
Table 3-803. Function <code>sdio_command_index_get</code> .....	562
Table 3-804. Function <code>sdio_response_get</code> .....	562
Table 3-805. Function <code>sdio_data_config</code> .....	563
Table 3-806. Function <code>sdio_data_transfer_config</code> .....	564
Table 3-807. Function <code>sdio_dsm_enable</code> .....	565
Table 3-808. Function <code>sdio_dsm_disable</code> .....	565
Table 3-809. Function <code>sdio_data_write</code> .....	566
Table 3-810. Function <code>sdio_data_read</code> .....	566
Table 3-811. Function <code>sdio_data_counter_get</code> .....	567
Table 3-812. Function <code>sdio_data_counter_get</code> .....	567
Table 3-813. Function <code>sdio_dma_enable</code> .....	568
Table 3-814. Function <code>sdio_dma_disable</code> .....	568
Table 3-815. Function <code>sdio_flag_get</code> .....	569
Table 3-816. Function <code>sdio_flag_clear</code> .....	570
Table 3-817. Function <code>sdio_interrupt_enable</code> .....	571
Table 3-818. Function <code>sdio_interrupt_disable</code> .....	572
Table 3-819. Function <code>sdio_interrupt_flag_get</code> .....	573
Table 3-820. Function <code>sdio_interrupt_flag_clear</code> .....	575
Table 3-821. Function <code>sdio_readwait_enable</code> .....	576
Table 3-822. Function <code>sdio_readwait_disable</code> .....	576
Table 3-823. Function <code>sdio_stop_readwait_enable</code> .....	577
Table 3-824. Function <code>sdio_stop_readwait_disable</code> .....	577
Table 3-825. Function <code>sdio_readwait_type_set</code> .....	578
Table 3-826. Function <code>sdio_operation_enable</code> .....	578
Table 3-827. Function <code>sdio_operation_disable</code> .....	579
Table 3-828. Function <code>sdio_suspend_enable</code> .....	579
Table 3-829. Function <code>sdio_suspend_disable</code> .....	580
Table 3-830. Function <code>sdio_ceata_command_enable</code> .....	580
Table 3-831. Function <code>sdio_ceata_command_disable</code> .....	581
Table 3-832. Function <code>sdio_ceata_interrupt_enable</code> .....	581
Table 3-833. Function <code>sdio_ceata_interrupt_disable</code> .....	582
Table 3-834. Function <code>sdio_ceata_command_completion_enable</code> .....	582
Table 3-835. Function <code>sdio_ceata_command_completion_disable</code> .....	583
Table 3-834. SPI/I2S Registers .....	583
Table 3-835. SPI/I2S firmware function.....	584
Table 3-836. <code>spi_parameter_struct</code> .....	585

Table 3-837. Function spi_i2s_deinit .....	585
Table 3-838. Function spi_struct_para_init .....	586
Table 3-839. Function spi_init .....	586
Table 3-840. Function spi_enable .....	587
Table 3-841. Function spi_disable .....	588
Table 3-842. Function i2s_init .....	588
Table 3-843. Function i2s_psc_config .....	589
Table 3-844. Function i2s_enable .....	591
Table 3-845. Function i2s_disable .....	591
Table 3-846. Function spi_nss_output_enable .....	592
Table 3-847. Function spi_nss_output_disable .....	592
Table 3-848. Function spi_nss_internal_high .....	593
Table 3-849. Function spi_nss_internal_low .....	593
Table 3-850. Function spi_dma_enable .....	594
Table 3-851. Function spi_dma_disable .....	594
Table 3-852. Function spi_i2s_data_frame_format_config .....	595
Table 3-853. Function spi_i2s_data_transmit .....	595
Table 3-854. Function spi_i2s_data_receive .....	596
Table 3-855. Function spi_bidirectional_transfer_config .....	597
Table 3-856. Function spi_i2s_format_error_clear .....	597
Table 3-857. Function spi_crc_polynomial_set .....	598
Table 3-858. Function spi_crc_polynomial_get .....	598
Table 3-859. Function spi_crc_on .....	599
Table 3-860. Function spi_crc_off .....	599
Table 3-861. Function spi_crc_next .....	600
Table 3-862. Function spi_crc_get .....	600
Table 3-863. Function spi_crc_error_clear .....	601
Table 3-864. Function spi_ti_mode_enable .....	602
Table 3-865. Function spi_ti_mode_disable .....	602
Table 3-866. Function spi_nssp_mode_enable .....	603
Table 3-867. Function spi_nssp_mode_disable .....	603
Table 3-868. Function i2s_init .....	604
Table 3-869. Function spi_quad_enable .....	605
Table 3-870. Function spi_quad_disable .....	605
Table 3-871. Function spi_quad_write_enable .....	606
Table 3-872. Function spi_quad_read_enable .....	606
Table 3-873. Function spi_quad_io23_output_enable .....	607
Table 3-874. Function spi_quad_io23_output_disable .....	607
Table 3-875. Function spi_i2s_interrupt_enable .....	608
Table 3-876. Function spi_i2s_interrupt_disable .....	609
Table 3-877. Function spi_i2s_interrupt_flag_get .....	609
Table 3-878. Function spi_i2s_flag_get .....	610
Table 3-880. SQPI Registers .....	611
Table 3-881. SQPI firmware function .....	612

Table 3-882. sqpi_parameter_struct.....	612
Table 3-883. Function sqpi_deinit .....	612
Table 3-884. Function sqpi_struct_para_init .....	613
Table 3-885. Function sqpi_init.....	613
Table 3-886. Function sqpi_read_id_command .....	614
Table 3-887. Function sqpi_special_command.....	614
Table 3-888. Function sqpi_read_command_config .....	615
Table 3-889. Function sqpi_write_command_config .....	616
Table 3-890. Function sqpi_low_id_receive.....	616
Table 3-891. Function sqpi_low_id_receive.....	617
Table 3-892. TIMEx Registers .....	618
Table 3-893. TIMEx firmware function.....	618
Table 3-894. Structure timer_parameter_struct .....	621
Table 3-895. Structure timer_break_parameter_struct.....	621
Table 3-896. Structure timer_oc_parameter_struct.....	621
Table 3-897. Structure timer_ic_parameter_struct.....	622
Table 3-898. Function timer_deinit.....	622
Table 3-899. Function timer_struct_para_init.....	622
Table 3-900. Function timer_init .....	623
Table 3-901. Function timer_enable .....	624
Table 3-902. Function timer_disable .....	624
Table 3-903. Function timer_auto_reload_shadow_enable .....	625
Table 3-904. Function timer_auto_reload_shadow_disable .....	625
Table 3-905. Function timer_update_event_enable .....	626
Table 3-906. Function timer_update_event_disable .....	626
Table 3-907. Function timer_counter_alignment .....	627
Table 3-908. Function timer_counter_up_direction .....	628
Table 3-909. timer_counter_down_direction .....	628
Table 3-910. Function timer_prescaler_config.....	629
Table 3-911. Function timer_repetition_value_config.....	629
Table 3-912. Function timer_autoreload_value_config .....	630
Table 3-913. Function timer_counter_value_config.....	631
Table 3-914. Function timer_counter_read .....	631
Table 3-915. Function timer_prescaler_read .....	632
Table 3-916. Function timer_single_pulse_mode_config .....	632
Table 3-917. Function timer_update_source_config.....	633
Table 3-918. Function timer_dma_enable .....	634
Table 3-919. Function timer_dma_disable .....	634
Table 3-920. Function timer_channel_dma_request_source_select.....	635
Table 3-921. Function timer_dma_transfer_config.....	636
Table 3-922. Function timer_event_software_generate.....	637
Table 3-923. Function timer_break_struct_para_init .....	638
Table 3-924. Function timer_break_config .....	639
Table 3-925. Function timer_break_enable.....	640

Table 3-926. Function timer_break_disable .....	640
Table 3-927. Function timer_automatic_output_enable .....	641
Table 3-928. Function timer_automatic_output_disable .....	641
Table 3-929. Function timer_primary_output_config .....	642
Table 3-930. Function timer_channel_control_shadow_config .....	642
Table 3-931. Function timer_channel_control_shadow_update_config .....	643
Table 3-932. Function timer_channel_output_struct_para_init .....	644
Table 3-933. Function timer_channel_output_config .....	644
Table 3-934. Function timer_channel_output_mode_config .....	645
Table 3-935. Function timer_channel_output_pulse_value_config .....	646
Table 3-936. Function timer_channel_output_shadow_config .....	647
Table 3-937. Function timer_channel_output_fast_config .....	648
Table 3-938. Function timer_channel_output_clear_config .....	649
Table 3-939. Function timer_channel_output_polarity_config .....	650
Table 3-940. Function timer_channel_complementary_output_polarity_config .....	650
Table 3-941. Function timer_channel_output_state_config .....	651
Table 3-942. Function timer_channel_complementary_output_state_config .....	652
Table 3-943. Function timer_channel_input_struct_para_init .....	653
Table 3-944. Function timer_input_capture_config .....	653
Table 3-945. Function timer_channel_input_capture_prescaler_config .....	654
Table 3-946. Function timer_channel_capture_value_register_read .....	655
Table 3-947. Function timer_input_pwm_capture_config .....	656
Table 3-948. Function timer_hall_mode_config .....	657
Table 3-949. Function timer_input_trigger_source_select .....	657
Table 3-950. Function timer_master_output_trigger_source_select .....	658
Table 3-951. Function timer_slave_mode_select .....	659
Table 3-952. Function timer_master_slave_mode_config .....	660
Table 3-953. Function timer_external_trigger_config .....	661
Table 3-954. Function timer_quadrature_decoder_mode_config .....	662
Table 3-955. Function timer_internal_clock_config .....	663
Table 3-956. Function timer_internal_trigger_as_external_clock_config .....	664
Table 3-957. Function timer_external_trigger_as_external_clock_config .....	664
Table 3-958. Function timer_external_clock_mode0_config .....	665
Table 3-959. Function timer_external_clock_mode1_config .....	666
Table 3-960. Function timer_external_clock_mode1_disable .....	667
Table 3-961. Function timer_write_chxval_register_config .....	668
Table 3-962. Function timer_output_value_selection_config .....	668
Table 3-963. Function timer_flag_get .....	669
Table 3-964. Function timer_flag_clear .....	670
Table 3-965. Function timer_interrupt_enable .....	671
Table 3-966. Function timer_interrupt_disable .....	671
Table 3-967. Function timer_interrupt_flag_get .....	672
Table 3-968. Function timer_interrupt_flag_clear .....	673
Table 3-969. TMU Registers .....	674



Table 3-970. TMU firmware function .....	674
Table 3-971. Function tmu_deinit .....	675
Table 3-972. Function tmu_enable .....	675
Table 3-973. Function tmu_mode_set.....	676
Table 3-974. Function tmu_idata0_write .....	676
Table 3-975. Function tmu_idata1_write .....	677
Table 3-976. Function tmu_data0_read .....	677
Table 3-977. Function tmu_data1_read .....	678
Table 3-978. Function tmu_interrupt_enable.....	678
Table 3-979. Function tmu_interrupt_disable.....	679
Table 3-980. Function tmu_flag_get.....	679
Table 3-981. Function tmu_interrupt_flag_get .....	680
Table 3-982. USART Registers .....	680
Table 3-983. USART firmware function.....	681
Table 3-984. Enum usart_flag_enum.....	684
Table 3-985. Enum usart5_flag_enum .....	684
Table 3-986. Enum usart_interrupt_flag_enum.....	685
Table 3-987. Enum usart5_interrupt_flag_enum.....	685
Table 3-988. Enum usart_interrupt_enum.....	686
Table 3-989. Enum usart5_interrupt_enum.....	686
Table 3-990. Enum usart_invert_enum.....	686
Table 3-991. Enum usart5_invert_enum.....	687
Table 3-992. Function usart_deinit.....	687
Table 3-993. Function usart_baudrate_set .....	688
Table 3-994. Function usart_parity_config .....	688
Table 3-995. Function usart_word_length_set.....	689
Table 3-996. Function usart_stop_bit_set.....	689
Table 3-997. Function usart_enable .....	690
Table 3-998. Function usart_disable .....	691
Table 3-999. Function usart_transmit_config.....	691
Table 3-1000. Function usart_receive_config .....	692
Table 3-1001. Function usart_oversample_config .....	692
Table 3-1002. Function usart_sample_bit_config.....	693
Table 3-1003. Function usart_receiver_timeout_enable .....	694
Table 3-1004. Function usart_receiver_timeout_disable .....	694
Table 3-1005. Function usart_receiver_timeout_threshold_config.....	695
Table 3-1006. Function usart_data_transmit .....	695
Table 3-1007. Function usart_data_receive .....	696
Table 3-1008. Function usart_mute_mode_enable.....	696
Table 3-1009. Function usart_mute_mode_disable .....	697
Table 3-1010. Function usart_mute_mode_wakeup_config .....	697
Table 3-1011. Function usart_lin_mode_enable .....	698
Table 3-1012. Function usart_lin_mode_disable .....	699
Table 3-1013. Function usart_lin_break_dection_length_config.....	699



Table 3-1014. Function <code>usart_halfduplex_enable</code> .....	700
Table 3-1015. Function <code>usart_halfduplex_disable</code> .....	700
Table 3-1016. Function <code>usart_synchronous_clock_enable</code> .....	701
Table 3-1017. Function <code>usart_synchronous_clock_disable</code> .....	701
Table 3-1018. Function <code>usart_synchronous_clock_config</code> .....	702
Table 3-1019. Function <code>usart_guard_time_config</code> .....	703
Table 3-1020. Function <code>usart_smartcard_mode_enable</code> .....	703
Table 3-1021. Function <code>usart_smartcard_mode_disable</code> .....	704
Table 3-1022. Function <code>usart_smartcard_mode_nack_enable</code> .....	704
Table 3-1023. Function <code>usart_smartcard_mode_nack_disable</code> .....	705
Table 3-1024. Function <code>usart_smartcard_autoretry_config</code> .....	705
Table 3-1025. Function <code>usart_block_length_config</code> .....	706
Table 3-1026. Function <code>usart_irda_mode_enable</code> .....	706
Table 3-1027. Function <code>usart_irda_mode_disable</code> .....	707
Table 3-1028. Function <code>usart_prescaler_config</code> .....	707
Table 3-1029. Function <code>usart_irda_lowpower_config</code> .....	708
Table 3-1030. Function <code>usart_dma_receive_config</code> .....	709
Table 3-1031. Function <code>usart_dma_transmit_config</code> .....	709
Table 3-1032. Function <code>usart_hardware_flow_rts_config</code> .....	710
Table 3-1033. Function <code>usart_hardware_flow_cts_config</code> .....	711
Table 3-1034. Function <code>usart_data_first_config</code> .....	711
Table 3-1035. Function <code>usart_invert_config</code> .....	712
Table 3-1036. Function <code>usart_address_config</code> .....	713
Table 3-1037. Function <code>usart_send_break</code> .....	713
Table 3-1038. Function <code>usart_flag_get</code> .....	714
Table 3-1039. Function <code>usart_flag_clear</code> .....	715
Table 3-1040. Function <code>usart_interrupt_enable</code> .....	715
Table 3-1041. Function <code>usart_interrupt_disable</code> .....	716
Table 3-1042. Function <code>usart_interrupt_flag_get</code> .....	717
Table 3-1043. Function <code>usart_interrupt_flag_clear</code> .....	718
Table 3-1044. Function <code>usart5_data_first_config</code> .....	719
Table 3-1045. Function <code>usart5_invert_config</code> .....	720
Table 3-1046. Function <code>usart5_overrun_enable</code> .....	721
Table 3-1047. Function <code>usart5_overrun_disable</code> .....	721
Table 3-1048. Function <code>usart5_address_config</code> .....	722
Table 3-1049. Function <code>usart5_address_detection_mode_config</code> .....	722
Table 3-1050. Function <code>usart5_smartcard_mode_early_nack_enable</code> .....	723
Table 3-1051. Function <code>usart5_smartcard_mode_early_nack_disable</code> .....	724
Table 3-1052. Function <code>usart5_reception_error_dma_enable</code> .....	724
Table 3-1053. Function <code>usart5_reception_error_dma_disable</code> .....	725
Table 3-1054. Function <code>usart5_wakeup_enable</code> .....	725
Table 3-1055. Function <code>usart5_wakeup_disable</code> .....	726
Table 3-1056. Function <code>usart5_wakeup_mode_config</code> .....	726
Table 3-1057. Function <code>usart5_receive_fifo_enable</code> .....	727

Table 3-1058. Function usart5_receive_fifo_disable.....	727
Table 3-1059. Function usart5_receive_fifo_counter_number.....	728
Table 3-1060. Function usart5_flag_get .....	728
Table 3-1061. Function usart5_flag_clear .....	729
Table 3-1062. Function usart5_interrupt_enable .....	730
Table 3-1063. Function usart5_interrupt_disable .....	731
Table 3-1064. Function usart5_command_enable .....	732
Table 3-1065. Function usart5_interrupt_flag_get.....	733
Table 3-1066. Function usart5_interrupt_flag_clear .....	734
Table 3-1067. WWDGT Registers .....	736
Table 3-1068. WWDGT firmware function .....	736
Table 3-1069. Function wwdgt_deinit .....	736
Table 3-1070. Function wwdgt_enable .....	737
Table 3-1071. Function wwdgt_counter_update .....	737
Table 3-1072. Function wwdgt_config .....	738
Table 3-1073. Function wwdgt_interrupt_enable.....	738
Table 3-1074. Function wwdgt_flag_get.....	739
Table 3-1075. Function wwdgt_flag_clear .....	740
Table 4-1. Revision history .....	741

## 1. Introduction

This manual introduces firmware library of GD32E50x 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 GD32E50x devices are involved in the package. The peripheral driving code and firmware examples on evaluation board are also included in firmware library. Users need not learn each peripherals in details and it's easy to apply a peripheral by using the firmware library. Using firmware library can greatly reduce programming time, thereby reducing development costs.

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

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

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

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

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

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

#### 1.1.1. Peripherals

Table 1-1. Peripherals

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

Peripherals	Descriptions
CMP	Comparator
DAC	Digital-to-analog converter
DBG	Debug
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
SHRTIMER	High-Precision Timer
I2C	Inter-integrated circuit interface
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
SQPI	Serial/Quad Parallel Interface
TIMER	TIMER
TMU	Trigonometric Math Unit
USART	Universal synchronous/asynchronous receiver /transmitter
WWDGT	Window watchdog timer
USB	Universal Serial Bus full-speed device interface
USBHS	Universal serial bus High-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 “gd32e50x\_”, such as: gd32e50x\_adc.h;
- The constants used only in one file should be defined in the used file; the constants used in many files should be defined in corresponding header file. All the constants are written in uppcase of English letters;
- Registers are handled as constants. The naming of them are written in uppcase of English letters. In most cases, register names are shortened accord with the user manual;
- Variables are written in lowercase, when concluded by several words, underlines should

be adapted among words;

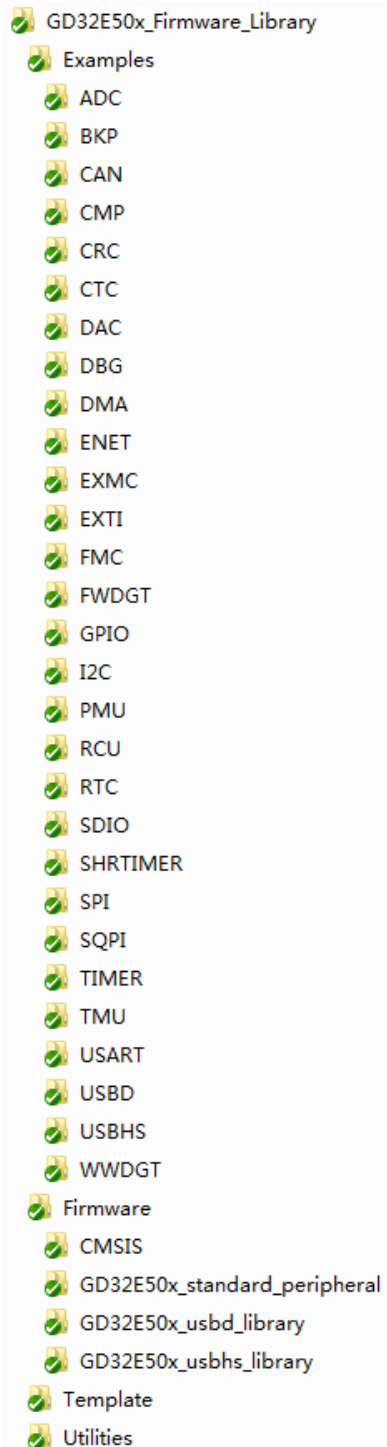
- The naming of peripheral functions are started with the peripheral abbreviation added with an underline, when the function name is concluded by several words, underlines should be adapted among words, and all the peripheral functions are written in lowercase.

## 2. Firmware Library Overview

### 2.1. File Structure of Firmware Library

GD32E50x\_Firmware\_Library, the file structure is shown as below:

**Figure 2-1. File structure of firmware library of GD32E50x**



### 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;
- `gd32e50x_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);
- `gd32e50x_it.c`: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- `gd32e50x_it.h`: the header file include all the prototypes of the interrupt service routines;
- `systick.c`: the source file include the precise time delay functions by using `systick`;
- `systick.h`: the header file include the prototype of the precise time delay functions by using `systick`;
- `main.c`: example code. Note: all the examples are not influenced by software IDEs.

### 2.1.2. Firmware Folder

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

- CMSIS subfolder includes the Cortex M33 kernel support files, the startup file based on the Cortex M33 kernel processor, the global header file of GD32E50x and system configuration file;
- GD32E50x\_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;
- GD32E50x\_usbd\_library subfolder includes all the related files about USB D peripheral:
  - Include subfolder includes the header files of USB D peripheral, users need not modify this folder;
  - Source subfolder includes the source files of USB D peripheral, users need not modify this folder;
- GD32E50x\_usbhs\_library subfolder includes all the related files about USB HS peripheral:
  - Include subfolder includes the header files of USB HS peripheral, users need not modify this folder;
  - Source subfolder includes the source files of USB HS peripheral, users need not modify this folder;

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

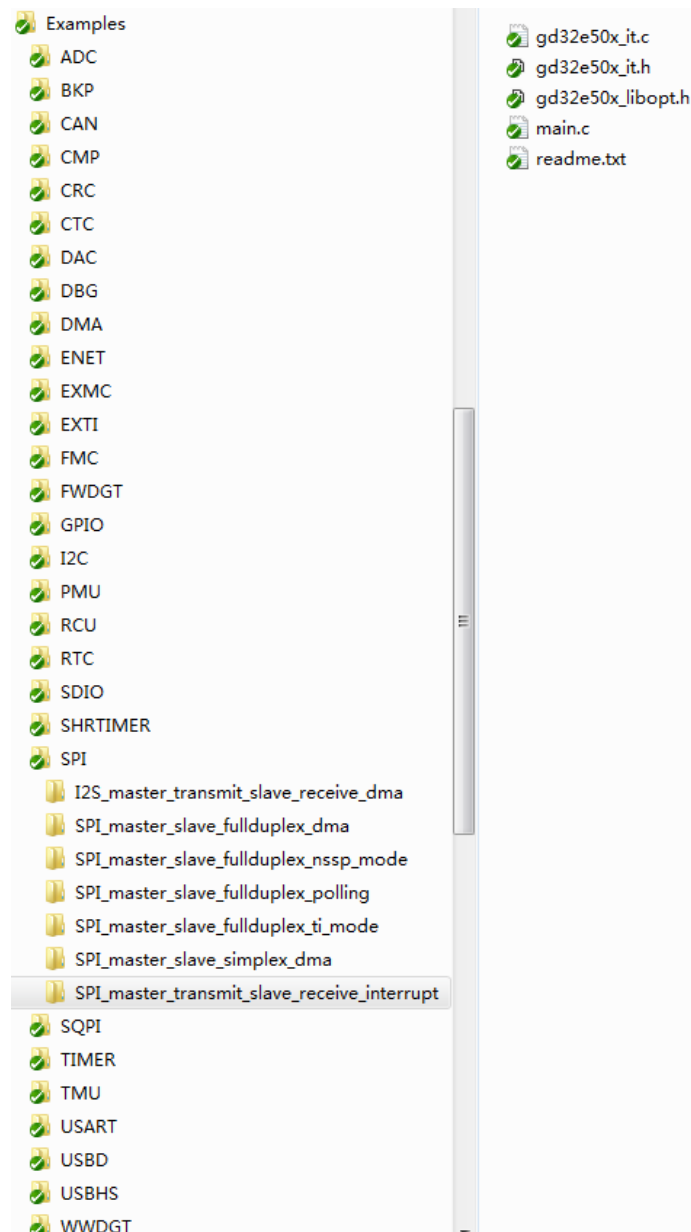
## 2.1.3. Template Folder

Template folder includes a simple demo of how to use LED, how to print by USART and use key to control, (IAR\_project is run in IAR, and Keil\_project is run in Keil5). 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**

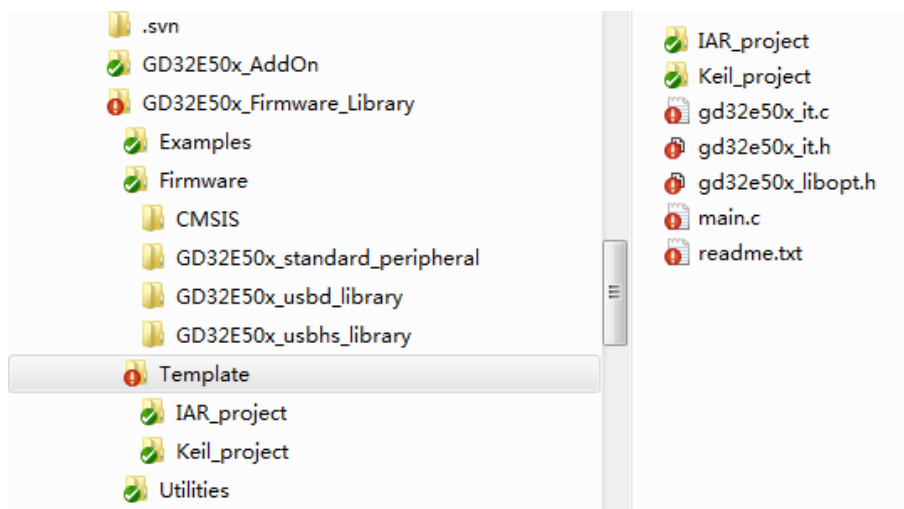




## Copy files

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:

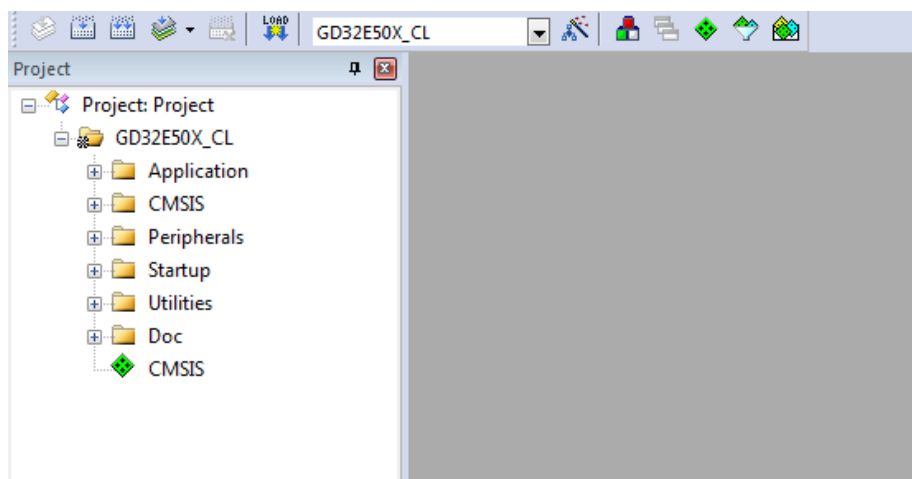
**Figure 2-3. Copy the peripheral example files**



## Open a project

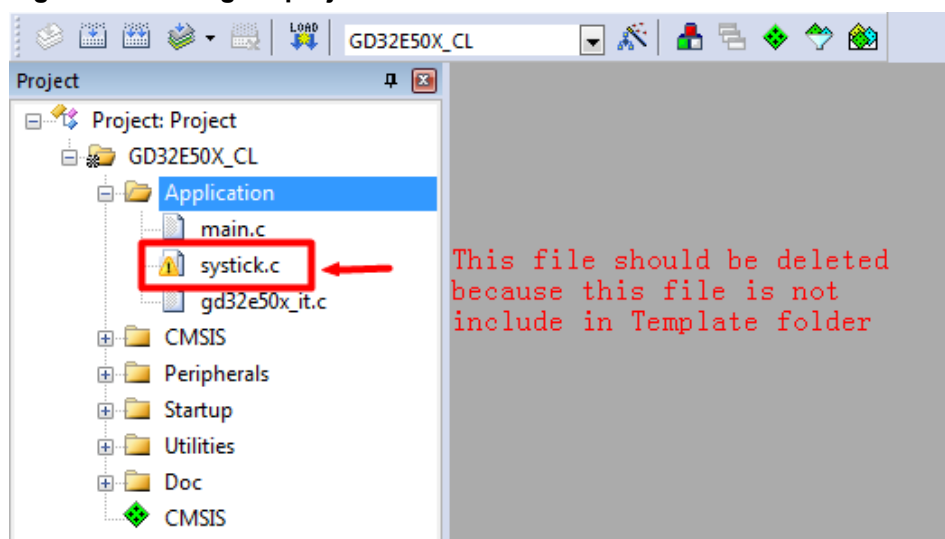
GD provides project in Keil and IAR, users can open project in different IDEs according to their need, such as "Keil\_project", open \Template\Keil\_project\Project.uvprojx, shown as below:

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



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

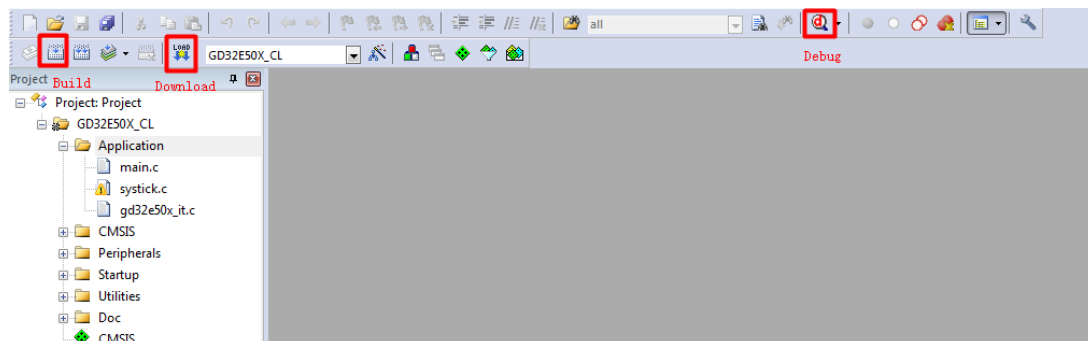
Figure 2-5. Configure project files



### Compile-Debug-Download

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

Figure 2-6. Compile-debug-download



#### 2.1.4. Utilities Folder

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

- LCD\_Commom subfolders include files for USB tests;
- gd32e50x\_eval.h and gd32e50x\_lcd\_eval.h are related header files of the evaluation board about running the firmware examples;
- gd32e50x\_eval.c and gd32e50x\_lcd\_eval.c are related source files of the evaluation board about running the firmware examples.

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

## 2.2. File descriptions of Firmware Library

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

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

Files	Descriptions
gd32e50x_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.
gd32e50x_it.h	Header file, including all the prototypes of interrupt service routines.
gd32e50x_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.
gd32e50x_xxx.h	The header file of peripheral PPP, including functions about peripheral PPP, and the variables used for functions.
gd32e50x_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_WDHT0	Watchdog 0 high threshold register

Registers	Descriptions
ADC_WDLT0	Watchdog 0 low threshold register
ADC_RSQ0	Regular sequence register 0
ADC_RSQ1	Regular sequence register 1
ADC_RSQ2	Regular sequence register 2
ADC_ISQ	Inserted sequence register
ADC_IDATAx (x=0..3)	Inserted data register x(x=0..3)
ADC_RDATA	Regular data register
ADC_OVSAMPCTL	Oversample control register
ADC_WD1SR	ADC watchdog 1 channel selection register
ADC_WD2SR	ADC watchdog 2 channel selection register
ADC_WDT1	ADC watchdog threshold register 1
ADC_WDT2	ADC watchdog threshold register 2
ADC_DIFCTL	ADC differential mode control register

### 3.2.2. Descriptions of Peripheral functions

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

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

Function name	Function description
adc_deinit	reset ADC peripheral
adc_enable	enable ADC interface
adc_disable	disable ADC interface
adc_calibration_enable	ADC calibration and reset calibration
adc_calibration_number	configure ADC calibration number
adc_dma_mode_enable	enable DMA request
adc_dma_mode_disable	disable DMA request
adc_tempsensor_vrefint_enable	enable the temperature sensor and vrefint channel
adc_tempsensor_vrefint_disable	disable the temperature sensor and vrefint channel
adc_discontinuous_mode_config	configure ADC discontinuous mode
adc_mode_config	configure the ADC sync mode
adc_special_function_config	enable or disable ADC special function
adc_data_alignment_config	configure ADCx data alignment
adc_channel_length_config	configure the length of regular channel group or inserted channel group
adc_regular_channel_config	configure ADC regular channel
adc_inserted_channel_config	configure ADC inserted channel
adc_inserted_channel_offset_config	configure ADC inserted channel offset
adc_channel_differential_mode_config	configure differential mode for ADC channel
adc_external_trigger_config	enable ADC external trigger

Function name	Function description
adc_external_trigger_source_config	configure ADC external trigger source
adc_software_trigger_enable	enable ADC software trigger
adc_regular_data_read	read ADC regular group data register
adc_inserted_data_read	read ADC inserted group data register
adc_sync_mode_convert_value_read	read the last ADC0 and ADC1 conversion result data in sync mode
adc_watchdog0_single_channel_enable	configure ADC analog watchdog 0 single channel
adc_watchdog0_group_channel_enable	configure ADC analog watchdog 0 group channel
adc_watchdog0_disable	disable ADC analog watchdog 0
adc_watchdog1_channel_config	configure ADC analog watchdog 1 channel
adc_watchdog2_channel_config	configure ADC analog watchdog 2 channel
adc_watchdog1_disable	disable ADC analog watchdog 1
adc_watchdog2_disable	disable ADC analog watchdog 2
adc_watchdog0_threshold_config	configure ADC analog watchdog 0 threshold
adc_watchdog1_threshold_config	configure ADC analog watchdog 1 threshold
adc_watchdog2_threshold_config	configure ADC analog watchdog 2 threshold
adc_resolution_config	configure ADC resolution
adc_oversample_mode_config	configure ADC oversample mode
adc_oversample_mode_enable	enable ADC oversample mode
adc_oversample_mode_disable	disable ADC oversample mode
adc_flag_get	get the ADC flag bits
adc_flag_clear	clear the ADC flag bits
adc_interrupt_enable	enable ADC interrupt
adc_interrupt_disable	disable ADC interrupt
adc_interrupt_flag_get	get the ADC interrupt bits
adc_interrupt_flag_clear	clear the ADC flag

## adc\_deinit

The description of adc\_deinit is shown as below:

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

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

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### adc\_enable

The description of adc\_enable is shown as below:

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

Function name	adc_enable
Function prototype	void adc_enable(uint32_t adc_periph);
Function descriptions	enable ADC 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 3-6. Function adc\_disable**

Function name	adc_disable
Function prototype	void adc_disable(uint32_t adc_periph);
Function descriptions	disable ADC 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:

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

### adc\_calibration\_enable

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

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

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

Example:

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

### adc\_calibration\_number

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

**Table 3-8. Function adc\_calibration\_number**

Function name	adc_calibration_number
Function prototype	void adc_calibration_number(uint32_t adc_periph, uint32_t clb_num);
Function descriptions	configure ADC calibration number
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection



Input parameter{in}	
<b>clb_num</b>	calibration number
<i>ADC_CALIBRATION_NUM1</i>	calibrate once
<i>ADC_CALIBRATION_NUM2</i>	calibrate twice
<i>ADC_CALIBRATION_NUM4</i>	calibrate 4 times
<i>ADC_CALIBRATION_NUM8</i>	calibrate 8 times
<i>ADC_CALIBRATION_NUM16</i>	calibrate 16 times
<i>ADC_CALIBRATION_NUM32</i>	calibrate 32 times
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC calibration number */
```

```
adc_calibration_number(ADC0, ADC_CALIBRATION_NUM16);
```

### adc\_dma\_mode\_enable

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

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

<b>Function name</b>	adc_dma_mode_enable
<b>Function prototype</b>	void adc_dma_mode_enable(uint32_t adc_periph);
<b>Function descriptions</b>	enable ADC DMA request
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
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 3-10. Function adc\_dma\_mode\_disable**

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

Example:

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

### adc\_tempsensor\_vrefint\_enable

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

**Table 3-11. Function adc\_tempsensor\_vrefint\_enable**

<b>Function name</b>	adc_tempsensor_vrefint_enable
<b>Function prototype</b>	void adc_tempsensor_vrefint_enable(void);
<b>Function descriptions</b>	enable the temperature sensor and vrefint channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the temperature sensor and vrefint channel */
adc_tempsensor_vrefint_enable();
```

## adc\_tempsensor\_vrefint\_disable

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

**Table 3-12. Function adc\_tempsensor\_vrefint\_disable**

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

Example:

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

```
adc_tempsensor_vrefint_disable();
```

## adc\_discontinuous\_mode\_config

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

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

<b>Function name</b>	adc_discontinuous_mode_config
<b>Function prototype</b>	void adc_discontinuous_mode_config(uint32_t adc_periph, uint8_t adc_channel_group, uint8_t length);
<b>Function descriptions</b>	configure ADC discontinuous mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<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_REGULAR_CHANNEL</i>	regular channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
<i>ADC_CHANNEL_DISCON_DISABLE</i>	disable discontinuous mode of regular and inserted channel
<b>Input parameter{in}</b>	

<b>length</b>	number of conversions in discontinuous mode, the number can be 1..8 for regular channel, the number has no effect for inserted channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### adc\_mode\_config

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

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

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

<i>ADC_DAUL_REGULAR_FOLLOWUP_SLOW</i>	ADC0 and ADC1 work in follow-up slow mode only
<i>ADC_DAUL_INSERTED_TRIGGERR_ROTATION</i>	ADC0 and ADC1 work in trigger rotation mode only
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### adc\_special\_function\_config

The description of `adc_special_function_config` is shown as below:

**Table 3-15. Function `adc_special_function_config`**

<b>Function name</b>	<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_CHANNEL_AUTO</i>	inserted channel group convert automatically
<i>ADC_CONTINUOUS_MODE</i>	continuous mode select
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### adc\_data\_alignment\_config

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

**Table 3-16. Function adc\_data\_alignment\_config**

<b>Function name</b>	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
ADCx(x=0..2)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>data_alignment</b>	data alignment select
ADC_DATAALIGN_RIGHT	right alignment
ADC_DATAALIGN_LEFT	left alignment
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### adc\_channel\_length\_config

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

**Table 3-17. Function adc\_channel\_length\_config**

<b>Function name</b>	adc_channel_length_config
<b>Function prototype</b>	void adc_channel_length_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t length);
<b>Function descriptions</b>	configure the length of regular channel group or inserted channel group
<b>Precondition</b>	-

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

Example:

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

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

### adc\_regular\_channel\_config

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

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

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

ADC_SAMPLETIME_1POINT5	1.5 cycles
ADC_SAMPLETIME_7POINT5	7.5 cycles
ADC_SAMPLETIME_13POINT5	13.5 cycles
ADC_SAMPLETIME_28POINT5	28.5 cycles
ADC_SAMPLETIME_41POINT5	41.5 cycles
ADC_SAMPLETIME_55POINT5	55.5 cycles
ADC_SAMPLETIME_71POINT5	71.5 cycles
ADC_SAMPLETIME_239POINT5	239.5 cycles
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

### adc\_inserted\_channel\_config

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

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

<b>Function name</b>	adc_inserted_channel_config
<b>Function prototype</b>	void adc_inserted_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
<b>Function descriptions</b>	configure ADC inserted channel
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>adc_periph</b>	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
<b>rank</b>	the inserted group sequencer rank, this parameter must be between 0 to 3
Input parameter{in}	
<b>adc_channel</b>	the selected ADC channel



<b>ADC_CHANNEL_x</b> (x=0..17)	ADC Channelx (x=0..17)(x=16 and x=17 are only for ADC0)
<b>Input parameter{in}</b>	
<b>sample_time</b>	the sample time value
<b>ADC_SAMPLETIME_1POINT5</b>	1.5 cycles
<b>ADC_SAMPLETIME_7POINT5</b>	7.5 cycles
<b>ADC_SAMPLETIME_13POINT5</b>	13.5 cycles
<b>ADC_SAMPLETIME_28POINT5</b>	28.5 cycles
<b>ADC_SAMPLETIME_41POINT5</b>	41.5 cycles
<b>ADC_SAMPLETIME_55POINT5</b>	55.5 cycles
<b>ADC_SAMPLETIME_71POINT5</b>	71.5 cycles
<b>ADC_SAMPLETIME_239POINT5</b>	239.5 cycles
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

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

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

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

Example:

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

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

### adc\_channel\_differential\_mode\_config

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

**Table 3-21. Function adc\_channel\_differential\_mode\_config**

<b>Function name</b>	adc_channel_differential_mode_config
<b>Function prototype</b>	void adc_channel_differential_mode_config(uint32_t adc_periph, uint32_t adc_channel, ControlStatus newvalue);
<b>Function descriptions</b>	configure differential mode for ADC channel
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
<b>inserted_channel</b>	the channel use differential mode
<i>ADC_DIFFERENTIAL_MODE_CHANNEL_x</i> (x=0..14), <i>ADC_DIFFERENTIAL_MODE_CHANNEL_ALL</i>	ADC channel for differential mode (just for channel0~channel14)
Input parameter{in}	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* configure differential mode for ADC channel */
```

```
adc_channel_differential_mode_config(ADC0,  
ADC_DIFFERENTIAL_MODE_CHANNEL_ALL, ENABLE);
```

### adc\_external\_trigger\_config

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

**Table 3-22. Function adc\_external\_trigger\_config**

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

Example:

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

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

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

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

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

<b>Function name</b>	<code>adc_external_trigger_source_config</code>
<b>Function prototype</b>	<code>void adc_external_trigger_source_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t external_trigger_source);</code>
<b>Function descriptions</b>	configure ADC external trigger source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<code>ADCx(x=0..2)</code>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_channel_group</b>	select the channel group
<code>ADC_REGULAR_CHANNEL</code>	regular channel group
<code>ADC_INSERTED_CHANNEL</code>	inserted channel group
<b>Input parameter{in}</b>	
<b>external_trigger_source</b>	regular or inserted group trigger source
<code>ADC0_1_EXTTRIG_REGULAR_T0_CH0</code>	TIMER0 CH0 event select for regular channel
<code>ADC0_1_EXTTRIG_REGULAR_T0_CH1</code>	TIMER0 CH1 event select for regular channel
<code>ADC0_1_EXTTRIG_REGULAR_T0_CH2</code>	TIMER0 CH2 event select for regular channel
<code>ADC0_1_EXTTRIG_REGULAR_T1_CH1</code>	TIMER1 CH1 event select for regular channel
<code>ADC0_1_EXTTRIG_REGULAR_T2_TRGO</code>	TIMER2 TRGO event select for regular channel
<code>ADC0_1_EXTTRIG_REGULAR_T3_CH3</code>	TIMER3 CH3 event select for regular channel
<code>ADC0_1_EXTTRIG_REGULAR_T7_TRGO</code>	TIMER7 TRGO event select for regular channel
<code>ADC0_1_EXTTRIG_REGULAR_EXTI_11</code>	external interrupt line 11 for regular channel
<code>ADC0_1_EXTTRIG_REGULAR_SHRTIMER_ADCTRG0</code>	SHRTIMER_ADCTRG0 output select for regular channel (for GD32E50X_HD and GD32E50X_CL devices)
<code>ADC0_1_EXTTRIG_REGULAR_SHRTIMER_ADCTRG2</code>	SHRTIMER_ADCTRG2 output select for regular channel (for GD32E50X_HD and GD32E50X_CL devices)
<code>ADC2_EXTTRIG_REGULAR_T2_CH0</code>	TIMER2 CH0 event select for regular channel

<i>ADC2_EXTTRIG_</i> <i>REGULAR_T1_CH2</i>	TIMER1 CH2 event select for regular channel
<i>ADC2_EXTTRIG_</i> <i>REGULAR_T0_CH2</i>	TIMER0 CH2 event select for regular channel
<i>ADC2_EXTTRIG_</i> <i>REGULAR_T7_CH0</i>	TIMER7 CH0 event select for regular channel
<i>ADC2_EXTTRIG_</i> <i>REGULAR_T7_TRGO</i>	TIMER7 TRGO event select for regular channel
<i>ADC2_EXTTRIG_</i> <i>REGULAR_T4_CH0</i>	TIMER4 CH0 event select for regular channel
<i>ADC2_EXTTRIG_</i> <i>REGULAR_T4_CH2</i>	TIMER4 CH2 event select for regular channel
<i>ADC0_1_2_EXTTRIG_</i> <i>REGULAR_NONE</i>	software trigger for regular channel
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_T0_TRGO</i>	TIMER0 TRGO event select for inserted channel
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_T0_CH3</i>	TIMER0 CH3 event select for inserted channel
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_T1_TRGO</i>	TIMER1 TRGO event select for inserted channel
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_T1_CH0</i>	TIMER1 CH0 event select for inserted channel
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_T2_CH3</i>	TIMER2 CH3 event select for inserted channel
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_T3_TRGO</i>	TIMER3 TRGO event select for inserted channel
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_T7_CH3</i>	TIMER7 CH3 event select for inserted channel
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_EXTI_15</i>	external interrupt line 15 for inserted channel
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_SHRTIME</i> <i>R_ADCTRG1</i>	SHRTIMER_ADCTRG1 output select for inserted channel (for GD32E50X_HD and GD32E50X_CL devices)
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_SHRTIME</i> <i>R_ADCTRG3</i>	SHRTIMER_ADCTRG3 output select for inserted channel (for GD32E50X_HD and GD32E50X_CL devices)
<i>ADC2_EXTTRIG_</i> <i>INSERTED_T0_TRGO</i>	TIMER0 TRGO event select for inserted channel
<i>ADC2_EXTTRIG_</i> <i>INSERTED_T0_CH3</i>	TIMER0 CH3 event select for inserted channel
<i>ADC2_EXTTRIG_</i> <i>INSERTED_T3_CH2</i>	TIMER3 CH2 event select for inserted channel
<i>ADC2_EXTTRIG_</i>	TIMER7 CH1 event select for inserted channel

<i>INSERTED_T7_CH1</i>	
<i>ADC2_EXTTRIG_INSERTED_T7_CH3</i>	TIMER7 CH3 event select for inserted channel
<i>ADC2_EXTTRIG_INSERTED_T4_TRGO</i>	TIMER4 TRGO event select for inserted channel
<i>ADC2_EXTTRIG_INSERTED_T4_CH3</i>	TIMER4 CH3 event select for inserted channel
<i>ADC0_1_2_EXTTRIG_INSERTED_NONE</i>	software trigger for inserted channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
adc_external_trigger_source_config(ADC0,ADC_REGULAR_CHANNEL,  
ADC0_1_EXTTRIG_REGULAR_T0_CH0);
```

### adc\_software\_trigger\_enable

The description of `adc_software_trigger_enable` is shown as below:

**Table 3-24. 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_REGULAR_CHANNEL</i>	regular channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

adc_software_trigger_enable(ADC0, ADC_REGULAR_CHANNEL);
```

### adc\_regular\_data\_read

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

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

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

Example:

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

uint16_t adc_value = 0;

adc_value = adc_regular_data_read(ADC0);
```

### adc\_inserted\_data\_read

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

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

The description of `adc_sync_mode_convert_value_read` is shown as below:

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

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

Example:

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

```
uint32_t adc_value = 0;
```

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

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

The description of `adc_watchdog0_single_channel_enable` is shown as below:

**Table 3-28. Function `adc_watchdog0_single_channel_enable`**

<b>Function name</b>	<code>adc_watchdog0_single_channel_enable</code>
<b>Function prototype</b>	<code>void adc_watchdog0_single_channel_enable(uint32_t adc_periph, uint8_t adc_channel);</code>
<b>Function descriptions</b>	configure ADC analog watchdog 0 single channel



<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_channel</b>	the selected ADC channel
<i>ADC_CHANNEL_x</i> (x=0..17)	ADC channelx(x=0..17) (x=16 and x=17 are only for ADC0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC0 analog watchdog 0 single channel */
```

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

### adc\_watchdog0\_group\_channel\_enable

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

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

<b>Function name</b>	adc_watchdog0_group_channel_enable
<b>Function prototype</b>	void adc_watchdog0_group_channel_enable(uint32_t adc_periph, uint8_t adc_channel_group);
<b>Function descriptions</b>	configure ADC analog watchdog 0 group 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>adc_channel_group</b>	the channel group use analog watchdog
<i>ADC_REGULAR_CHANNEL</i>	regular channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
<i>ADC_REGULAR_INSERTED_CHANNEL</i>	both regular and inserted group
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* configure ADC0 analog watchdog 0 group channel */

adc_watchdog0_group_channel_enable(ADC0, ADC_REGULAR_CHANNEL);
```

### adc\_watchdog0\_disable

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

**Table 3-30. Function adc\_watchdog0\_disable**

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

Example:

```
/* disable ADC0 analog watchdog 0 */

adc_watchdog0_disable(ADC0);
```

### adc\_watchdog1\_channel\_config

The description of adc\_watchdog1\_channel\_config is shown as below:

**Table 3-31. Function adc\_watchdog1\_channel\_config**

<b>Function name</b>	adc_watchdog1_channel_config
<b>Function prototype</b>	void adc_watchdog1_channel_config(uint32_t adc_periph, uint32_t adc_channel, ControlStatus newvalue);
<b>Function descriptions</b>	configure ADC analog watchdog 1 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>adc_channel</b>	the selected ADC channel

ADC_AWD1_2_SELECTION_CHANNEL_x (x=0..17), ADC_AWD1_2_SELECTION_CHANNEL_ALL	ADC channel analog watchdog 1/2 selection (x=0..17, x=16 and x=17 are only for 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:

```
/* configure ADC analog watchdog 1 channel */
```

```
adc_watchdog1_channel_config(ADC0, ADC_AWD1_2_SELECTION_CHANNEL_1,  
ENABLE);
```

### adc\_watchdog2\_channel\_config

The description of adc\_watchdog2\_channel\_config is shown as below:

**Table 3-32. Function adc\_watchdog2\_channel\_config**

<b>Function name</b>	adc_watchdog2_channel_config
<b>Function prototype</b>	void adc_watchdog2_channel_config(uint32_t adc_periph, uint32_t adc_channel, ControlStatus newvalue);
<b>Function descriptions</b>	configure ADC analog watchdog 2 channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_channel</b>	the selected ADC channel
ADC_AWD1_2_SELECTION_CHANNEL_x (x=0..17), ADC_AWD1_2_SELECTION_CHANNEL_ALL	ADC channel analog watchdog 1/2 selection (x=0..17, x=16 and x=17 are only for ADC0)
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
ENABLE	enable function

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

Example:

```
/* configure ADC analog watchdog 2 channel */
```

```
adc_watchdog2_channel_config(ADC0, ADC_AWD1_2_SELECTION_CHANNEL_1,
ENABLE);
```

### adc\_watchdog1\_disable

The description of adc\_watchdog1\_disable is shown as below:

**Table 3-33. Function adc\_watchdog1\_disable**

<b>Function name</b>	adc_watchdog1_disable
<b>Function prototype</b>	void adc_watchdog1_disable(uint32_t adc_periph);
<b>Function descriptions</b>	disable ADC analog watchdog 1
<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 analog watchdog 1 */
```

```
adc_watchdog1_disable(ADC0);
```

### adc\_watchdog2\_disable

The description of adc\_watchdog2\_disable is shown as below:

**Table 3-34. Function adc\_watchdog2\_disable**

<b>Function name</b>	adc_watchdog2_disable
<b>Function prototype</b>	void adc_watchdog2_disable(uint32_t adc_periph);
<b>Function descriptions</b>	disable ADC analog watchdog 2
<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 analog watchdog 2 */
```

```
adc_watchdog2_disable(ADC0);
```

### adc\_watchdog0\_threshold\_config

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

**Table 3-35. Function adc\_watchdog0\_threshold\_config**

<b>Function name</b>	adc_watchdog0_threshold_config
<b>Function prototype</b>	void adc_watchdog0_threshold_config(uint32_t adc_periph , uint16_t low_threshold , uint16_t high_threshold);
<b>Function descriptions</b>	configure ADC analog watchdog 0 threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>low_threshold</b>	analog watchdog low threshold, 0..4095
<b>Input parameter{in}</b>	
<b>high_threshold</b>	analog watchdog high threshold, 0..4095
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC0 analog watchdog 0 threshold */
```

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

### adc\_watchdog1\_threshold\_config

The description of adc\_watchdog1\_threshold\_config is shown as below:

**Table 3-36. Function adc\_watchdog1\_threshold\_config**

<b>Function name</b>	adc_watchdog1_threshold_config
----------------------	--------------------------------

<b>Function prototype</b>	void adc_watchdog1_threshold_config(uint32_t adc_periph , uint8_t low_threshold , uint8_t high_threshold);
<b>Function descriptions</b>	configure ADC analog watchdog 1 threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<b>ADCx(x=0..2)</b>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>low_threshold</b>	analog watchdog low threshold, 0..255
<b>Input parameter{in}</b>	
<b>high_threshold</b>	analog watchdog high threshold, 0..255
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC0 analog watchdog 1 threshold */
```

```
adc_watchdog1_threshold_config(ADC0, 0x40, 0xA0);
```

## adc\_watchdog2\_threshold\_config

The description of adc\_watchdog2\_threshold\_config is shown as below:

**Table 3-37. Function adc\_watchdog2\_threshold\_config**

<b>Function name</b>	adc_watchdog2_threshold_config
<b>Function prototype</b>	void adc_watchdog2_threshold_config(uint32_t adc_periph , uint8_t low_threshold , uint8_t high_threshold);
<b>Function descriptions</b>	configure ADC analog watchdog 2 threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<b>ADCx(x=0..2)</b>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>low_threshold</b>	analog watchdog low threshold, 0..255
<b>Input parameter{in}</b>	
<b>high_threshold</b>	analog watchdog high threshold, 0..255
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC0 analog watchdog 2 threshold */

adc_watchdog2_threshold_config(ADC0, 0x40, 0xA0);
```

### adc\_resolution\_config

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

**Table 3-38. 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 data alignment */

adc_data_alignment_config(ADC0, ADC_DATAALIGN_RIGHT);
```

### adc\_oversample\_mode\_config

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

**Table 3-39. 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
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>mode</b>	ADC oversampling mode
<i>ADC_OVERSAMPLING_ALL_CONVERT</i>	all oversampled conversions for a channel are done consecutively after a trigger
<i>ADC_OVERSAMPLING_ONE_CONVERT</i>	each oversampled conversion for a channel needs a trigger
<b>Input parameter{in}</b>	
<b>shift</b>	ADC oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_NONE</i>	no oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_1B</i>	1-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_2B</i>	2-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_3B</i>	3-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_4B</i>	4-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_5B</i>	5-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_6B</i>	6-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_7B</i>	7-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_8B</i>	8-bit oversampling shift
<b>Input parameter{in}</b>	
<b>ratio</b>	ADC oversampling ratio
<i>ADC_OVERSAMPLING_RATIO_MUL2</i>	oversampling ratio multiple 2
<i>ADC_OVERSAMPLING_RATIO_MUL4</i>	oversampling ratio multiple 4
<i>ADC_OVERSAMPLING_RATIO_MUL8</i>	oversampling ratio multiple 8
<i>ADC_OVERSAMPLING_RATIO_MUL16</i>	oversampling ratio multiple 16



<code>_RATIO_MUL16</code>	
<code>ADC_OVERSAMPLING_RATIO_MUL32</code>	oversampling ratio multiple 32
<code>ADC_OVERSAMPLING_RATIO_MUL64</code>	oversampling ratio multiple 64
<code>ADC_OVERSAMPLING_RATIO_MUL128</code>	oversampling ratio multiple 128
<code>ADC_OVERSAMPLING_RATIO_MUL256</code>	oversampling ratio multiple 256
<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 3-40. Function `adc_oversample_mode_enable`**

<b>Function name</b>	<code>adc_oversample_mode_enable</code>
<b>Function prototype</b>	<code>void adc_oversample_mode_enable(uint32_t adc_periph);</code>
<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
<code>ADCx(x=0..2)</code>	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 3-41. Function adc\_oversample\_mode\_disable**

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

Example:

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

## adc\_flag\_get

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

**Table 3-42. Function adc\_flag\_get**

<b>Function name</b>	adc_flag_get
<b>Function prototype</b>	FlagStatus adc_flag_get(uint32_t adc_periph, uint32_t flag);
<b>Function descriptions</b>	get the ADC flag
<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>flag</b>	the adc flag bits
<i>ADC_FLAG_WDE0</i>	analog watchdog 0 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 regular channel group
<i>ADC_FLAG_WDE1</i>	analog watchdog 1 event flag
<i>ADC_FLAG_WDE2</i>	analog watchdog 2 event flag

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

Example:

```
/* get the ADC0 analog watchdog 0 flag bits */
```

```
FlagStatus flag_value;
```

```
flag_value = adc_flag_get(ADC0, ADC_FLAG_WDE0);
```

### adc\_flag\_clear

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

**Table 3-43. Function adc\_flag\_clear**

Function name	adc_flag_clear
Function prototype	void adc_flag_clear(uint32_t adc_periph, uint32_t 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}	
flag	the adc flag bits
ADC_FLAG_WDE0	analog watchdog 0 event flag
ADC_FLAG_EOC	end of group conversion flag
ADC_FLAG_EOIC	end of inserted group conversion flag
ADC_FLAG_STIC	start flag of inserted channel group
ADC_FLAG_STRC	start flag of regular channel group
ADC_FLAG_WDE1	analog watchdog 1 event flag
ADC_FLAG_WDE2	analog watchdog 2 event flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the ADC0 analog watchdog 0 flag bits */
```

```
adc_flag_clear(ADC0, ADC_FLAG_WDE0);
```

## adc\_interrupt\_enable

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

**Table 3-44. Function adc\_interrupt\_enable**

<b>Function name</b>	adc_interrupt_enable
<b>Function prototype</b>	void adc_interrupt_enable(uint32_t adc_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable ADC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>interrupt</b>	the adc interrupt
ADC_INT_WDE0	analog watchdog 0 interrupt
ADC_INT_EOC	end of group conversion interrupt
ADC_INT_EOIC	end of inserted group conversion interrupt
ADC_INT_WDE1	analog watchdog 1 interrupt
ADC_INT_WDE2	analog watchdog 2 interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC0 analog watchdog 0 interrupt */
adc_interrupt_enable(ADC0, ADC_INT_WDE0);
```

## adc\_interrupt\_disable

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

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

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

<i>ADC_INT_WDE0</i>	analog watchdog 0 interrupt
<i>ADC_INT_EOC</i>	end of group conversion interrupt
<i>ADC_INT_EOIC</i>	end of inserted group conversion interrupt
<i>ADC_INT_WDE1</i>	analog watchdog 1 interrupt
<i>ADC_INT_WDE2</i>	analog watchdog 2 interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable ADC0 analog watchdog 0 interrupt */
adc_interrupt_disable(ADC0, ADC_INT_WDE0);
```

### adc\_interrupt\_flag\_get

The description of `adc_interrupt_flag_get` is shown as below:

**Table 3-46. Function `adc_interrupt_flag_get`**

<b>Function name</b>	<code>adc_interrupt_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus adc_interrupt_flag_get(uint32_t adc_periph, uint32_t int_flag);</code>
<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>int_flag</b>	the adc interrupt bits
<i>ADC_INT_FLAG_WDE0</i>	analog watchdog 0 interrupt
<i>ADC_INT_FLAG_EOC</i>	end of group conversion interrupt
<i>ADC_INT_FLAG_EOIC</i>	end of inserted group conversion interrupt
<i>ADC_INT_FLAG_WDE1</i>	analog watchdog 1 interrupt
<i>ADC_INT_FLAG_WDE2</i>	analog watchdog 2 interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the ADC0 analog watchdog 0 interrupt bits*/
```

```
FlagStatus flag_value;
```

```
flag_value = adc_interrupt_flag_get(ADC0, ADC_INT_WDE0);
```

### adc\_interrupt\_flag\_clear

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

**Table 3-47. Function adc\_interrupt\_flag\_clear**

<b>Function name</b>	adc_interrupt_flag_clear
<b>Function prototype</b>	void adc_interrupt_flag_clear(uint32_t adc_periph, uint32_t int_flag);
<b>Function descriptions</b>	clear the ADC interrupt bits
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>int_flag</b>	the adc interrupt bits
ADC_INT_FLAG_WDE0	analog watchdog 0 interrupt
ADC_INT_FLAG_EOC	end of group conversion interrupt
ADC_INT_FLAG_EOIC	end of inserted group conversion interrupt
ADC_INT_FLAG_WDE1	analog watchdog 1 interrupt
ADC_INT_FLAG_WDE2	analog watchdog 2 interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the ADC0 analog watchdog 0 interrupt bits*/
```

```
adc_interrupt_flag_clear(ADC0, ADC_INT_WDE0);
```

## 3.3. BKP

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

functions are introduced in chapter [3.3.2](#).

### 3.3.1. Descriptions of Peripheral registers

BKP registers are listed in the table shown as below:

**Table 3-48. BKP Registers**

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

### 3.3.2. Descriptions of Peripheral functions

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

**Table 3-49. BKP firmware function**

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

## Enum bkp\_data\_register\_enum

**Table 3-50. Enum bkp\_data\_register\_enum**

Member name	Function description
BKP_DATA_0	bkp data register number 0
BKP_DATA_1	bkp data register number 1
BKP_DATA_2	bkp data register number 2
BKP_DATA_3	bkp data register number 3
BKP_DATA_4	bkp data register number 4
BKP_DATA_5	bkp data register number 5
BKP_DATA_6	bkp data register number 6
BKP_DATA_7	bkp data register number 7
BKP_DATA_8	bkp data register number 8
BKP_DATA_9	bkp data register number 9
BKP_DATA_10	bkp data register number 10
BKP_DATA_11	bkp data register number 11
BKP_DATA_12	bkp data register number 12
BKP_DATA_13	bkp data register number 13
BKP_DATA_14	bkp data register number 14
BKP_DATA_15	bkp data register number 15
BKP_DATA_16	bkp data register number 16
BKP_DATA_17	bkp data register number 17
BKP_DATA_18	bkp data register number 18
BKP_DATA_19	bkp data register number 19
BKP_DATA_20	bkp data register number 20
BKP_DATA_21	bkp data register number 21
BKP_DATA_22	bkp data register number 22
BKP_DATA_23	bkp data register number 23
BKP_DATA_24	bkp data register number 24
BKP_DATA_25	bkp data register number 25
BKP_DATA_26	bkp data register number 26
BKP_DATA_27	bkp data register number 27
BKP_DATA_28	bkp data register number 28
BKP_DATA_29	bkp data register number 29
BKP_DATA_30	bkp data register number 30
BKP_DATA_31	bkp data register number 31
BKP_DATA_32	bkp data register number 32
BKP_DATA_33	bkp data register number 33
BKP_DATA_34	bkp data register number 34
BKP_DATA_35	bkp data register number 35
BKP_DATA_36	bkp data register number 36
BKP_DATA_37	bkp data register number 37
BKP_DATA_38	bkp data register number 38



Member name	Function description
BKP_DATA_39	bkp data register number 39
BKP_DATA_40	bkp data register number 40
BKP_DATA_41	bkp data register number 41

## bkp\_deinit

The description of bkp\_deinit is shown as below:

**Table 3-51. Function bkp\_deinit**

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

Example:

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

```
bkp_deinit ();
```

## bkp\_write\_data

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

**Table 3-52. Function bkp\_write\_data**

Function name	bkp_write_data
Function prototype	void bkp_write_data(bkp_data_register_enum register_number, uint16_t data);
Function descriptions	write BKP data register
Precondition	-
The called functions	-
Input parameter{in}	
register_number	refer to <a href="#">Table 3-50. Enum bkp_data_register_enum</a>
BKP_DATA_x(x = 0..41)	bkp data register number x
Input parameter{in}	
data	the data to be write in BKP data register
0-0xffff	data value

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

## bkp\_read\_data

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

**Table 3-53. Function bkp\_data\_read**

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

Example:

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

## bkp\_rtc\_calibration\_output\_enable

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

**Table 3-54. Function bkp\_rtc\_calibration\_output\_enable**

Function name	bkp_rtc_calibration_output_enable
Function prototype	void bkp_rtc_calibration_output_enable(void);
Function descriptions	enable RTC clock calibration output
Precondition	-
The called functions	-

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

Example:

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

```
bkp_rtc_calibration_output_enable();
```

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

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

**Table 3-55. Function bkp\_rtc\_calibration\_output\_disable**

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

Example:

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

```
bkp_rtc_calibration_output_disable();
```

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

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

**Table 3-56. Function bkp\_rtc\_signal\_output\_enable**

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

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
bkp_rtc_signal_output_enable();
```

## bkp\_rtc\_signal\_output\_disable

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

**Table 3-57. Function bkp\_rtc\_signal\_output\_disable**

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

Example:

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

```
bkp_rtc_signal_output_disable();
```

## bkp\_rtc\_output\_select

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

**Table 3-58. Function bkp\_rtc\_output\_select**

Function name	bkp_rtc_output_select
Function prototype	void bkp_rtc_output_select (uint16_t outputsel);
Function descriptions	select RTC output, the RTC output can be select as alarm pulse or second pulse
Precondition	-
The called functions	-
Input parameter{in}	
<b>outputsel</b>	RTC output selection
<i>RTC_OUTPUT_ALAR</i>	RTC alarm pulse is selected as the RTC output

<i>M_PULSE</i>	
<i>RTC_OUTPUT_SECONDPULSE</i>	RTC second pulse is selected as the RTC output
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
bkp_rtc_output_select (RTC_OUTPUT_ALARM_PULSE);
```

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

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

**Table 3-59. Function bkp\_rtc\_clock\_output\_select**

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

Example:

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

```
bkp_rtc_clock_output_select (RTC_CLOCK_DIV_64);
```

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

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

**Table 3-60. Function bkp\_rtc\_clock\_calibration\_direction**

<b>Function name</b>	bkp_rtc_clock_calibration_direction
<b>Function prototype</b>	void bkp_rtc_clock_calibration_direction(uint16_t direction);
<b>Function descriptions</b>	select RTC clock calibration direction, the RTC clock calibration direction

	can be select as slowed down or speed up
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>direction</b>	RTC clock calibration direction
<i>RTC_CLOCK_SLOWED_DOWN</i>	RTC clock slowed down
<i>RTC_CLOCK_SPEED_UP</i>	RTC clock speed up
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
bkp_rtc_clock_calibration_direction(RTC_CLOCK_SLOWED_DOWN);
```

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

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

**Table 3-61. Function bkp\_rtc\_calibration\_value\_set**

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

Example:

```
/* set RTC clock calibration value */
```

```
bkp_rtc_calibration_value_set(0x7f);
```

### bkp\_tamper\_detection\_enable

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

**Table 3-62. Function bkp\_tamper\_detection\_enable**

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

Example:

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

### bkp\_tamper\_detection\_disable

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

**Table 3-63. Function bkp\_tamper\_detection\_disable**

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

Example:

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

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

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

**Table 3-64. Function bkp\_tamper\_active\_level\_set**

<b>Function name</b>	bkp_tamper_active_level_set
----------------------	-----------------------------

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

Example:

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

```
bkp_tamper_active_level_set (TAMPER_PIN_ACTIVE_HIGH);
```

### bkp\_tamper\_interrupt\_enable

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

**Table 3-65. Function bkp\_tamper\_interrupt\_enable**

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

Example:

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

```
bkp_tamper_interrupt_enable ();
```

### bkp\_tamper\_interrupt\_disable

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



Table 3-66. Function bkp\_tamper\_interrupt\_disable

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

Example:

```
/* disable tamper pin interrupt */
bkp_tamper_interrupt_disable ();
```

### bkp\_flag\_get

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

Table 3-67. Function bkp\_flag\_get

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

Example:

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

### bkp\_flag\_clear

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

Table 3-68. Function bkp\_flag\_clear

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

Example:

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

```
bkp_flag_clear (BKP_FLAG_TAMPER);
```

### bkp\_interrupt\_flag\_get

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

Table 3-69. Function bkp\_interrupt\_flag\_get

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

Example:

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

```
bkp_interrupt_flag_get (BKP_INT_FLAG_TAMPER);
```

## bkp\_interrupt\_flag\_clear

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

**Table 3-70. Function bkp\_interrupt\_flag\_clear**

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

Example:

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

```
bkp_interrupt_flag_clear (BKP_INT_FLAG_TAMPER);
```

## 3.4. CAN

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

### 3.4.1. Descriptions of Peripheral registers

CAN registers are listed in the table shown as below:

**Table 3-71. CAN Registers**

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

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

**Table 3-72. CAN-FD Registers**

Registers	Descriptions
CAN_CTL	Control register
CAN_STAT	Status register
CAN_TSTAT	Transmit status register
CAN_RFIFO0	Receive message FIFO0 register
CAN_RFIFO1	Receive message FIFO1 register
CAN_INTEN	Interrupt enable register
CAN_ERR	Error register
CAN_BT	Bit timing register
CAN_FDCTL	FD control register
CAN_FDSTAT	FD status register
CAN_FDTDC	FD transmitter delay compensation register
CAN_DBT	Date Bit timing register
CAN_TMIx	Transmit mailbox identifier register
CAN_TMPx	Transmit mailbox property register
CAN_TMDATA0x	Transmit mailbox data0 register
CAN_TMDATA1x	Transmit mailbox data1 register
CAN_RFIFOMIx	Receive FIFO mailbox identifier register
CAN_RFIFOMPx	Receive FIFO mailbox property register
CAN_RFIFOMDAT A0x	Receive FIFO mailbox data0 register
CAN_RFIFOMDAT A1x	Receive FIFO mailbox data1 register
CAN_FCTL	Filter control register

Registers	Descriptions
CAN_FMCFG	Filter mode configuration register
CAN_FSCFG	Filter scale configuration register
CAN_FAFIFO	Filter associated FIFO register
CAN_FW	Filter working register
CAN_FxDATAy	Filter x data y register

### 3.4.2. Descriptions of Peripheral functions

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

**Table 3-73. CAN firmware function**

Function name	Function description
can_deinit	deinitialize CAN
can_init	initialize CAN
can_filter_init	initialize CAN filter
can_filter_mask_mode_init	CAN filter mask mode initialization
can_struct_para_init	initialize CAN parameter struct with a default value
can_monitor_mode_set	CAN communication mode configure
can_fd_init	initialize CAN FD function
can_fd_function_enable	CAN FD frame function enable
can_fd_function_disable	CAN FD frame function disable
can1_filter_start_bank	set can1 filter start bank number
can_debug_freeze_enable	CAN debug freeze enable
can_debug_freeze_disable	CAN debug freeze disable
can_time_trigger_mode_enable	CAN time trigger mode enable
can_time_trigger_mode_disable	CAN time trigger mode disable
can_message_transmit	transmit CAN message
can_transmit_states	get CAN transmit state
can_transmission_stop	stop CAN transmission
can_message_receive	CAN receive message
can_fifo_release	CAN release fifo
can_receive_message_length_get	CAN receive message length
can_working_mode_set	CAN working mode
can_wakeup	CAN wakeup from sleep mode
can_error_get	CAN get error
can_receive_error_number_get	get CAN receive error number
can_transmit_error_number_get	get CAN transmit error number
can_interrupt_enable	CAN interrupt enable
can_interrupt_disable	CAN interrupt disable
can_flag_get	CAN get flag state
can_flag_clear	CAN clear flag state
can_interrupt_flag_get	CAN get interrupt flag state

Function name	Function description
can_interrupt_flag_clear	CAN clear interrupt flag state

## Structure can\_parameter\_struct

**Table 3-74. 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 3-75. 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

**Table 3-76. can\_transmit\_message\_struct (for CAN-FD)**

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

## Structure can\_receive\_message\_struct

**Table 3-77. 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

**Table 3-78. can\_receive\_message\_struct (for CAN-FD)**

Member name	Function description
rx_sfid	standard format frame identifier
rx_efid	extended format frame identifier
rx_ff	format of frame, standard or extended format
rx_ft	type of frame, data or remote
rx_dlen	data length
rx_data[64]	receive data
rx_fi	filtering index
fd_flag	CAN FD frame flag
fd_brs	bit rate of data switch
fd_esi	error status indicator

## Structure can\_fd\_tdc\_struct

**Table 3-79. can\_fd\_tdc\_struct (for CAN-FD)**

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

## Structure can\_fdframe\_struct

**Table 3-80. can\_fdframe\_struct (for CAN-FD)**

Member name	Function description
fd_frame	FD operation function
excp_event_detect	protocol exception event detection function
delay_compensation	transmitter delay compensation mode
p_delay_compensation	pointer to the struct of the transmitter delay compensation, refer to <a href="#">Table 3-79. can_fd_tdc_struct (for CAN-FD)</a> .
iso_bosch	ISO/Bosch mode choice
esi_mode	error state indicator mode

data_resync_jump_width	CAN resynchronization jump width
data_time_segment_1	time segment 1
data_time_segment_2	time segment 2
data_prescaler	baudrate prescaler

## Structure can\_filter\_parameter\_struct

**Table 3-81. 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

## Enum can\_interrupt\_flag\_enum

**Table 3-82. Enum can\_interrupt\_flag\_enum**

enum name	enum description
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_RFL0	receive FIFO0 not empty interrupt flag
CAN_INT_FLAG_RFO1	receive FIFO1 overfull interrupt flag
CAN_INT_FLAG_RFF1	receive FIFO1 full interrupt flag
CAN_INT_FLAG_RFL1	receive FIFO1 not empty interrupt flag
CAN_INT_FLAG_ERRN	error number interrupt flag
CAN_INT_FLAG_BOERR	bus-off error interrupt flag
CAN_INT_FLAG_PERR	passive error interrupt flag
CAN_INT_FLAG_WERR	warning error interrupt flag



## Enum can\_flag\_enum

**Table 3-83. Enum can\_flag\_enum**

enum name	enum description
CAN_FLAG_RXL	CAN flags
CAN_FLAG_LASTRX	RX level
CAN_FLAG_RS	last sample value of RX pin
CAN_FLAG_TS	receiving state
CAN_FLAG_SLPIF	transmitting state
CAN_FLAG_WUIF	status change flag of entering sleep working mode
CAN_FLAG_ERRIF	status change flag of wakeup from sleep working mode
CAN_FLAG_SLPWS	error flag
CAN_FLAG_IWS	sleep working state
CAN_FLAG_TMLS2	transmit mailbox 2 last sending in Tx FIFO
CAN_FLAG_TMLS1	transmit mailbox 1 last sending in Tx FIFO
CAN_FLAG_TMLS0	transmit mailbox 0 last sending in Tx FIFO
CAN_FLAG_TME2	transmit mailbox 2 empty
CAN_FLAG_TME1	transmit mailbox 1 empty
CAN_FLAG_TME0	transmit mailbox 0 empty
CAN_FLAG_MTE2	mailbox 2 transmit error
CAN_FLAG_MTE1	mailbox 1 transmit error
CAN_FLAG_MTE0	mailbox 0 transmit error
CAN_FLAG_MAL2	mailbox 2 arbitration lost
CAN_FLAG_MAL1	mailbox 1 arbitration lost
CAN_FLAG_MAL0	mailbox 0 arbitration lost
CAN_FLAG_MTFNERR 2	mailbox 2 transmit finished with no error
CAN_FLAG_MTFNERR 1	mailbox 1 transmit finished with no error
CAN_FLAG_MTFNERR 0	mailbox 0 transmit finished with no 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

## Enum can\_error\_enum

**Table 3-84. Enum can\_error\_enum**

enum name	enum description
CAN_ERROR_NONE	no error
CAN_ERROR_FILL	fill error
CAN_ERROR_FORMATE	format error
CAN_ERROR_ACK	ACK error
CAN_ERROR_BITRECESSIVE	bit recessive error
CAN_ERROR_BITDOMINANTER	bit dominant error
CAN_ERROR_CRC	CRC error
CAN_ERROR_SOFTWARECFG	software configure

## Enum can\_transmit\_state\_enum

**Table 3-85. Enum can\_transmit\_state\_enum**

enum name	enum description
CAN_TRANSMIT_FAILED	CAN transmitted failure
CAN_TRANSMIT_OK	CAN transmitted success
CAN_TRANSMIT_PENDING	CAN transmitted pending
CAN_TRANSMIT_NOMAILBOX	no empty mailbox to be used for CAN

## Enum can\_format\_fifo\_enum

**Table 3-86. Enum can\_format\_fifo\_enum**

enum name	enum description
CAN_STANDARD_FIFO0	standard frame and used FIFO0
CAN_STANDARD_FIFO1	standard frame and used FIFO1
CAN_EXTENDED_FIFO0	extended frame and used FIFO0
CAN_EXTENDED_FIFO1	extended frame and used FIFO1

## Enum can\_struct\_type\_enum

**Table 3-87. Enum can\_struct\_type\_enum**

enum name	enum description
CAN_INIT_STRUCT	CAN initilize parameters struct
CAN_FILTER_STRUCT	CAN filter struct
CAN_TX_MESSAGE_STRUCT	CAN transmit message struct
CAN_RX_MESSAGE_STRUCT	CAN receive message struct

## can\_deinit

The description of can\_deinit is shown as below:

Table 3-88. Function can\_deinit

Function name	can_deinit
Function prototype	void can_deinit(uint32_t can_periph);
Function descriptions	deinitialize CAN
Precondition	-
The called functions	rcu_periph_reset_enable/ rcu_periph_reset_disable
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0, 1, 2)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### can\_struct\_para\_init

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

Table 3-89. Function can\_struct\_para\_init

Function name	can_struct_para_init
Function prototype	void can_struct_para_init(can_struct_type_enum type, void* p_struct)
Function descriptions	initialize CAN parameter struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
type	CAN peripheral refer to <a href="#">Table 3-87. Enum can_struct_type_enum</a>
CAN_INIT_STRUCT	CAN initialize parameters struct
CAN_FILTER_STRUCT	CAN filter parameters struct
CAN_FD_FRAME_STRUCT	CAN initialize FD frame parameters struct
CAN_TX_MESSAGE_STRUCT	CAN transmit message struct
CAN_RX_MESSAGE_STRUCT	CAN receive message struct
Output parameter{out}	
p_struct	the struct pointer that needs initialize
Return value	
-	-

Example:

```
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 3-90. 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>	
can_periph	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
<b>Input parameter{in}</b>	
can_parameter_init	CAN parameter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-74. can_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
ErrStatus	SUCCESS / ERROR

Example:

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

```
can_parameter_struct can_parameter_init;
```

```
can_init (CAN0, &can_parameter_init);
```

## can\_filter\_init

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

**Table 3-91. 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_periph	CAN peripheral

<i>CANx(x=0,2)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>can_filter_parameter_init</b>	CAN filter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-81. can_filter_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
can_filter_init(CAN0, &can_filter);
```

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

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

**Table 3-92. Function can\_filter\_mask\_mode\_init**

<b>Function name</b>	can_filter_mask_mode_init
<b>Function prototype</b>	void can_filter_mask_mode_init(uint32_t id, uint32_t mask, can_format_fifo_enum format_fifo, uint16_t filter_number)
<b>Function descriptions</b>	CAN filter mask mode initialization
<b>Precondition</b>	-
<b>The called functions</b>	can_filter_init()
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<i>CANx(x=0,2)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>id</b>	value range (0x00000000 - 0xFFFFFFFF)
<b>Input parameter{in}</b>	
<b>mask</b>	value range (0x00000000 - 0xFFFFFFFF)
<b>Input parameter{in}</b>	
<b>format_fifo</b>	format and fifo states refer to <a href="#">Table 3-86. Enum can_format_fifo_enum</a> , only one parameter can be selected which is shown as below
<i>CAN_STANDARD_FIFO0</i>	standard format and store to FIFO0
<i>CAN_STANDARD_FIFO1</i>	standard format and store to FIFO1
<i>CAN_EXTENDED_FIFO0</i>	extended format and store to FIFO0
<i>CAN_EXTENDED_FIFO1</i>	extended format and store to FIFO1
<b>Input parameter{in}</b>	

<b>filter_number</b>	filter sequence number, value range(0x00 - 0x1C)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
can_filter_mask_mode_init(CAN0, 0x11, 0x11, CAN_STANDARD_FIFO0, 0);
```

### can\_monitor\_mode\_set

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

**Table 3-93. Function can\_monitor\_mode\_set**

<b>Function name</b>	can_monitor_mode_set
<b>Function prototype</b>	ErrStatus can_monitor_mode_set(uint32_t can_periph, uint8_t mode)
<b>Function descriptions</b>	CAN communication mode configure
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<i>CANx(x=0,1,2)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>mode</b>	communication mode, only one parameter can be selected which is shown as below
<i>CAN_NORMAL_MODE</i>	normal mode
<i>CAN_LOOPBACK_MODE</i>	loopback mode
<i>CAN_SILENT_MODE</i>	silent mode
<i>CAN_SILENT_LOOPBACK_MODE</i>	silent loopback mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS / ERROR

Example:

```
can_monitor_mode_set(CAN0, CAN_NORMAL_MODE);
```

### can\_fd\_init (for CAN-FD)

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

**Table 3-94. Function can\_fd\_init**

<b>Function name</b>	can_fd_init
----------------------	-------------

<b>Function prototype</b>	ErrStatus can_fd_init(uint32_t can_periph, can_fdframe_struct* can_fdframe_init);
<b>Function descriptions</b>	initialize CAN FD function
<b>Precondition</b>	can_struct_para_init()
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<b>CANx(x=0,1,2)</b>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>can_fdframe_init</b>	parameters for CAN FD initialization, the structure members can refer to members of the structure <a href="#">Table 3-79. can_fd_tdc_struct (for CAN-FD)</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS / ERROR

Example:

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

## can\_fd\_function\_enable (for CAN-FD)

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

**Table 3-95. Function can\_fd\_function\_enable**

<b>Function name</b>	can_fd_function_enable
<b>Function prototype</b>	void can_fd_function_enable(uint32_t can_periph)
<b>Function descriptions</b>	CAN FD frame function enable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<b>CANx(x=0,1,2)</b>	CAN peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
can_fd_function_enable(CAN0);
```

## can\_fd\_function\_disable (for CAN-FD)

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

**Table 3-96. Function can\_fd\_function\_disable**

<b>Function name</b>	can_fd_function_disable
<b>Function prototype</b>	void can_fd_function_disable(uint32_t can_periph)
<b>Function descriptions</b>	CAN FD frame function disable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
can_fd_function_disable(CAN0);
```

## can1\_filter\_start\_bank

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

**Table 3-97. 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 fliter start bank number 15*/
```

```
can1_filter_start_bank (15);
```



## can\_debug\_freeze\_enable

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

**Table 3-98. 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=0,1,2)	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 3-99. 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
CANx(x=0,1,2)	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 3-100. 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
CANx(x=0,1)	CAN peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-
<b>Function name</b>	can_time_trigger_mode_enable
<b>Function prototype</b>	void can_time_trigger_mode_enable(uint32_t can_periph);
<b>Function descriptions</b>	enable CAN time trigger mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## can\_time\_trigger\_mode\_disable

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

**Table 3-101. 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>	-

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

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 3-102. 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_trasmit_message_struct* transmit_message);
<b>Function descriptions</b>	transmit CAN message
<b>Precondition</b>	can_struct_para_init()
<b>The called functions</b>	-
Input parameter{in}	
<b>can_periph</b>	CAN peripheral
<i>CANx(x=0, 1, 2)</i>	CAN peripheral selection
Input parameter{in}	
<b>transmit_message</b>	CAN transmit message stuct, the structure members refer to <a href="#">Table 3-75. can_trasmit_message_struct</a> CAN-FD transmit message stuct, the structure members refer to <a href="#">Table 3-76. can_trasmit_message_struct (for CAN-FD)</a>
Output parameter{out}	
-	-
Return value	
<b>uint8_t</b>	0x00-0x03

Example:

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

## can\_transmit\_states

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

**Table 3-103. Function can\_transmit\_states**

<b>Function name</b>	can_transmit_states
<b>Function prototype</b>	can_transmit_state_enum can_transmit_states(uint32_t can_periph, uint8_t mailbox_number);
<b>Function descriptions</b>	get CAN transmit state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>mailbox_number</b>	Mailbox number
CAN_MAILBOXx	CAN_MAILBOXx(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
can_transmit_state_enum	the return value refer to <a href="#">Table 3-85. Enum can_transmit_state_enum</a>

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 3-104. 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
CANx(x=0,1,2)	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>mailbox_number</b>	Mailbox number

CAN_MAILBOXx	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 3-105. 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=0,1,2)	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 refer to <a href="#">Table 3-77. can_receive_message_struct</a> CAN-FD message receive struct, the structure members refer to <a href="#">Table 3-78. can_receive_message_struct (for CAN-FD)</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 3-106. Function can\_fifo\_release**

<b>Function name</b>	can_fifo_release
<b>Function prototype</b>	void can_fifo_release(uint32_t can_periph, uint8_t fifo_number);
<b>Function descriptions</b>	release FIFO0
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>fifo_number</b>	Fifo number
CAN_FIFOx	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 3-107. 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
CANx(x=0,1,2)	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>fifo_number</b>	Fifo number
CAN_FIFOx	CAN_FIFOx(x=0,1)
<b>Output parameter{out}</b>	
-	-

Return value	
uint8_t	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 3-108. 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>	
can_periph	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
<b>Input parameter{in}</b>	
can_working_mode	Mode select
CAN_MODE_INITIALIZE	Initialize mode
CAN_MODE_NORMAL	Normal mode
CAN_MODE_SLEEP	Sleep mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
ErrStatus	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 3-109. 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=0,1,2)</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 3-110. 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=0,1,2)</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 3-111. 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
<b>CANx(x=0,1,2)</b>	CAN peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	0..255

Example:

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

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

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

**Table 3-112. Function can\_transmit\_error\_number\_get**

<b>Function name</b>	can_transmit_error_number_get
<b>Function prototype</b>	uint8_t can_transmit_error_number_get(uint32_t can_periph);
<b>Function descriptions</b>	get CAN transmit error number
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<b>CANx(x=0,1,2)</b>	CAN peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	the return value refer to <a href="#">Table 3-84. Enum can_error_enum</a>

Example:

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

## can\_interrupt\_enable

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

**Table 3-113. Function can\_interrupt\_enable**

<b>Function name</b>	can_interrupt_enable
----------------------	----------------------

<b>Function prototype</b>	void can_interrupt_enable(uint32_t can_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable CAN interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>interrupt</b>	Interrupt type
CAN_INT_TME	transmit mailbox empty interrupt enable
CAN_INT_RFNE0	receive FIFO0 not empty interrupt enable
CAN_INT_RFF0	receive FIFO0 full interrupt enable
CAN_INT_RFO0	receive FIFO0 overfull interrupt enable
CAN_INT_RFNE1	receive FIFO1 not empty interrupt enable
CAN_INT_RFF1	receive FIFO1 full interrupt enable
CAN_INT_RFO1	receive FIFO1 overfull interrupt enable
CAN_INT_WERR	warning error interrupt enable
CAN_INT_PERR	passive error interrupt enable
CAN_INT_BO	bus-off interrupt enable
CAN_INT_ERRN	error number interrupt enable
CAN_INT_ERR	error interrupt enable
CAN_INT_WU	wakeup interrupt enable
CAN_INT_SLPW	sleep working interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

## can\_interrupt\_disable

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

**Table 3-114. 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, 2)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>interrupt</b>	Interrupt type
<i>CAN_INT_TME</i>	transmit mailbox empty interrupt enable
<i>CAN_INT_RFNE0</i>	receive FIFO0 not empty interrupt enable
<i>CAN_INT_RFF0</i>	receive FIFO0 full interrupt enable
<i>CAN_INT_RFO0</i>	receive FIFO0 overfull interrupt enable
<i>CAN_INT_RFNE1</i>	receive FIFO1 not empty interrupt enable
<i>CAN_INT_RFF1</i>	receive FIFO1 full interrupt enable
<i>CAN_INT_RFO1</i>	receive FIFO1 overfull interrupt enable
<i>CAN_INT_WERR</i>	warning error interrupt enable
<i>CAN_INT_PERR</i>	passive error interrupt enable
<i>CAN_INT_BO</i>	bus-off interrupt enable
<i>CAN_INT_ERRN</i>	error number interrupt enable
<i>CAN_INT_ERR</i>	error interrupt enable
<i>CAN_INT_WU</i>	wakeup interrupt enable
<i>CAN_INT_SLPW</i>	sleep working interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

## can\_flag\_get

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

**Table 3-115. Function can\_flag\_get**

<b>Function name</b>	can_flag_get
<b>Function prototype</b>	FlagStatus can_flag_get(uint32_t can_periph, can_flag_enum flag);
<b>Function descriptions</b>	get CAN flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<i>CANx(x=0, 1, 2)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>flag</b>	CAN flags refer to <a href="#">Table 3-83. Enum can_flag_enum</a>
<i>CAN_FLAG_RXL</i>	RX level

<code>CAN_FLAG_LSTRX</code>	last sample value of RX pin
<code>CAN_FLAG_RS</code>	receiving state
<code>CAN_FLAG_TS</code>	transmitting state
<code>CAN_FLAG_SLPIF</code>	status change flag of entering sleep working mode
<code>CAN_FLAG_WUIF</code>	status change flag of wakeup from sleep working mode
<code>CAN_FLAG_ERRIF</code>	error flag
<code>CAN_FLAG_SLPWS</code>	sleep working state
<code>CAN_FLAG_IWS</code>	initial working state
<code>CAN_FLAG_TMLS2</code>	transmit mailbox 2 last sending in Tx FIFO
<code>CAN_FLAG_TMLS1</code>	transmit mailbox 1 last sending in Tx FIFO
<code>CAN_FLAG_TMLS0</code>	transmit mailbox 0 last sending in Tx FIFO
<code>CAN_FLAG_TME2</code>	transmit mailbox 2 empty
<code>CAN_FLAG_TME1</code>	transmit mailbox 1 empty
<code>CAN_FLAG_TME0</code>	transmit mailbox 0 empty
<code>CAN_FLAG_MTE2</code>	mailbox 2 transmit error
<code>CAN_FLAG_MTE1</code>	mailbox 1 transmit error
<code>CAN_FLAG_MTE0</code>	mailbox 0 transmit error
<code>CAN_FLAG_MAL2</code>	mailbox 2 arbitration lost
<code>CAN_FLAG_MAL1</code>	mailbox 1 arbitration lost
<code>CAN_FLAG_MAL0</code>	mailbox 0 arbitration lost
<code>CAN_FLAG_MTFNER</code> <code>R2</code>	mailbox 2 transmit finished with no error
<code>CAN_FLAG_MTFNER</code> <code>R1</code>	mailbox 1 transmit finished with no error
<code>CAN_FLAG_MTFNER</code> <code>R0</code>	mailbox 0 transmit finished with no error
<code>CAN_FLAG_MTF2</code>	mailbox 2 transmit finished
<code>CAN_FLAG_MTF1</code>	mailbox 1 transmit finished
<code>CAN_FLAG_MTF0</code>	mailbox 0 transmit finished
<code>CAN_FLAG_RFO0</code>	receive FIFO0 overfull
<code>CAN_FLAG_RFF0</code>	receive FIFO0 full
<code>CAN_FLAG_RFO1</code>	receive FIFO1 overfull
<code>CAN_FLAG_RFF1</code>	receive FIFO1 full
<code>CAN_FLAG_BOERR</code>	bus-off error
<code>CAN_FLAG_PERR</code>	passive error
<code>CAN_FLAG_WERR</code>	warning error
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

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

can\_flag\_get (CAN0, CAN\_FLAG\_MTF0);

## can\_flag\_clear

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

**Table 3-116. 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, 2)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Input parameter{in}</b>	
<b>flag</b>	CAN flags refer to <a href="#">Table 3-83. Enum can_flag_enum</a>
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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

## can\_interrupt\_flag\_get

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

**Table 3-117. Function can\_interrupt\_flag\_get**

<b>Function name</b>	can_interrupt_flag_get
<b>Function prototype</b>	FlagStatus can_interrupt_flag_get(uint32_t can_periph, can_interrupt_flag_enum flag);

<b>Function descriptions</b>	get CAN interrupt flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<i>CANx(x=0,1,2)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>flag</b>	CAN interrupt flags refer to <a href="#">Table 3-82. Enum can interrupt flag enum</a>
<i>CAN_INT_FLAG_SLPIF</i>	status change interrupt flag of sleep working mode entering
<i>CAN_INT_FLAG_WUIF</i>	status change interrupt flag of wakeup from sleep working mode
<i>CAN_INT_FLAG_ERRIF</i>	error interrupt flag
<i>CAN_INT_FLAG_MTF2</i>	mailbox 2 transmit finished interrupt flag
<i>CAN_INT_FLAG_MTF1</i>	mailbox 1 transmit finished interrupt flag
<i>CAN_INT_FLAG_MTF0</i>	mailbox 0 transmit finished interrupt flag
<i>CAN_INT_FLAG_RFO0</i>	receive FIFO0 overfull interrupt flag
<i>CAN_INT_FLAG_RFF0</i>	receive FIFO0 full interrupt flag
<i>CAN_INT_FLAG_RFO1</i>	receive FIFO1 overfull interrupt flag
<i>CAN_INT_FLAG_RFF1</i>	receive FIFO1 full interrupt flag
<i>CAN_INT_FLAG_RFL1</i>	receive FIFO1 not empty interrupt flag
<i>CAN_INT_FLAG_ERRN</i>	error number interrupt flag
<i>CAN_INT_FLAG_BOERR</i>	bus-off error interrupt flag
<i>CAN_INT_FLAG_PERR</i>	passive error interrupt flag
<i>CAN_INT_FLAG_WERR</i>	warning error 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 3-118. Function can\_interrupt\_flag\_clear

Function name	can_interrupt_flag_clear
Function prototype	void can_interrupt_flag_clear(uint32_t can_periph, can_interrupt_flag_enum flag);
Function descriptions	clear CAN interrupt flag state
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
Input parameter{in}	
flag	CAN interrupt flags refer to <a href="#">Table 3-82. Enum can_interrupt_flag_enum</a>
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
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

## 3.5. CRC

A cycle redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. The CRC registers are listed in chapter [3.5.1](#), the CRC firmware functions are introduced in chapter [3.5.2](#).

### 3.5.1. Descriptions of Peripheral registers

CRC registers are listed in the table shown as below:

**Table 3-119. CRC Registers**

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

### 3.5.2. Descriptions of Peripheral functions

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

**Table 3-120. CRC firmware function**

Function name	Function description
crc_deinit	deinitialize CRC calculation unit
crc_data_register_reset	reset data register to the value of initialization data register
crc_reverse_output_data_enable	enable the reverse operation of output data
crc_reverse_output_data_disable	disable the reverse operation of output data
crc_input_data_reverse_config	configure the CRC input data function
crc_data_register_read	read the data register
crc_free_data_register_read	read the free data register
crc_free_data_register_write	write the free data register
crc_init_data_register_write	write the initialization value register
crc_polynomial_size_set	configure the CRC size of polynomial function
crc_polynomial_set	configure the CRC polynomial value function
crc_single_data_calculate	CRC calculate a 32-bit data
crc_block_data_calculate	CRC calculate a 32-bit data array

#### crc\_deinit

The description of crc\_deinit is shown as below:

**Table 3-121. Function crc\_deinit**

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



Example:

```
/* deinitialize CRC */

crc_deinit();
```

## **crc\_data\_register\_reset**

The description of `crc_data_register_reset` is shown as below:

**Table 3-122. Function `crc_data_register_reset`**

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

Example:

```
/* reset crc data register */

crc_data_register_reset ();
```

## **crc\_reverse\_output\_data\_enable**

The description of `crc_reverse_output_data_enable` is shown as below:

**Table 3-123. Function `crc_reverse_output_data_enable`**

<b>Function name</b>	<code>crc_reverse_output_data_enable</code>
<b>Function prototype</b>	<code>void crc_reverse_output_data_enable (void);</code>
<b>Function descriptions</b>	enable the reverse operation of output data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable CRC reverse operation of output data */
```

```
crc_reverse_output_data_enable ();
```

## crc\_reverse\_output\_data\_disable

The description of `crc_reverse_output_data_disable` is shown as below:

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

<b>Function name</b>	<code>crc_reverse_output_data_disable</code>
<b>Function prototype</b>	<code>void crc_reverse_output_data_disable (void);</code>
<b>Function descriptions</b>	disable the reverse operation of output data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable crc reverse operation of output data */
```

```
crc_reverse_output_data_disable ();
```

## crc\_input\_data\_reverse\_config

The description of `crc_input_data_reverse_config` is shown as below:

**Table 3-125. Function `crc_input_data_reverse_config`**

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

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
crc_input_data_reverse_config (CRC_INPUT_DATA_WORD);
```

### crc\_data\_register\_read

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

**Table 3-126. Function crc\_data\_register\_read**

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

Example:

```
/* read crc data register */
```

```
uint32_t crc_value = 0;
```

```
crc_value = crc_data_register_read();
```

### crc\_free\_data\_register\_read

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

**Table 3-127. Function crc\_free\_data\_register\_read**

Function name	crc_free_data_register_read
Function prototype	uint8_t crc_free_data_register_read(void);
Function descriptions	read the free data register
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
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 3-128. Function crc\_free\_data\_register\_write**

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

Example:

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

crc_free_data_register_write(0x11);
```

### crc\_init\_data\_register\_write

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

**Table 3-129. Function crc\_init\_data\_register\_write**

Function name	crc_init_data_register_write
Function prototype	void crc_init_data_register_write(uint32_t init_data)
Function descriptions	write the initializaiton data register
Precondition	-
The called functions	-
Input parameter{in}	
init_data	specify 32-bit data

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write crc initializaiton data register */
crc_init_data_register_write (0x11223344);
```

### crc\_polynomial\_size\_set

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

**Table 3-130. Function crc\_polynomial\_size\_set**

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

Example:

```
/* configure the CRC polynomial size*/
crc_polynomial_size_set (CRC_CTL_PS_7);
```

### crc\_polynomial\_set

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

**Table 3-131. Function crc\_polynomial\_set**

Function name	crc_polynomial_set
Function prototype	void crc_polynomial_set(uint32_t poly)
Function descriptions	configure the CRC polynomial value function
Precondition	-
The called functions	-

Input parameter{in}	
<b>poly</b>	configurable polynomial value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the CRC polynomial value */
```

```
crc_polynomial_set (0x11223344);
```

### crc\_single\_data\_calculate

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

**Table 3-132. 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>	CRC calculate a 32-bit data
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>sdata</b>	specify 32-bit data
Output parameter{out}	
-	-
Return value	
<b>uint32_t</b>	32-bit CRC calculate value (0-0xFFFFFFFF)

Example:

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

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

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

```
rcu_periph_clock_enable(RCU_CRC);
```

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

### crc\_block\_data\_calculate

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

**Table 3-133. 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 data words
<b>Input parameter{in}</b>	
<b>size</b>	size of the array
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	32-bit CRC calculate value (0-0xFFFFFFFF)

Example:

```

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

#define BUFFER_SIZE    6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {

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

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

```

## 3.6. CTC

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

### 3.6.1. Descriptions of Peripheral registers

CTC registers are listed in the table shown as below:

**Table 3-134. CTC Registers**

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

### 3.6.2. Descriptions of Peripheral functions

CTC registers are listed in the table shown as below:

**Table 3-135. CTC firmware function**

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

#### ctc\_deinit

The description of ctc\_deinit is shown as below:

**Table 3-136. 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 3-137. 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 3-138. Function ctc\_counter\_disable**

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

Return value	
-	-

Example:

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

```
ctc_counter_disable ();
```

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

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

**Table 3-139. Function ctc\_irc48m\_trim\_value\_config**

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

Example:

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

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

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

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

**Table 3-140. Function ctc\_software\_refsource\_pulse\_generate**

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

Example:

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

ctc_software_refsource_pulse_generate ();
```

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

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

**Table 3-141. Function ctc\_hardware\_trim\_mode\_config**

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

Example:

```
/* enable CTC hardware trim */

ctc_hardware_trim_mode_config (CTC_HARDWARE_TRIM_MODE_ENABLE);
```

### ctc\_refsource\_polarity\_config

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

**Table 3-142. Function ctc\_refsource\_polarity\_config**

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

<i>CTC_REFSOURCE_POLARITY_RISING</i>	reference signal source polarity is rising edge
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
ctc_refsource_polarity_config (CTC_REFSOURCE_POLARITY_RISING);
```

## ctc\_refsource\_signal\_select

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

**Table 3-143. 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
<i>CTC_REFSOURCE_GPIO</i>	GPIO is selected
<i>CTC_REFSOURCE_LXTAL</i>	LXTAL is selected
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
ctc_refsource_signal_select (CTC_REFSOURCE_LXTAL);
```

## ctc\_refsource\_prescaler\_config

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

**Table 3-144. Function ctc\_refsource\_prescaler\_config**

<b>Function name</b>	ctc_refsource_prescaler_config
<b>Function prototype</b>	void ctc_refsource_prescaler_config(uint32_t prescaler);
<b>Function descriptions</b>	configure reference signal source prescaler

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>prescaler</b>	Prescaler factor
<i>CTC_REFSOURCE_P</i> <i>SC_OFF</i>	reference signal not divided
<i>CTC_REFSOURCE_P</i> <i>SC_DIV2</i>	reference signal divided by 2
<i>CTC_REFSOURCE_P</i> <i>SC_DIV4</i>	reference signal divided by 4
<i>CTC_REFSOURCE_P</i> <i>SC_DIV8</i>	reference signal divided by 8
<i>CTC_REFSOURCE_P</i> <i>SC_DIV16</i>	reference signal divided by 16
<i>CTC_REFSOURCE_P</i> <i>SC_DIV32</i>	reference signal divided by 32
<i>CTC_REFSOURCE_P</i> <i>SC_DIV64</i>	reference signal divided by 64
<i>CTC_REFSOURCE_P</i> <i>SC_DIV128</i>	reference signal divided by 128
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
ctc_refsource_prescaler_config(CTC_REFSOURCE_PSC_DIV2);
```

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

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

**Table 3-145. Function ctc\_clock\_limit\_value\_config**

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

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 3-146. 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 3-147. Function ctc\_counter\_capture\_value\_read**

Function name	ctc_counter_capture_value_read
Function prototype	uint16_t ctc_counter_capture_value_read(void);
Function descriptions	read CTC counter capture value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	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 3-148. Function ctc\_counter\_direction\_read**

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

Example:

```
/* read ctc counter direction */

FlagStatus ctc_direction = SET;

ctc_direction = ctc_counter_direction_read ();
```

## ctc\_counter\_reload\_value\_read

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

**Table 3-149. Function ctc\_counter\_reload\_value\_read**

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

<b>uint16_t</b>	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 3-150. Function ctc\_irc48m\_trim\_value\_read**

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

Example:

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

uint8_t ctc_trim_value = 0;

ctc_trim_value = ctc_irc48m_trim_value_read ();
```

### ctc\_flag\_get

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

**Table 3-151. 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
<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_EREFP</i>	expect reference interrupt flag
<i>CTC_FLAG_CKERR</i>	clock trim error bit
<i>CTC_FLAG_REFMIS</i>	reference sync pulse miss flag
<i>CTC_FLAG_TRIMERR</i>	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 3-152. 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_EREFP</i>	expect reference interrupt flag
<i>CTC_FLAG_CKERR</i>	clock trim error bit
<i>CTC_FLAG_REFMIS</i>	reference sync pulse miss flag
<i>CTC_FLAG_TRIMERR</i>	trim value error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
ctc_flag_clear (CTC_FLAG_CKOK);
```

## ctc\_interrupt\_enable

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

**Table 3-153. Function ctc\_interrupt\_enable**

<b>Function name</b>	ctc_interrupt_enable
<b>Function prototype</b>	void ctc_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable the CTC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	CTC interrupt
<i>CTC_INT_CKOK</i>	clock trim OK interrupt
<i>CTC_INT_CKWARN</i>	clock trim warning interrupt
<i>CTC_INT_ERR</i>	error interrupt
<i>CTC_INT_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 3-154. 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 3-155. Function ctc\_interrupt\_flag\_get**

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

Example:

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

FlagStatus state = ctc_interrupt_flag_get (CTC_INT_FLAG_CKOK);
```

### ctc\_interrupt\_flag\_clear

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

Table 3-156. Function `ctc_interrupt_flag_clear`

<b>Function name</b>	<code>ctc_interrupt_flag_clear</code>
<b>Function prototype</b>	<code>void ctc_interrupt_flag_clear(uint32_t int_flag);</code>
<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
<code>CTC_INT_FLAG_CKOK</code>	clock trim OK interrupt
<code>CTC_INT_FLAG_CKWARN</code>	clock trim warning interrupt
<code>CTC_INT_FLAG_ERR</code>	error interrupt
<code>CTC_INT_FLAG_EREFP</code>	expect reference interrupt
<code>CTC_INT_FLAG_CKEERR</code>	clock trim error bit interrupt
<code>CTC_INT_FLAG_REFMISS</code>	reference sync pulse miss interrupt
<code>CTC_INT_FLAG_TRIMERR</code>	trim value error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear CTC interrupt flag status */
ctc_interrupt_flag_clear (CTC_INT_FLAG_CKOK);
```

## 3.7. CMP

The general purpose comparator can work either standalone(all terminal are available on I/Os) or together with the timers. It provide a trigger source when an analog signal is in a certain condition. The CMP registers are listed in chapter [3.7.1](#), the CMP firmware functions are introduced in chapter [3.7.2](#).

### 3.7.1. Descriptions of Peripheral registers

CMP registers are listed in the table shown as below:

Table 3-157. CMP Registers

Registers	Descriptions
CMP1_CS	CMP1 control and status register
CMP3_CS	CMP3 control and status register
CMP5_CS	CMP5 control and status register

### 3.7.2. Descriptions of Peripheral functions

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

Table 3-158. CMP firmware function

Function name	Function description
cmp_deinit	CMP deinit
cmp_mode_init	CMP mode init
cmp_output_init	CMP output init
cmp_blanking_init	CMP output blanking function init
cmp_enable	enable CMP
cmp_disable	disable CMP
cmp_lock_enable	lock the CMP
cmp_output_level_get	get output level

#### Enum cmp\_enum

Table 3-159. Enum cmp\_enum

Member name	Function description
CMP1	cmoparator 1
CMP3	cmoparator 3
CMP5	cmoparator 5

#### cmp\_deinit

The description of cmp\_deinit is shown as below:

Table 3-160. Function cmp\_deinit

Function name	cmp_deinit
Function prototype	void cmp_deinit(cmp_enum cmp_periph);
Function descriptions	CMP deinit
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-159. Enum cmp_enum</a>
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* deinitialize CMP1 */
```

```
cmp_deinit(CMP1);
```

### cmp\_mode\_init

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

**Table 3-161. Function cmp\_mode\_init**

<b>Function name</b>	cmp_mode_init
<b>Function prototype</b>	void cmp_mode_init(cmp_enum cmp_periph, uint32_t inverting_input);
<b>Function descriptions</b>	CMP mode init
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-159. Enum cmp_enum</a>
<b>Input parameter{in}</b>	
<b>inverting_input</b>	inverting input select
<i>CMP_INVERTING_INP UT_1_4VREFINT</i>	VREFINT *1/4 input
<i>CMP_INVERTING_INP UT_1_2VREFINT</i>	VREFINT *1/2 input
<i>CMP_INVERTING_INP UT_3_4VREFINT</i>	VREFINT *3/4 input
<i>CMP_INVERTING_INP UT_VREFINT</i>	VREFINT input
<i>CMP_INVERTING_INP UT_DAC0_OUT0</i>	PA4(DAC) input
<i>CMP_INVERTING_INP UT_PA5</i>	PA5 input
<i>CMP_INVERTING_INP UT_PA2</i>	PA2 only for CMP1
<i>CMP_INVERTING_INP UT_PB2_PB15</i>	PB2 for CMP3 or PB15 for CMP5 as inverting input
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize CMP1 mode */
```

cmp\_mode\_init (CMP1, CMP\_INVERTING\_INPUT\_1\_4VREFINT);

## cmp\_output\_init

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

**Table 3-162. Function cmp\_output\_init**

<b>Function name</b>	cmp_output_init
<b>Function prototype</b>	void cmp_output_init(cmp_enum cmp_periph, uint32_t output_selection, uint32_t output_polarity)
<b>Function descriptions</b>	CMP output init
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-159. Enum cmp_enum</a>
<b>Input parameter{in}</b>	
<b>output_selection</b>	CMP output select
CMP_OUTPUT_NONE	output no selection
CMP_OUTPUT_TIMER0_BKIN	CMP output TIMER0 break input
CMP_OUTPUT_TIMER2_IC2	CMP output TIMER2_CH2 input capture only for CMP3
CMP_OUTPUT_TIMER1_IC1	CMP output TIMER1_CH1 input capture only for CMP5
CMP_OUTPUT_TIMER0_IC0	CMP output TIMER0_CH0 input capture only for CMP1
CMP_OUTPUT_TIMER1_IC3	CMP output TIMER1_CH3 input capture only for CMP1
CMP_OUTPUT_TIMER2_IC0	CMP output TIMER2_CH0 input capture only for CMP1
<b>Input parameter{in}</b>	
<b>output_polarity</b>	CMP output polarity select
CMP_OUTPUT_POLARITY_INVERTED	output is inverted
CMP_OUTPUT_POLARITY_NOINVERTED	output is not inverted
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize CMP1 output */
```

cmp\_output\_init (CMP1,CMP\_OUTPUT\_NONE,  
CMP\_OUTPUT\_POLARITY\_NOINVERTED);

## cmp\_blanking\_init

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

**Table 3-163. Function cmp\_blanking\_init**

<b>Function name</b>	cmp_blanking_init
<b>Function prototype</b>	void cmp_blanking_init(cmp_enum cmp_periph, uint32_t blanking_source_selection);
<b>Function descriptions</b>	CMP output blanking function init
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-159. Enum cmp_enum</a>
<b>Input parameter{in}</b>	
<b>blanking_source_selection</b>	blanking source selection
<i>CMP_BLANKING_NONE</i>	CMP no blanking source
<i>CMP_BLANKING_TIMER2_OC3</i>	CMP TIMER2_CH3 output compare signal selected as blanking source only for CMP3
<i>CMP_BLANKING_TIMER1_OC2</i>	CMP TIMER1_CH2 output compare signal selected as blanking source only for CMP1
<i>CMP_BLANKING_TIMER2_OC2</i>	CMP TIMER2_CH2 output compare signal selected as blanking source only for CMP1
<i>CMP_BLANKING_TIMER1_OC3</i>	CMP TIMER1_CH3 output compare signal selected as blanking source only for CMP5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize CMP1 blanking function */
```

```
cmp_blanking_init (CMP1, CMP_BLANKING_TIMER1_OC2);
```

## cmp\_enable

The description of cmp\_enable is shown as below:

**Table 3-164. Function cmp\_enable**

<b>Function name</b>	cmp_enable
----------------------	------------



<b>Function prototype</b>	void cmp_enable(cmp_enum cmp_periph);
<b>Function descriptions</b>	enable CMP
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-159. Enum cmp_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable CMP1 */
cmp_enable (CMP1);
```

### cmp\_disable

The description of cmp\_disable is shown as below:

**Table 3-165. Function cmp\_disable**

<b>Function name</b>	cmp_disable
<b>Function prototype</b>	void cmp_disable(cmp_enum cmp_periph);
<b>Function descriptions</b>	disable CMP
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-159. Enum cmp_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable CMP1 */
cmp_disable (CMP1);
```

### cmp\_lock\_enable

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

**Table 3-166. Function cmp\_lock\_enable**

<b>Function name</b>	cmp_lock_enable
<b>Function prototype</b>	void cmp_lock_enable(cmp_enum cmp_periph);
<b>Function descriptions</b>	lock the CMP

Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-159. Enum cmp_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock CMP1 register */
cmp_lock_enable (CMP1);
```

### cmp\_output\_level\_get

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

**Table 3-167. Function cmp\_output\_level\_get**

Function name	cmp_output_level_get
Function prototype	uint32_t cmp_output_level_get(cmp_enum cmp_periph);
Function descriptions	get output level
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-159. Enum cmp_enum</a>
Output parameter{out}	
-	-
Return value	
uint32_t	the output level
CMP_OUTPUTLEVEL_HIGH	comparator output high
CMP_OUTPUTLEVEL_LOW	comparator output low

Example:

```
uint32_t level;

/* get CMP1 output level */
level = cmp_output_level_get(CMP1);
```

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

## 3.8.1. Descriptions of Peripheral registers

DAC registers are listed in the table shown as below:

**Table 3-168. DAC Registers**

Registers	Descriptions
DAC_CTL0	DACx control register 0
DAC_SWT	DACx software trigger register
DAC_OUT0_R12DH	DACx_OUT0 12-bit right-aligned data holding register
DAC_OUT0_L12DH	DACx_OUT0 12-bit left-aligned data holding register
DAC_OUT0_R8DH	DACx_OUT0 8-bit right-aligned data holding register
DAC_OUT1_R12DH	DACx_OUT1 12-bit right-aligned data holding register
DAC_OUT1_L12DH	DACx_OUT1 12-bit left-aligned data holding register
DAC_OUT1_R8DH	DACx_OUT1 8-bit right-aligned data holding register
DACC_R12DH	DACx concurrent mode 12-bit right-aligned data holding register
DACC_L12DH	DACx concurrent mode 12-bit left-aligned data holding register
DACC_R8DH	DACx concurrent mode 8-bit right-aligned data holding register
DAC_OUT0_DO	DACx_OUT0 data output register
DAC_OUT1_DO	DACx_OUT1 data output register
DAC_STAT0	DACx status register 0
DAC_CTL1	DACx control register 1
DAC_STAT1	DACx status register 1

## 3.8.2. Descriptions of Peripheral functions

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

**Table 3-169. DAC firmware functions**

Function name	Function description
<code>dac_deinit</code>	deinitialize DAC
<code>dac_enable</code>	enable DAC
<code>dac_disable</code>	disable DAC
<code>dac_dma_enable</code>	enable DAC DMA function
<code>dac_dma_disable</code>	disable DAC DMA function
<code>dac_output_buffer_enable</code>	enable DAC output buffer
<code>dac_output_buffer_disable</code>	disable DAC output buffer
<code>dac_output_value_get</code>	get DAC output value
<code>dac_data_set</code>	set DAC data holding register value
<code>dac_trigger_enable</code>	enable DAC trigger
<code>dac_trigger_disable</code>	disable DAC trigger
<code>dac_trigger_source_config</code>	configure DAC trigger source

Function name	Function description
<code>dac_software_trigger_enable</code>	enable DAC software trigger
<code>dac_wave_mode_config</code>	configure DAC wave mode
<code>dac_lfsr_noise_config</code>	configure DAC LFSR noise mode
<code>dac_triangle_noise_config</code>	configure DAC triangle noise mode
<code>dac_concurrent_enable</code>	enable DAC concurrent mode
<code>dac_concurrent_disable</code>	disable DAC concurrent mode
<code>dac_concurrent_software_trigger_enable</code>	enable DAC concurrent software trigger
<code>dac_concurrent_output_buffer_enable</code>	enable DAC concurrent buffer function
<code>dac_concurrent_output_buffer_disable</code>	disable DAC concurrent buffer function
<code>dac_concurrent_data_set</code>	set DAC concurrent mode data holding register value
<code>dac_output_fifo_enable</code>	enable DAC output FIFO
<code>dac_output_fifo_disable</code>	disable DAC output FIFO
<code>dac_output_fifo_number_get</code>	get DAC output FIFO number
<code>dac_flag_get</code>	get DAC flag
<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 3-170. Function `dac_deinit`**

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

Example:

```
/* deinitialize DAC0 */
dac_deinit(DAC0);
```

## dac\_enable

The description of dac\_enable is shown as below:

**Table 3-171. Function dac\_enable**

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

Example:

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

## dac\_disable

The description of dac\_disable is shown as below:

**Table 3-172. Function dac\_disable**

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

-	-
---	---

Example:

```
/* disable DAC0_OUT0 */
dac_disable(DAC0, DAC_OUT0);
```

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

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

**Table 3-173. Function dac\_dma\_enable**

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

Example:

```
/* enable DAC0_OUT0 DMA function */
dac_dma_enable(DAC0, DAC_OUT0);
```

### **dac\_dma\_disable**

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

**Table 3-174. Function dac\_dma\_disable**

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

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 3-175. Function dac\_output\_buffer\_enable**

<b>Function name</b>	dac_output_buffer_enable
<b>Function prototype</b>	void dac_output_buffer_enable(uint32_t dac_periph, uint8_t dac_out);
<b>Function descriptions</b>	enable DAC output buffer
<b>Precondition</b>	-
<b>The called functions</b>	-
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 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 3-176. Function dac\_output\_buffer\_disable**

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

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

Example:

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

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

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

**Table 3-177. Function dac\_output\_value\_get**

<b>Function name</b>	dac_output_value_get
<b>Function prototype</b>	uint16_t dac_output_value_get(uint32_t dac_periph, uint8_t dac_out);
<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 3-178. 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 3-179. Function `dac_trigger_enable`**

<b>Function name</b>	<code>dac_trigger_enable</code>
<b>Function prototype</b>	<code>void dac_trigger_enable(uint32_t dac_periph, uint8_t dac_out);</code>
<b>Function descriptions</b>	enable DAC trigger
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
<b>Input parameter{in}</b>	
<b>dac_out</b>	DAC output
<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
<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 3-180. 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 3-181. 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_TRG</i> 0	TIMER5 TRGO
<i>DAC_TRIGGER_T7_TRG</i> 0	TIMER7 TRGO (for GD32E50X_HD devices)
<i>DAC_TRIGGER_T2_TRG</i> 0	TIMER2 TRGO (for GD32E50X_CL devices)
<i>DAC_TRIGGER_T6_TRG</i> 0	TIMER6 TRGO
<i>DAC_TRIGGER_T4_TRG</i> 0	TIMER4 TRGO
<i>DAC_TRIGGER_T1_TRG</i> 0	TIMER1 TRGO
<i>DAC_TRIGGER_T3_TRG</i> 0	TIMER3 TRGO
<i>DAC_TRIGGER_EXTI_9</i>	EXTI interrupt line9 event
<i>DAC_TRIGGER_SOFTWARE</i>	software trigger
<i>DAC_TRIGGER_SHRTIMER_DACTRIG0</i>	SHRTIMER_DACTRIG0 trigger (for GD32E50X_HD and GD32E50X_CL devices)
<i>DAC_TRIGGER_SHRTIMER_DACTRIG1</i>	SHRTIMER_DACTRIG1 trigger (for GD32E50X_HD and GD32E50X_CL devices)
<i>DAC_TRIGGER_SHRTIMER_DACTRIG2</i>	SHRTIMER_DACTRIG2 trigger (for GD32E50X_HD and GD32E50X_CL devices)
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

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 3-182. 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 3-183. Function `dac_wave_mode_config`**

<b>Function name</b>	<code>dac_wave_mode_config</code>
<b>Function prototype</b>	<code>void dac_wave_mode_config(uint32_t dac_periph, uint8_t dac_out, uint32_t wave_mode);</code>
<b>Function descriptions</b>	configure DAC wave mode
<b>Precondition</b>	-

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

Example:

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

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

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

The description of `dac_lfsr_noise_config` is shown as below:

**Table 3-184. Function `dac_lfsr_noise_config`**

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

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

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

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

**Table 3-185. Function dac\_triangle\_noise\_config**

Function name	dac_triangle_noise_config
Function prototype	void dac_triangle_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t amplitude);
Function descriptions	configure DAC triangle noise 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>amplitude</b>	the amplitude of the triangle
<i>DAC_TRIANGLE_AMPLITUDE_x</i>	$x = 2^n - 1$ (n = 1..12)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

## **dac\_concurrent\_enable**

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

Table 3-186. 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>	-
<b>Input parameter{in}</b>	
<b><code>dac_periph</code></b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **`dac_concurrent_disable`**

The description of `dac_concurrent_disable` is shown as below:

Table 3-187. 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>	-
<b>Input parameter{in}</b>	
<b><code>dac_periph</code></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 3-188. 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><code>dac_periph</code></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 3-189. 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><code>dac_periph</code></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 3-190. 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><code>dac_periph</code></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 3-191. 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><code>dac_periph</code></b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b><code>dac_align</code></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><code>data0</code></b>	DACx_OUT0 data to be loaded (0~4095)
<b>Input parameter{in}</b>	
<b><code>data1</code></b>	DACx_OUT1 data to be loaded (0~4095)
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

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

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

### **dac\_output\_fifo\_enable**

The description of `dac_output_fifo_enable` is shown as below:

**Table 3-192. Function `dac_output_fifo_enable`**

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

Example:

```
/* enable DAC0_OUT0 output FIFO */
```

```
dac_output_fifo_enable(DAC0, DAC_OUT0);
```

### **dac\_output\_fifo\_disable**

The description of `dac_output_fifo_disable` is shown as below:

**Table 3-193. Function `dac_output_fifo_disable`**

Function name	<code>dac_output_fifo_disable</code>
Function prototype	<code>void dac_output_fifo_disable(uint32_t dac_periph, uint8_t dac_out);</code>
Function descriptions	disable DAC output FIFO
Precondition	-
The called functions	-
Input parameter{in}	
<code>dac_periph</code>	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 FIFO */
dac_output_fifo_disable(DAC0, DAC_OUT0);
```

### **dac\_output\_fifo\_number\_get**

The description of `dac_output_fifo_number_get` is shown as below:

**Table 3-194. Function `dac_output_fifo_number_get`**

<b>Function name</b>	<code>dac_output_fifo_number_get</code>
<b>Function prototype</b>	<code>uint16_t dac_output_fifo_number_get(uint32_t dac_periph, uint8_t dac_out);</code>
<b>Function descriptions</b>	get DAC output FIFO number
<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 FIFO number (0~4)

Example:

```
/* get DAC0_OUT0 output FIFO number */
uint16_t number =0;
number = dac_output_fifo_number_get (DAC0, DAC_OUT0);
```

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

The description of `dac_flag_get` is shown as below:

Table 3-195. 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_FIFOF0</i>	DACx_OUT0 FIFO full flag
<i>DAC_FLAG_FIFOE0</i>	DACx_OUT0 FIFO empty flag
<i>DAC_FLAG_FIFOVR0</i>	DACx_OUT0 FIFO overflow flag
<i>DAC_FLAG_FIFOU0R0</i>	DACx_OUT0 FIFO underflow flag
<i>DAC_FLAG_DDUDR1</i>	DACx_OUT1 DMA underrun flag
<i>DAC_FLAG_FIFOF1</i>	DACx_OUT1 FIFO full flag
<i>DAC_FLAG_FIFOE1</i>	DACx_OUT1 FIFO empty flag
<i>DAC_FLAG_FIFOVR1</i>	DACx_OUT1 FIFO overflow flag
<i>DAC_FLAG_FIFOU1R1</i>	DACx_OUT1 FIFO underflow 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 3-196. Function `dac_flag_clear`

<b>Function name</b>	<code>dac_flag_clear</code>
<b>Function prototype</b>	<code>void dac_flag_clear(uint32_t dac_periph, uint32_t flag);</code>
<b>Function descriptions</b>	clear DAC flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral

<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b>flag</b>	DAC flag
<i>DAC_FLAG_DDUDR0</i>	DACx_OUT0 DMA underrun flag
<i>DAC_FLAG_FIFOOVR0</i>	DACx_OUT0 FIFO overflow flag
<i>DAC_FLAG_FIFOUDR0</i>	DACx_OUT0 FIFO underflow flag
<i>DAC_FLAG_DDUDR1</i>	DACx_OUT1 DMA underrun flag
<i>DAC_FLAG_FIFOOVR1</i>	DACx_OUT1 FIFO overflow flag
<i>DAC_FLAG_FIFOUDR1</i>	DACx_OUT1 FIFO underflow 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 3-197. Function `dac_interrupt_enable`**

<b>Function name</b>	<code>dac_interrupt_enable</code>
<b>Function prototype</b>	<code>void dac_interrupt_enable(uint32_t dac_periph, uint32_t interrupt);</code>
<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_FIFOOVR0</i>	DACx_OUT0 FIFO overflow interrupt
<i>DAC_INT_FIFOUDR0</i>	DACx_OUT0 FIFO underflow interrupt
<i>DAC_INT_DDUDR1</i>	DACx_OUT1 DMA underrun interrupt
<i>DAC_INT_FIFOOVR1</i>	DACx_OUT1 FIFO overflow interrupt
<i>DAC_INT_FIFOUDR1</i>	DACx_OUT1 FIFO underflow 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 3-198. 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_FIFOVR0</i>	DACx_OUT0 FIFO overflow interrupt
<i>DAC_INT_FIFODR0</i>	DACx_OUT0 FIFO underflow interrupt
<i>DAC_INT_DDUDR1</i>	DACx_OUT1 DMA underrun interrupt
<i>DAC_INT_FIFOVR1</i>	DACx_OUT1 FIFO overflow interrupt
<i>DAC_INT_FIFODR1</i>	DACx_OUT1 FIFO underflow 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 3-199. 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 int_flag);</code>
<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_FIFOOV R0</i>	DACx_OUT0 FIFO overflow interrupt flag
<i>DAC_INT_FLAG_FIFOU R0</i>	DACx_OUT0 FIFO underflow interrupt flag
<i>DAC_INT_FLAG_DDUDR1</i>	DACx_OUT1 DMA underrun interrupt flag
<i>DAC_INT_FLAG_FIFOOV R1</i>	DACx_OUT1 FIFO overflow interrupt flag
<i>DAC_INT_FLAG_FIFOU R1</i>	DACx_OUT1 FIFO underflow 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 3-200. Function `dac_interrupt_flag_clear`**

<b>Function name</b>	<code>dac_interrupt_flag_clear</code>
<b>Function prototype</b>	<code>void dac_interrupt_flag_clear(uint32_t dac_periph, uint32_t int_flag);</code>
<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_FIFOOV</i> <i>R0</i>	DACx_OUT0 FIFO overflow interrupt flag
<i>DAC_INT_FLAG_FIFOU</i> <i>R0</i>	DACx_OUT0 FIFO underflow interrupt flag
<i>DAC_INT_FLAG_DDUDR1</i>	DACx_OUT1 DMA underrun interrupt flag
<i>DAC_INT_FLAG_FIFOOV</i> <i>R1</i>	DACx_OUT1 FIFO overflow interrupt flag
<i>DAC_INT_FLAG_FIFOU</i> <i>R1</i>	DACx_OUT1 FIFO underflow 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);
```

## 3.9. DBG

The DBG hold unit helps debugger to debug power saving mode. The DBG registers are listed in chapter [3.9.1](#), the DBG firmware functions are introduced in chapter [3.9.2](#).

### 3.9.1. Descriptions of Peripheral registers

DBG registers are listed in the table shown as below:

**Table 3-201. DBG Registers**

Registers	Descriptions
DBG_ID	DBG ID code register
DBG_CTL	DBG control register

### 3.9.2. Descriptions of Peripheral functions

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

**Table 3-202. DBG firmware function**

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



Function name	Function description
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 3-203. Enum dbg\_periph\_enum**

Member name	Function description
DBG_CAN2_HOLD	hold CAN2 receive register counter when core is halted
DBG_FWDGT_HOLD	hold FWDGT counter when core is halted
DBG_WWDGT_HOLD	hold WWDGT counter 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	hold CAN0 receive register counter when core is halted
DBG_I2C0_HOLD	hold I2C0 smbus when core is halted
DBG_I2C1_HOLD	hold I2C1 smbus when core is halted
DBG_TIMER7_HOLD	hold TIMER7 counter when core is halted
DBG_TIMER4_HOLD	hold TIMER4 counter when core is halted
DBG_TIMER5_HOLD	hold TIMER5 counter when core is halted
DBG_TIMER6_HOLD	hold TIMER6 counter when core is halted
DBG_CAN1_HOLD	hold CAN1 receive register counter when core
DBG_I2C2_HOLD	hold I2C2 smbus timeout 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_SHRTIMER_HOLD	hold SHRTIMER counter when core is halted, except for GD32EPRT series

## dbg\_deinit

The description of dbg\_deinit is shown as below:

**Table 3-204. Function dbg\_deinit**

Function name	dbg_deinit
Function prototype	void dbg_deinit(void);
Function descriptions	deinitialize the DBG
Precondition	-
The called functions	-

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

Example:

```
/* deinitialize the DBG */
```

```
dbg_deinit();
```

### dbg\_id\_get

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

**Table 3-205. Function dbg\_id\_get**

Function name	dbg_id_get
Function prototype	uint32_t dbg_id_get(void);
Function descriptions	Read DBG_ID code register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
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 3-206. 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}	
<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
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* keep debugger connection during sleep mode */
dbg_low_power_enable(DBG_LOW_POWER_SLEEP);
```

## dbg\_low\_power\_disable

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

**Table 3-207. 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>	-
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* do not keep debugger connection during sleep mode */
dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

## dbg\_periph\_enable

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

**Table 3-208. Function dbg\_periph\_enable**

Function name	dbg_periph_enable
Function prototype	void dbg_periph_enable(dbg_periph_enum dbg_periph);
Function descriptions	Enable peripheral behavior when the mcu is in debug mode
Precondition	-
The called functions	-
Input parameter{in}	
dbg_periph	Peripheral refer to <a href="#">Table 3-203. Enum dbg_periph_enum</a>
DBG_FWDGT_HOLD	debug FWDGT kept when core is halted
DBG_CANx_HOLD	x=0,1,2 hold CANx counter when core is halted
DBG_WWDGT_HOLD	debug WWDGT kept when core is halted
DBG_I2Cx_HOLD	x=0,1,2 hold I2Cx smbus when core is halted
DBG_TIMERx_HOLD	x=0,1,2,3,4,5,6,7,8,9,10,11,12,13, hold TIMERx counter when core is halted. TIMER8..13 are not available for GD32EPRT series.
DBG_SHRTIMER_HOLD	hold SHRTIMER counter when core is halted, except for GD32EPRT series
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* hold TIMER0 counter when core is halted */
```

```
dbg_periph_enable(DBG_TIMER0_HOLD);
```

## dbg\_periph\_disable

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

**Table 3-209. Function dbg\_periph\_disable**

Function name	dbg_periph_disable
Function prototype	void dbg_periph_disable(dbg_periph_enum dbg_periph);
Function descriptions	Disable peripheral behavior when the mcu is in debug mode
Precondition	-
The called functions	-
Input parameter{in}	
dbg_periph	Peripheral refer to <a href="#">Table 3-203. Enum dbg_periph_enum</a>
DBG_FWDGT_HOLD	debug FWDGT kept when core is halted
DBG_WWDGT_HOLD	debug WWDGT kept when core is halted
DBG_CANx_HOLD	x=0,1,2 hold CANx counter when core is halted

<i>DBG_I2Cx_HOLD</i>	x=0,1,2hold 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. TIMER8..13 are not available for GD32EPRT series.
<i>DBG_SHRTIMER_HOLD</i>	hold SHRTIMER counter when core is halted, except for GD32EPRT series
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* do not hold TIMER0 counter when core is halted */
```

```
dbg_periph_disable(DBG_TIMER0_HOLD);
```

### dbg\_trace\_pin\_enable

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

**Table 3-210. 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 3-211. 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>	-

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

Example:

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

## 3.10. 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 [3.10.1](#), the DMA firmware functions are introduced in chapter [3.10.2](#).

### 3.10.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

**Table 3-212. DMA Registers**

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

### 3.10.2. Descriptions of Peripheral functions

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

**Table 3-213. DMA firmware function**

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

## Enum `dma_channel_enum`

**Table 3-214. Enum `dma_channel_enum`**

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

## Structure dma\_parameter\_struct

**Table 3-215. Structure dma\_parameter\_struct**

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

## dma\_deinit

The description of dma\_deinit is shown as below:

**Table 3-216. Function dma\_deinit**

<b>Function name</b>	dma_deinit
<b>Function prototype</b>	void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	deinitialize DMA a channel registers
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
<i>DMA_CHx(DMA0: x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## dma\_struct\_para\_init

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



**Table 3-217. Function dma\_struct\_para\_init**

<b>Function name</b>	dma_struct_para_init
<b>Function prototype</b>	void dma_struct_para_init(dma_parameter_struct* init_struct);
<b>Function descriptions</b>	initialize the parameters of DMA struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>*init_struct</b>	A dma_parameter_struct address, refer to <a href="#">Table 3-215. Structure dma_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## **dma\_init**

The description of dma\_init is shown as below:

**Table 3-218. Function dma\_init**

<b>Function name</b>	dma_init
<b>Function prototype</b>	void dma_init(uint32_t dma_periph, dma_channel_enum channelx, dma_parameter_struct* init_struct);
<b>Function descriptions</b>	initialize DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-215. Structure dma_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

dma_init_struct.direction = DMA_PERIPHERAL_TO_MEMORY;
dma_init_struct.memory_addr = (uint32_t)g_destbuf;
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_struct.memory_width = DMA_MEMORY_WIDTH_32BIT;
dma_init_struct.number = TRANSFER_NUM/4;
dma_init_struct.periph_addr = (uint32_t)FLASH_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_32BIT;
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_init(DMA0, DMA_CH0, &dma_init_struct);
```

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

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

**Table 3-219. Function dma\_circulation\_enable**

Function name	dma_circulation_enable
Function prototype	void dma_circulation_enable(uint32_t dma_periph, dma_channel_enum channelx);
Function descriptions	enable DMA circulation mode
Precondition	-
The called functions	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

## dma\_circulation\_disable

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

**Table 3-220. 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(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## dma\_memory\_to\_memory\_enable

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

**Table 3-221. Function dma\_memory\_to\_memory\_enable**

<b>Function name</b>	dma_memory_to_memory_enable
<b>Function prototype</b>	void dma_memory_to_memory_enable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	enable memory to memory mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>

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

Example:

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

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

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

**Table 3-222. Function dma\_memory\_to\_memory\_disable**

<b>Function name</b>	dma_memory_to_memory_disable
<b>Function prototype</b>	void dma_memory_to_memory_disable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	disable memory to memory mode
<b>Precondition</b>	-
<b>The called functions</b>	-
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(DMA0:x=0.. 6; DMA1: x=0..4)	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

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

**Table 3-223. Function dma\_channel\_enable**

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

<b>Function descriptions</b>	enable DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

**Table 3-224. Function dma\_channel\_disable**

<b>Function name</b>	dma_channel_disable
<b>Function prototype</b>	void dma_channel_disable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	disable DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA0 channel0 */
```

`dma_channel_disable(DMA0, DMA_CH0);`

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

The description of `dma_periph_address_config` is shown as below:

**Table 3-225. Function `dma_periph_address_config`**

<b>Function name</b>	<code>dma_periph_address_config</code>
<b>Function prototype</b>	<code>void dma_periph_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);</code>
<b>Function descriptions</b>	set DMA peripheral base address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>address</b>	peripheral base address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

The description of `dma_memory_address_config` is shown as below:

**Table 3-226. Function `dma_memory_address_config`**

<b>Function name</b>	<code>dma_memory_address_config</code>
<b>Function prototype</b>	<code>void dma_memory_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);</code>
<b>Function descriptions</b>	set DMA memory base address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral

<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>address</b>	memory base address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

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

**Table 3-227. Function dma\_transfer\_number\_config**

<b>Function name</b>	dma_transfer_number_config
<b>Function prototype</b>	void dma_transfer_number_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t number);
<b>Function descriptions</b>	set the number of remaining data to be transferred by the DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>number</b>	data transfer number (0x00000000 – 0x0000FFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
#define TRANSFER_NUM
```

```
0x400
```

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

## dma\_transfer\_number\_get

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

**Table 3-228. Function dma\_transfer\_number\_get**

<b>Function name</b>	dma_transfer_number_get
<b>Function prototype</b>	uint32_t dma_transfer_number_get(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	get the number of remaining data to be transferred by the DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	0x00000000 – 0x0000FFFF

Example:

```
uint32_t number = 0;
```

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

## dma\_priority\_config

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

**Table 3-229. Function dma\_priority\_config**

<b>Function name</b>	dma_priority_config
<b>Function prototype</b>	void dma_priority_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t priority);
<b>Function descriptions</b>	configure priority level of DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	



<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>priority</b>	priority Level of this channel
<i>DMA_PRIORITY_LOW</i>	low priority
<i>DMA_PRIORITY_MEDIUM</i>	medium priority
<i>DMA_PRIORITY_HIGH</i>	high priority
<i>DMA_PRIORITY_ULTRA_HIGH</i>	ultra high priority
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

**Table 3-230. Function dma\_memory\_width\_config**

<b>Function name</b>	dma_memory_width_config
<b>Function prototype</b>	void dma_memory_width_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t mwidth);
<b>Function descriptions</b>	configure transfer data size of memory
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>mwidth</b>	transfer data width of memory
<i>DMA_MEMORY_WIDTH_8BIT</i>	transfer data width of memory is 8-bit
<i>DMA_MEMORY_WIDTH_16BIT</i>	transfer data width of memory is 16-bit
<i>DMA_MEMORY_WIDTH_32BIT</i>	transfer data width of memory is 32-bit

<i>H_32BIT</i>	
<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 3-231. 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(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>pwidth</b>	transfer data width of peripheral
<i>DMA_PERIPHERAL_WIDTH_8BIT</i>	transfer data width of peripheral is 8-bit
<i>DMA_PERIPHERAL_WIDTH_16BIT</i>	transfer data width of peripheral is 16-bit
<i>DMA_PERIPHERAL_WIDTH_32BIT</i>	transfer data width of peripheral is 32-bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## dma\_memory\_increase\_enable

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

**Table 3-232. Function dma\_memory\_increase\_enable**

<b>Function name</b>	dma_memory_increase_enable
<b>Function prototype</b>	void dma_memory_increase_enable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	enable next address increasement algorithm of memory
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
dma_memory_increase_enable(DMA0, DMA_CH0);
```

## dma\_memory\_increase\_disable

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

**Table 3-233. Function dma\_memory\_increase\_disable**

<b>Function name</b>	dma_memory_increase_disable
<b>Function prototype</b>	void dma_memory_increase_disable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	disable next address increasement algorithm of memory
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_memory_increase_disable(DMA0, DMA_CH0);
```

### dma\_periph\_increase\_enable

The description of dma\_periph\_increase\_enable is shown as below:

**Table 3-234. Function dma\_periph\_increase\_enable**

Function name	dma_periph_increase_enable
Function prototype	void dma_periph_increase_enable(uint32_t dma_periph, dma_channel_enum channelx);
Function descriptions	enable next address increasement algorithm of peripheral
Precondition	-
The called functions	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_periph_increase_enable(DMA0, DMA_CH0);
```

### dma\_periph\_increase\_disable

The description of dma\_periph\_increase\_disable is shown as below:

**Table 3-235. Function dma\_periph\_increase\_disable**

Function name	dma_periph_increase_disable
Function prototype	void dma_periph_increase_disable(uint32_t dma_periph, dma_channel_enum channelx);
Function descriptions	disable next address increasement algorithm of peripheral
Precondition	-
The called functions	-

Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_periph_increase_disable(DMA0, DMA_CH0);
```

## dma\_transfer\_direction\_config

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

**Table 3-236. Function dma\_transfer\_direction\_config**

<b>Function name</b>	dma_transfer_direction_config
<b>Function prototype</b>	void dma_transfer_direction_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t direction);
<b>Function descriptions</b>	configure the direction of data transfer on the channel
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
Input parameter{in}	
<b>direction</b>	specify the direction of data transfer
<i>DMA_PERIPHERAL_TO_MEMORY</i>	read from peripheral and write to memory
<i>DMA_MEMORY_TO_PERIPHERAL</i>	read from memory and write to peripheral
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

## **dma\_flag\_get**

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

**Table 3-237. Function dma\_flag\_get**

<b>Function name</b>	dma_flag_get
<b>Function prototype</b>	FlagStatus dma_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
<b>Function descriptions</b>	check DMA flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>DMA_FLAG_G</i>	global interrupt flag of channel
<i>DMA_FLAG_FTF</i>	full transfer finish flag of channel
<i>DMA_FLAG_HTF</i>	half transfer finish flag of channel
<i>DMA_FLAG_ERR</i>	error flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
FlagStatus flag = RESET;
```

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

## **dma\_flag\_clear**

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

**Table 3-238. Function dma\_flag\_clear**

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

<b>Function descriptions</b>	clear the flag of a DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>DMA_FLAG_G</i>	global interrupt flag of channel
<i>DMA_FLAG_FTF</i>	full transfer finish flag of channel
<i>DMA_FLAG_HTF</i>	half transfer finish flag of channel
<i>DMA_FLAG_ERR</i>	error flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## dma\_interrupt\_enable

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

**Table 3-239. Function dma\_interrupt\_enable**

<b>Function name</b>	dma_interrupt_enable
<b>Function prototype</b>	void dma_interrupt_enable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
<b>Function descriptions</b>	enable DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>source</b>	DMA interrupt source

<i>DMA_INT_FTF</i>	full transfer finish interrupt of channel
<i>DMA_INT_HTF</i>	half transfer finish interrupt of channel
<i>DMA_INT_ERR</i>	error interrupt of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* DMA0 channel0 interrupt configuration */
```

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

### **dma\_interrupt\_disable**

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

**Table 3-240. Function dma\_interrupt\_disable**

<b>Function name</b>	dma_interrupt_disable
<b>Function prototype</b>	void dma_interrupt_disable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
<b>Function descriptions</b>	disable DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>source</b>	DMA interrupt source
<i>DMA_INT_FTF</i>	full transfer finish interrupt of channel
<i>DMA_INT_HTF</i>	half transfer finish interrupt of channel
<i>DMA_INT_ERR</i>	error interrupt of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* DMA0 channel0 interrupt configuration */
```

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



## dma\_interrupt\_flag\_get

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

**Table 3-241. Function dma\_interrupt\_flag\_get**

<b>Function name</b>	dma_interrupt_flag_get
<b>Function prototype</b>	FlagStatus dma_interrupt_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
<b>Function descriptions</b>	check DMA flag and interrupt enable bit is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>DMA_INT_FLAG_FTF</i>	full transfer finish interrupt flag of channel
<i>DMA_INT_FLAG_HTF</i>	half transfer finish interrupt flag of channel
<i>DMA_INT_FLAG_ERR</i>	error interrupt flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
if(dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_G);
}
```

## dma\_interrupt\_flag\_clear

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

**Table 3-242. Function dma\_interrupt\_flag\_clear**

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

Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to <a href="#">Table 3-214. Enum dma channel enum</a>
Input parameter{in}	
<b>flag</b>	specify get which flag
<i>DMA_INT_FLAG_G</i>	global interrupt flag of channel
<i>DMA_INT_FLAG_FTF</i>	full transfer finish interrupt flag of channel
<i>DMA_INT_FLAG_HTF</i>	half transfer finish interrupt flag of channel
<i>DMA_INT_FLAG_ERR</i>	error interrupt flag of channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
if(dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_G);
}
```

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

### 3.11.1. Descriptions of Peripheral registers

ENET registers are listed in the table shown as below:

**Table 3-243. 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_	MAC PHY control register

Registers	Descriptions
CTL	
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
ENET_MAC_ADDR1H	MAC address 1 high register
ENET_MAC_ADDR1L	MAC address 1 low register
ENET_MAC_ADDT2H	MAC address 2 high register
ENET_MAC_ADDR2L	MAC address 2 low register
ENET_MAC_ADDR3H	MAC address 3 high register
ENET_MAC_ADDR3L	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_RINTMSK	MSC receive interrupt mask register
ENET_MSC_TINTMSK	MSC transmit interrupt mask register
ENET_MSC_SCCNT	MSC transmitted good frames after a single collision counter register
ENET_MSC_MSCCNT	MSC transmitted good frames after more than a single collision counter register
ENET_MSC_TGFCNT	MSC transmitted good frames counter register
ENET_MSC_RFCE	MSC received frames with CRC error counter register

Registers	Descriptions
CNT	
ENET_MSC_RFAE CNT	MSC received frames with alignment error counter register
ENET_MSC_RGUF CNT	MSC received good unicast frames counter register
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_PPSCT L	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 DDR	DMA current receive descriptor address register
ENET_DMA_CTBA DDR	DMA current transmit buffer address register
ENET_DMA_CRBA DDR	DMA current receive buffer address register

### 3.11.2. Descriptions of Peripheral functions

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

**Table 3-244. 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
enet_flag_get	get the ENET MAC/MSC/PTP/DMA status flag
enet_flag_clear	clear the ENET DMA status flag
enet_interrupt_enable	enable ENET MAC/MSC/DMA interrupt
enet_interrupt_disable	disable ENET MAC/MSC/DMA interrupt
enet_interrupt_flag_get	get ENET MAC/MSC/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

Function name	Function description
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
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	

Function name	Function description
enet_rx_desc_enhanced_status_get	get the bit of extended status flag in ENET DMA descriptor
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
PTP function	
enet_ptp_subsecond_2_nanosecond	change subsecond to nanosecond
enet_ptp_nanosecond_2_subsecond	change nanosecond to subsecond
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

Function name	Function description
	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
enet_ptp_start	configure and start PTP timestamp counter
enet_ptp_finecorrection_adjfreq	adjust frequency in fine method by configure addend register
enet_ptp_coarsecorrection_systime_update	update system time in coarse method
enet_ptp_finecorrection_settime	set system time in fine method
enet_ptp_flag_get	get the ptp flag status
internal function	
enet_initpara_reset	reset the ENET initpara struct, call it before using enet_initpara_config()

### Structure enet\_initpara\_struct

**Table 3-245. 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 3-246. 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 3-247. Structure enet\_ptp\_systime\_struct**

member name	Function description
second	second of system time
nanosecond	nanosecond of system time
sign	sign of system time

## Enum enet\_flag\_enum

**Table 3-248. Enum enet\_flag\_enum**

member name	Function description
ENET_MAC_FLAG_MPKR	magic packet received flag
ENET_MAC_FLAG_WUFR	wakeup frame received flag
ENET_MAC_FLAG_FLOWCONTROL	flow control status flag
ENET_MAC_FLAG_WUM	WUM status flag
ENET_MAC_FLAG_MSC	MSC status flag
ENET_MAC_FLAG_MSCR	MSC receive status flag
ENET_MAC_FLAG_MSCT	MSC transmit status flag
ENET_MAC_FLAG_TMST	timestamp trigger status flag
ENET_PTP_FLAG_TSSCO	timestamp second counter overflow flag
ENET_PTP_FLAG_TTM	target time match flag
ENET_MSC_FLAG_RFCE	received frames CRC error flag
ENET_MSC_FLAG_RFAE	received frames alignment error flag
ENET_MSC_FLAG_	received good unicast frames flag

RGUF	
ENET_MSC_FLAG_ TGFSC	transmitted good frames single collision flag
ENET_MSC_FLAG_ TGFMSC	transmitted good frames more single collision flag
ENET_MSC_FLAG_ TGF	transmitted good frames flag
ENET_DMA_FLAG_ TS	transmit status flag
ENET_DMA_FLAG_ TPS	transmit process stopped status flag
ENET_DMA_FLAG_ TBU	transmit buffer unavailable status flag
ENET_DMA_FLAG_ TJT	transmit jabber timeout status flag
ENET_DMA_FLAG_ RO	receive overflow status flag
ENET_DMA_FLAG_ TU	transmit underflow status flag
ENET_DMA_FLAG_ RS	receive status flag
ENET_DMA_FLAG_ RBU	receive buffer unavailable status flag
ENET_DMA_FLAG_ RPS	receive process stopped status flag
ENET_DMA_FLAG_ RWT	receive watchdog timeout status flag
ENET_DMA_FLAG_ ET	early transmit status flag
ENET_DMA_FLAG_ FBE	fatal bus error status flag
ENET_DMA_FLAG_ ER	early receive status flag
ENET_DMA_FLAG_ AI	abnormal interrupt summary flag
ENET_DMA_FLAG_ NI	normal interrupt summary flag
ENET_DMA_FLAG_ EB_DMA_ERROR	error during data transfer by RxDMA/TxDMA flag
ENET_DMA_FLAG_ EB_TRANSFER_E RROR	error during write/read transfer flag
ENET_DMA_FLAG_ 	error during data buffer/descriptor access flag

EB_ACCESS_ERR OR	
ENET_DMA_FLAG_ MSC	MSC status flag
ENET_DMA_FLAG_ WUM	WUM status flag
ENET_DMA_FLAG_ TST	timestamp trigger status flag

## Enum enet\_flag\_clear\_enum

**Table 3-249. Enum enet\_flag\_clear\_enum**

member name	Function description
ENET_DMA_FLAG_ TS_CLR	transmit status flag clear
ENET_DMA_FLAG_ TPS_CLR	transmit process stopped status flag clear
ENET_DMA_FLAG_ TBU_CLR	transmit buffer unavailable status flag clear
ENET_DMA_FLAG_ TJT_CLR	transmit jabber timeout status flag clear
ENET_DMA_FLAG_ RO_CLR	receive overflow status flag clear
ENET_DMA_FLAG_ TU_CLR	transmit underflow status flag clear
ENET_DMA_FLAG_ RS_CLR	receive status flag clear
ENET_DMA_FLAG_ RBU_CLR	receive buffer unavailable status flag clear
ENET_DMA_FLAG_ RPS_CLR	receive process stopped status flag clear
ENET_DMA_FLAG_ RWT_CLR	receive watchdog timeout status flag clear
ENET_DMA_FLAG_ ET_CLR	early transmit status flag clear
ENET_DMA_FLAG_ FBE_CLR	fatal bus error status flag clear
ENET_DMA_FLAG_ ER_CLR	early receive status flag clear
ENET_DMA_FLAG_ AI_CLR	abnormal interrupt summary flag clear
ENET_DMA_FLAG_ NI_CLR	normal interrupt summary flag clear

## Enum enet\_int\_enum

**Table 3-250. Enum enet\_int\_enum**

member name	Function description
<i>ENET_MAC_INT_WUMIM</i>	WUM interrupt mask
<i>ENET_MAC_INT_TSTMIM</i>	timestamp trigger interrupt mask
<i>ENET_MSC_INT_RFCEIM</i>	received frame CRC error interrupt mask
<i>ENET_MSC_INT_RFAEIM</i>	received frames alignment error interrupt mask
<i>ENET_MSC_INT_RGUFIM</i>	received good unicast frames interrupt mask
<i>ENET_MSC_INT_TGFSCIM</i>	transmitted good frames single collision interrupt mask
<i>ENET_MSC_INT_TGFMSCIM</i>	transmitted good frames more single collision interrupt mask
<i>ENET_MSC_INT_TGFIM</i>	transmitted good frames interrupt mask
<i>ENET_DMA_INT_TIE</i>	transmit interrupt enable
<i>ENET_DMA_INT_TPSIE</i>	transmit process stopped interrupt enable
<i>ENET_DMA_INT_TBUIE</i>	transmit buffer unavailable interrupt enable
<i>ENET_DMA_INT_TJTIE</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_FF</i>	fatal bus error interrupt enable

<i>BEIE</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

## Enum enet\_int\_flag\_enum

**Table 3-251. Enum enet\_int\_flag\_enum**

member name	Function description
<i>ENET_MAC_INT_FLAG_WUM</i>	WUM status flag
<i>ENET_MAC_INT_FLAG_MSC</i>	MSC status flag
<i>ENET_MAC_INT_FLAG_MSCR</i>	MSC receive status flag
<i>ENET_MAC_INT_FLAG_MSCT</i>	MSC transmit status flag
<i>ENET_MAC_INT_FLAG_TMST</i>	time stamp trigger status flag
<i>ENET_MSC_INT_FLAG_RFCE</i>	received frames CRC error flag
<i>ENET_MSC_INT_FLAG_RFAE</i>	received frames alignment error flag
<i>ENET_MSC_INT_FLAG_RGUF</i>	received good unicast frames flag
<i>ENET_MSC_INT_FLAG_TGFSC</i>	transmitted good frames single collision flag
<i>ENET_MSC_INT_FLAG_TGFMSC</i>	transmitted good frames more single collision flag
<i>ENET_MSC_INT_FLAG_TGF</i>	transmitted good frames flag
<i>ENET_DMA_INT_FLAG_TS</i>	transmit status flag
<i>ENET_DMA_INT_FLAG_TPS</i>	transmit process stopped status flag
<i>ENET_DMA_INT_FLAG_TBU</i>	transmit buffer unavailable status flag
<i>ENET_DMA_INT_FLAG_TJT</i>	transmit jabber timeout status flag
<i>ENET_DMA_INT_F</i>	receive overflow status flag

<i>LAG_RO</i>	
<i>ENET_DMA_INT_F</i> <i>LAG_TU</i>	transmit underflow status flag
<i>ENET_DMA_INT_F</i> <i>LAG_RS</i>	receive status flag
<i>ENET_DMA_INT_F</i> <i>LAG_RBU</i>	receive buffer unavailable status flag
<i>ENET_DMA_INT_F</i> <i>LAG_RPS</i>	receive process stopped status flag
<i>ENET_DMA_INT_F</i> <i>LAG_RWT</i>	receive watchdog timeout status flag
<i>ENET_DMA_INT_F</i> <i>LAG_ET</i>	early transmit status flag
<i>ENET_DMA_INT_F</i> <i>LAG_FBE</i>	fatal bus error status flag
<i>ENET_DMA_INT_F</i> <i>LAG_ER</i>	early receive status flag
<i>ENET_DMA_INT_F</i> <i>LAG_AI</i>	abnormal interrupt summary flag
<i>ENET_DMA_INT_F</i> <i>LAG_NI</i>	normal interrupt summary flag
<i>ENET_DMA_INT_F</i> <i>LAG_MSC</i>	MSC status flag
<i>ENET_DMA_INT_F</i> <i>LAG_WUM</i>	WUM status flag
<i>ENET_DMA_INT_F</i> <i>LAG_TST</i>	timestamp trigger status flag

## Enum enet\_int\_flag\_clear\_enum

**Table 3-252. Enum enet\_int\_flag\_clear\_enum**

member name	Function description
<i>ENET_DMA_INT_F</i> <i>LAG_TS_CLR</i>	transmit status flag
<i>ENET_DMA_INT_F</i> <i>LAG_TPS_CLR</i>	transmit process stopped status flag
<i>ENET_DMA_INT_F</i> <i>LAG_TBU_CLR</i>	transmit buffer unavailable status flag
<i>ENET_DMA_INT_F</i> <i>LAG_TJT_CLR</i>	transmit jabber timeout status flag
<i>ENET_DMA_INT_F</i> <i>LAG_RO_CLR</i>	receive overflow status flag
<i>ENET_DMA_INT_F</i>	transmit underflow status flag

<i>LAG_TU_CLR</i>	
<i>ENET_DMA_INT_F</i> <i>LAG_RS_CLR</i>	receive status flag
<i>ENET_DMA_INT_F</i> <i>LAG_RBU_CLR</i>	receive buffer unavailable status flag
<i>ENET_DMA_INT_F</i> <i>LAG_RPS_CLR</i>	receive process stopped status flag
<i>ENET_DMA_INT_F</i> <i>LAG_RWT_CLR</i>	receive watchdog timeout status flag
<i>ENET_DMA_INT_F</i> <i>LAG_ET_CLR</i>	early transmit status flag
<i>ENET_DMA_INT_F</i> <i>LAG_FBE_CLR</i>	fatal bus error status flag
<i>ENET_DMA_INT_F</i> <i>LAG_ER_CLR</i>	early receive status flag
<i>ENET_DMA_INT_F</i> <i>LAG_AI_CLR</i>	abnormal interrupt summary flag
<i>ENET_DMA_INT_F</i> <i>LAG_NI_CLR</i>	normal interrupt summary flag

## Enum enet\_desc\_reg\_enum

**Table 3-253. Enum enet\_desc\_reg\_enum**

member name	Function description
<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

## Enum enet\_msc\_counter\_enum

**Table 3-254. Enum enet\_msc\_counter\_enum**

member name	Function description
<i>ENET_MSC_TX_SINGLE_COLL_CNT</i>	MSC transmitted good frames after a single collision counter
<i>ENET_MSC_TX_MULT_COLL_CNT</i>	MSC transmitted good frames after more than a single collision counter

<i>SCCNT</i>	
<i>ENET_MSC_TX_TGFCNT</i>	MSC transmitted good frames counter
<i>ENET_MSC_RX_RFCECNT</i>	MSC received frames with CRC error counter
<i>ENET_MSC_RX_RFAECNT</i>	MSC received frames with alignment error counter
<i>ENET_MSC_RX_RGUFCNT</i>	MSC received good unicast frames counter

## Enum enet\_option\_enum

**Table 3-255. Enum enet\_option\_enum**

member name	Function description
<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_OPTION</i>	choose to configure the DMA arbitration related parameters
<i>STORE_OPTION</i>	choose to configure the store forward mode related parameters
<i>DMA_OPTION</i>	choose to configure the DMA related parameters
<i>VLAN_OPTION</i>	choose to configure vlan related parameters
<i>FLOWCTL_OPTION</i>	choose to configure flow control related parameters
<i>HASHH_OPTION</i>	choose to configure hash high
<i>HASHL_OPTION</i>	choose to configure hash low
<i>FILTER_OPTION</i>	choose to configure frame filter related parameters
<i>HALFDUPLEX_OPTION</i>	choose to configure halfduplex mode related parameters
<i>TIMER_OPTION</i>	choose to configure time counter related parameters
<i>INTERFRAMEGAP_OPTION</i>	choose to configure the inter frame gap related parameters

## Enum enet\_mediamode\_enum

**Table 3-256. Enum enet\_mediamode\_enum**

member name	Function description
<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

## Enum enet\_chksumconf\_enum

**Table 3-257. Enum enet\_chksumconf\_enum**

member name	Function description
<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

## Enum enet\_frmrecept\_enum

**Table 3-258. Enum enet\_frmrecept\_enum**

member name	Function description
<i>ENET_PROMISCUOUS_MODE</i>	promiscuous mode enabled
<i>ENET_RECEIVEALL</i>	all received frame are forwarded to application
<i>ENET_BROADCAST_FRAMES_PASS</i>	the address filters pass all received broadcast frames
<i>ENET_BROADCAST_FRAMES_DROP</i>	the address filters filter all incoming broadcast frames

## Enum enet\_registers\_type\_enum

**Table 3-259. Enum enet\_registers\_type\_enum**

member name	Function description
<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

## Enum enet\_dmadirection\_enum

**Table 3-260. Enum enet\_dmadirection\_enum**

member name	Function description
<i>ENET_DMA_TX</i>	DMA Tx descriptors
<i>ENET_DMA_RX</i>	DMA Rx descriptors

## Enum enet\_phydirection\_enum

**Table 3-261. Enum enet\_phydirection\_enum**

member name	Function description
<i>ENET_PHY_WRITE</i>	write data to phy register
<i>ENET_PHY_READ</i>	read data from phy register

## Enum enet\_regdirection\_enum

**Table 3-262. Enum enet\_regdirection\_enum**

member name	Function description
<i>ENET_REG_READ</i>	read register
<i>ENET_REG_WRITE</i>	write register

## Enum enet\_macaddress\_enum

**Table 3-263. Enum enet\_macaddress\_enum**

member name	Function description
<i>ENET_MAC_ADDR</i> <i>ESS0</i>	set MAC address 0 filter
<i>ENET_MAC_ADDR</i> <i>ESS1</i>	set MAC address 1 filter
<i>ENET_MAC_ADDR</i> <i>ESS2</i>	set MAC address 2 filter
<i>ENET_MAC_ADDR</i> <i>ESS3</i>	set MAC address 3 filter

## Enum enet\_descstate\_enum

**Table 3-264. Enum enet\_descstate\_enum**

member name	Function description
<i>TXDESC_COLLISION_COUNT</i>	the number of collisions occurred before the frame was transmitted
<i>TXDESC_BUFFER_1_ADDR</i>	the buffer1 address of the Tx frame
<i>RXDESC_FRAME_</i>	the byte length of the received frame that was transferred to the buffer

<i>LENGTH</i>	
<i>RXDESC_BUFFER</i> <i>_1_SIZE</i>	receive buffer 1 size
<i>RXDESC_BUFFER</i> <i>_2_SIZE</i>	receive buffer 2 size
<i>RXDESC_BUFFER</i> <i>_1_ADDR</i>	the buffer1 address of the Rx frame

### Enum `enet_msc_preset_enum`

**Table 3-265. Enum `enet_msc_preset_enum`**

member name	Function description
<i>ENET_MSC_PRES</i> <i>ET_NONE</i>	do not preset MSC counter
<i>ENET_MSC_PRES</i> <i>ET_HALF</i>	preset all MSC counters to almost-half(0x7FFF FFF0) value
<i>ENET_MSC_PRES</i> <i>ET_FULL</i>	preset all MSC counters to almost-full(0xFFFF FFF0) value

### `enet_deinit`

The description of `enet_deinit` is shown as below:

**Table 3-266. Function `enet_deinit`**

<b>Function name</b>	<code>enet_deinit</code>
<b>Function prototype</b>	<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>	<code>rcu_periph_reset_enable()</code> / <code>rcu_periph_reset_disable()</code> / <code>enet_initpara_reset()</code>
<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 3-267. 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, refer to <a href="#">Table 3-255. Enum enet_option_enum</a> only one parameter can be selected which is shown as below
FORWARD_OPTION	choose to configure the frame forward related parameters
DMABUS_OPTION	choose to configure the DMA bus mode related parameters
DMA_MAXBURST_OPTION	choose to configure the DMA max burst related parameters
DMA_ARBITRATION_OPTION	choose to configure the DMA arbitration related parameters
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
<b>Input parameter{in}</b>	
<b>para</b> (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 <i>DMABUS_OPTION</i>	
value1	<i>ENET_ADDRESS_ALIGN_ENABLE / ENET_ADDRESS_ALIGN_DISABLE</i>
value2	<i>ENET_FIXED_BURST_ENABLE / ENET_FIXED_BURST_DISABLE</i>
value3	<i>ENET_MIXED_BURST_ENABLE / ENET_MIXED_BURST_DISABLE</i>
When value of parameter <b>option</b> is <i>DMA_MAXBURST_OPTION</i>	
value1	<i>ENET_RXDP_1BEAT / ENET_RXDP_2BEAT / ENET_RXDP_4BEAT / 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	ENET_ZERO_QUANTA_PAUSE_ENABLE / ENET_ZERO_QUANTA_PAUSE_DISABLE
value3	ENET_PAUSETIME_MINUS4 / ENET_PAUSETIME_MINUS28 / ENET_PAUSETIME_MINUS144/ENET_PAUSETIME_MINUS256
value4	ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDTECT / ENET_UNIQUE_PAUSEDTECT
value5	ENET_RX_FLOWCONTROL_ENABLE / ENET_RX_FLOWCONTROL_DISABLE
value6	ENET_TX_FLOWCONTROL_ENABLE / ENET_TX_FLOWCONTROL_DISABLE
When value of parameter <b>option</b> is HASHH_OPTION	
value1	0x0~0xFFFF FFFFU
When value of parameter <b>option</b> is HASHL_OPTION	
value1	0x0~0xFFFF FFFFU
When value of parameter <b>option</b> is FILTER_OPTION	
value1	ENET_SRC_FILTER_NORMAL_ENABLE / ENET_SRC_FILTER_INVERSE_ENABLE / ENET_SRC_FILTER_DISABLE
value2	ENET_DEST_FILTER_INVERSE_ENABLE / ENET_DEST_FILTER_INVERSE_DISABLE
value3	ENET_MULTICAST_FILTER_HASH_OR_PERFECT / ENET_MULTICAST_FILTER_HASH / ENET_MULTICAST_FILTER_PERFECT / ENET_MULTICAST_FILTER_NONE
value4	ENET_UNICAST_FILTER_EITHER / ENET_UNICAST_FILTER_HASH / ENET_UNICAST_FILTER_PERFECT
value5	ENET_PCFRM_PREVENT_ALL / ENET_PCFRM_PREVENT_PAUSEFRAME / ENET_PCFRM_FORWARD_ALL / ENET_PCFRM_FORWARD_FILTERED
When value of parameter <b>option</b> is HALFDUPLEX_OPTION	
value1	ENET_CARRIERSENSE_ENABLE / ENET_CARRIERSENSE_DISABLE
value2	ENET_RECEIVEOWN_ENABLE / ENET_RECEIVEOWN_DISABLE
value3	ENET_RETRYTRANSMISSION_ENABLE / ENET_RETRYTRANSMISSION_DISABLE
value4	ENET_BACKOFFLIMIT_10 / ENET_BACKOFFLIMIT_8 / ENET_BACKOFFLIMIT_4 / ENET_BACKOFFLIMIT_1
value5	ENET_DEFERRALCHECK_ENABLE / ENET_DEFERRALCHECK_DISABLE
When value of parameter <b>option</b> is TIMER_OPTION	
value1	ENET_WATCHDOG_ENABLE / ENET_WATCHDOG_DISABLE
value2	ENET_JABBER_ENABLE / ENET_JABBER_DISABLE
When value of parameter <b>option</b> is INTERFRAMEGAP_OPTION	
value1	ENET_INTERFRAMEGAP_96BIT / ENET_INTERFRAMEGAP_88BIT / ENET_INTERFRAMEGAP_80BIT / ENET_INTERFRAMEGAP_72BIT /

	<i>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 3-268. 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 ()
<b>The called functions</b>	enet_phy_config /enet_phy_write_read
<b>Input parameter{in}</b>	
<b>mediamode</b>	PHY mode and mac loopback configurations, refer to <a href="#">Table 3-256. Enum enet_mediamode_enum</a> , 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
<b>Input parameter{in}</b>	
<b>checksum</b>	IP frame checksum offload function, refer to <a href="#">Table 3-257. Enum enet_chksumconf_enum</a> , only one parameter can be selected

<i>ENET_NO_AUTOCHESUM</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
<b>Input parameter{in}</b>	
<b>recept</b>	frame filter function, refer to <a href="#">Table 3-258. Enum enet_frmrecept_enum</a> , 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_FRAMES_PASS</i>	the address filters pass all received broadcast frames
<i>ENET_BROADCAST_FRAMES_DROP</i>	the address filters filter all incoming broadcast frames
<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 3-269. 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>	
-	-



Return value	
ErrStatus	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 3-270. Function enet\_rxframe\_size\_get**

Function name	enet_rxframe_size_get
Function prototype	uint32_t enet_rxframe_size_get(void);
Function descriptions	check receive frame valid and return frame size
Precondition	-
The called functions	enet_rxframe_drop()
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
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 3-271. Function enet\_descriptors\_chain\_init**

Function name	enet_descriptors_chain_init
Function prototype	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, refer to <a href="#">Table 3-260. Enum enet_dmadirection_enum</a> 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 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 3-272. 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, refer to <a href="#">Table 3-260. Enum enet_dmadirection_enum</a> 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 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 3-273. 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 3-274. 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 3-275. Function enet\_transmit\_checksum\_config**

<b>Function name</b>	enet_transmit_checksum_config
<b>Function prototype</b>	ErrStatus 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="#">Table 3-246. 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
ENET_CHECKSUM_DISABLE	checksum insertion disabled
ENET_CHECKSUM_IPV4HEADER	only IP header checksum calculation and insertion are enabled
ENET_CHECKSUM_TCPUDPICMP_SEGMENT	TCP/UDP/ICMP checksum insertion calculated but pseudo-header
ENET_CHECKSUM_TCPUDPICMP_FULL	TCP/UDP/ICMP checksum insertion fully calculated
<b>Output parameter{out}</b>	
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

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_FULL);
```

## enet\_enable

The description of enet\_enable is shown as below:

**Table 3-276. 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 3-277. 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 3-278. 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, refer to <a href="#">Table 3-263. Enum enet_macaddress_enum</a> 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>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 3-279. Function enet\_mac\_address\_get**

Function name	enet_mac_address_get
Function prototype	ErrStatus enet_mac_address_get(enet_macaddress_enum mac_addr, uint8_t paddr[], uint8_t bufsize);
Function descriptions	get MAC address
Precondition	-
The called functions	-
Input parameter{in}	
mac_addr	select which MAC address will be set, refer to <a href="#">Table 3-263. Enum enet_macaddress_enum</a> 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
Output parameter{out}	
paddr	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}
Input parameter{in}	
bufsize	refer to the size of the buffer which stores the MAC address
Return value	
-	-

Example:

```
/* get mac address */
```

```
enet_mac_address_get (ENET_MAC_ADDRESS0, netif->hwaddr, 0x100);
```

## enet\_flag\_get

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

**Table 3-280. Function enet\_flag\_get**

Function name	enet_flag_get
---------------	---------------

<b>Function prototype</b>	FlagStatus enet_flag_get(enet_flag_enum enet_flag);
<b>Function descriptions</b>	get the ENET MAC/MSR/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 <a href="#">Table 3-248. Enum enet_flag_enum</a> only one parameter can be selected which is shown as below
ENET_MAC_FLAG_MPKR	magic packet received flag
ENET_MAC_FLAG_WUFR	wakeup frame received flag
ENET_MAC_FLAG_FLOWCONTROL	flow control status flag
ENET_MAC_FLAG_WUM	WUM status flag
ENET_MAC_FLAG_MSC	MSC status flag
ENET_MAC_FLAG_MSCR	MSC receive status flag
ENET_MAC_FLAG_MSCCT	MSC transmit status flag
ENET_MAC_FLAG_TMST	time stamp trigger status flag
ENET_PTP_FLAG_TSSCO	timestamp second counter overflow flag
ENET_PTP_FLAG_TTM	target time match flag
ENET_MSC_FLAG_RFCR	received frames CRC error flag
ENET_MSC_FLAG_RFAE	received frames alignment error flag
ENET_MSC_FLAG_RGUF	received good unicast frames flag
ENET_MSC_FLAG_TGFSC	transmitted good frames single collision flag
ENET_MSC_FLAG_TGFMSC	transmitted good frames more single collision flag
ENET_MSC_FLAG_TGF	transmitted good frames flag
ENET_DMA_FLAG_TST	transmit status flag
ENET_DMA_FLAG_TPS	transmit process stopped status flag
ENET_DMA_FLAG_TBF	transmit buffer unavailable status flag



<i>U</i>	
<i>ENET_DMA_FLAG_TJ</i> <i>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</i> <i>U</i>	receive buffer unavailable status flag
<i>ENET_DMA_FLAG_RP</i> <i>S</i>	receive process stopped status flag
<i>ENET_DMA_FLAG_R</i> <i>WT</i>	receive watchdog timeout status flag
<i>ENET_DMA_FLAG_ET</i>	early transmit status flag
<i>ENET_DMA_FLAG_FB</i> <i>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</i> <i>_DMA_ERROR</i>	DMA error flag
<i>ENET_DMA_FLAG_EB</i> <i>_TRANSFER_ERROR</i>	transfer error flag
<i>ENET_DMA_FLAG_EB</i> <i>_ACCESS_ERROR</i>	access error flag
<i>ENET_DMA_FLAG_MS</i> <i>C</i>	MSC status flag
<i>ENET_DMA_FLAG_W</i> <i>UM</i>	WUM status flag
<i>ENET_DMA_FLAG_TS</i> <i>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 3-281. 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 <a href="#">Table 3-249. Enum enet_flag_clear_enum</a> only one parameter can be selected which is shown as below
ENET_DMA_FLAG_TS_CLR	transmit status flag clear
ENET_DMA_FLAG_TPS_CLR	transmit process stopped status flag clear
ENET_DMA_FLAG_TB_U_CLR	transmit buffer unavailable status flag clear
ENET_DMA_FLAG_TJT_CLR	transmit jabber timeout status flag clear
ENET_DMA_FLAG_RO_CLR	receive overflow status flag clear
ENET_DMA_FLAG_TU_CLR	transmit underflow status flag clear
ENET_DMA_FLAG_RS_CLR	receive status flag clear
ENET_DMA_FLAG_RBU_CLR	receive buffer unavailable status flag clear
ENET_DMA_FLAG_RPS_CLR	receive process stopped status flag clear
ENET_DMA_FLAG_RWT_CLR	receive watchdog timeout status flag clear
ENET_DMA_FLAG_ET_CLR	early transmit status flag clear
ENET_DMA_FLAG_FBE_CLR	fatal bus error status flag clear
ENET_DMA_FLAG_ER_CLR	early receive status flag clear
ENET_DMA_FLAG_AICLR	abnormal interrupt summary flag clear
ENET_DMA_FLAG_NICLR	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 3-282. Function enet\_interrupt\_enable**

Function name	enet_interrupt_enable
Function prototype	void enet_interrupt_enable(enet_int_enum enet_int);
Function descriptions	enable ENET MAC/MSD/DMA interrupt
Precondition	-
The called functions	-
Input parameter{in}	
enet_int	ENET interrupt, refer to <a href="#">Table 3-250. Enum enet_int_enum</a> only one parameter can be selected which is shown as below
ENET_MAC_INT_WUMIM	WUM interrupt mask
ENET_MAC_INT_TMSIM	timestamp trigger interrupt mask
ENET_MSC_INT_RFCIM	received frame CRC error interrupt mask
ENET_MSC_INT_RFAIM	received frames alignment error interrupt mask
ENET_MSC_INT_RGUFIM	received good unicast frames interrupt mask
ENET_MSC_INT_TGFSIM	transmitted good frames single collision interrupt mask
ENET_MSC_INT_TGFMSIM	transmitted good frames more single collision interrupt mask
ENET_MSC_INT_TGFMIM	transmitted good frames interrupt mask
ENET_DMA_INT_TIE	transmit interrupt enable
ENET_DMA_INT_TPSIE	transmit process stopped interrupt enable
ENET_DMA_INT_TBUIE	transmit buffer unavailable interrupt enable
ENET_DMA_INT_TJTIE	transmit jabber timeout interrupt enable
ENET_DMA_INT_ROIE	receive overflow interrupt enable
ENET_DMA_INT_TUIE	transmit underflow interrupt enable
ENET_DMA_INT_RIE	receive interrupt enable

<i>ENET_DMA_INT_RBUI</i> <i>E</i>	receive buffer unavailable interrupt enable
<i>ENET_DMA_INT_RPSI</i> <i>E</i>	receive process stopped interrupt enable
<i>ENET_DMA_INT_RWT</i> <i>IE</i>	receive watchdog timeout interrupt enable
<i>ENET_DMA_INT_ETIE</i>	early transmit interrupt enable
<i>ENET_DMA_INT_FBEI</i> <i>E</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 3-283. 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/MSC/DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>enet_int</b>	ENET interrupt, refer to <a href="#">Table 3-250. Enum enet_int_enum</a> only one parameter can be selected which is shown as below
<i>ENET_MAC_INT_WUM</i> <i>IM</i>	WUM interrupt mask
<i>ENET_MAC_INT_TMS</i> <i>TIM</i>	timestamp trigger interrupt mask
<i>ENET_MSC_INT_RFC</i> <i>EIM</i>	received frame CRC error interrupt mask
<i>ENET_MSC_INT_RFA</i> <i>EIM</i>	received frames alignment error interrupt mask
<i>ENET_MSC_INT_RGU</i>	received good unicast frames interrupt mask

<i>FIM</i>	
<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 E</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:

```
/* 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 3-284. Function `enet_interrupt_flag_get`**

<b>Function name</b>	<code>enet_interrupt_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus enet_interrupt_flag_get(enet_int_flag_enum int_flag);</code>
<b>Function descriptions</b>	get ENET MAC/MSC/DMA interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	ENET interrupt flag, refer to <a href="#">Table 3-251. Enum <code>enet_int_flag_enum</code></a> only one parameter can be selected which is shown as below
<code>ENET_MAC_INT_FLAG_WUM</code>	WUM status flag
<code>ENET_MAC_INT_FLAG_MSC</code>	MSC status flag
<code>ENET_MAC_INT_FLAG_MSCR</code>	MSC receive status flag
<code>ENET_MAC_INT_FLAG_MSCT</code>	MSC transmit status flag
<code>ENET_MAC_INT_FLAG_TMST</code>	time stamp trigger status flag
<code>ENET_MSC_INT_FLAG_RFCE</code>	received frames CRC error flag
<code>ENET_MSC_INT_FLAG_RFAE</code>	received frames alignment error flag
<code>ENET_MSC_INT_FLAG_RGUF</code>	received good unicast frames flag
<code>ENET_MSC_INT_FLAG_TGFSC</code>	transmitted good frames single collision flag
<code>ENET_MSC_INT_FLAG_TGFMSC</code>	transmitted good frames more single collision flag
<code>ENET_MSC_INT_FLAG_TGF</code>	transmitted good frames flag
<code>ENET_DMA_INT_FLAG_TS</code>	transmit status flag
<code>ENET_DMA_INT_FLAG_TPS</code>	transmit process stopped status flag
<code>ENET_DMA_INT_FLAG_TBU</code>	transmit buffer unavailable status flag
<code>ENET_DMA_INT_FLAG_TJT</code>	transmit jabber timeout status flag
<code>ENET_DMA_INT_FLAG_RO</code>	receive overflow status flag
<code>ENET_DMA_INT_FLAG_TU</code>	transmit underflow status flag

<i>ENET_DMA_INT_FLAG_RS</i>	receive status flag
<i>ENET_DMA_INT_FLAG_RBU</i>	receive buffer unavailable status flag
<i>ENET_DMA_INT_FLAG_RPS</i>	receive process stopped status flag
<i>ENET_DMA_INT_FLAG_RWT</i>	receive watchdog timeout status flag
<i>ENET_DMA_INT_FLAG_ET</i>	early transmit status flag
<i>ENET_DMA_INT_FLAG_FBE</i>	fatal bus error status flag
<i>ENET_DMA_INT_FLAG_ER</i>	early receive status flag
<i>ENET_DMA_INT_FLAG_AI</i>	abnormal interrupt summary flag
<i>ENET_DMA_INT_FLAG_NI</i>	normal interrupt summary flag
<i>ENET_DMA_INT_FLAG_MSC</i>	MSC status flag
<i>ENET_DMA_INT_FLAG_WUM</i>	WUM status flag
<i>ENET_DMA_INT_FLAG_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 3-285. 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>	-

Input parameter{in}	
<b>int_flag_clear</b>	clear ENET interrupt flag, refer to <a href="#">Table 3-252. Enum enet_int_flag_clear_enum</a> . only one parameter can be selected which is shown as below
ENET_DMA_INT_FLAG_TS_CLR	transmit status flag
ENET_DMA_INT_FLAG_TPS_CLR	transmit process stopped status flag
ENET_DMA_INT_FLAG_TBU_CLR	transmit buffer unavailable status flag
ENET_DMA_INT_FLAG_TJT_CLR	transmit jabber timeout status flag
ENET_DMA_INT_FLAG_RO_CLR	receive overflow status flag
ENET_DMA_INT_FLAG_TU_CLR	transmit underflow status flag
ENET_DMA_INT_FLAG_RS_CLR	receive status flag
ENET_DMA_INT_FLAG_RBU_CLR	receive buffer unavailable status flag
ENET_DMA_INT_FLAG_RPS_CLR	receive process stopped status flag
ENET_DMA_INT_FLAG_RWT_CLR	receive watchdog timeout status flag
ENET_DMA_INT_FLAG_ET_CLR	early transmit status flag
ENET_DMA_INT_FLAG_FBE_CLR	fatal bus error status flag
ENET_DMA_INT_FLAG_ER_CLR	early receive status flag
ENET_DMA_INT_FLAG_AI_CLR	abnormal interrupt summary flag
ENET_DMA_INT_FLAG_NI_CLR	normal interrupt summary flag
Output parameter{out}	
-	-
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 3-286. 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 3-287. 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 3-288. 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 3-289. 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 3-290. 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, refer to <a href="#">Table 3-259. Enum enet_registers_type_enum</a> only one parameter can be selected which is shown as below
ALL_MAC_REG	get the registers within the offset scope range ENET_MAC_CFG to ENET_MAC_FCTH
ALL_MSC_REG	get the registers within the offset scope range ENET_MSC_CTL to ENET_MSC_RGUFCNT
ALL_PTP_REG	get the registers within the offset scope range ENET_PTP_TSCTL to ENET_PTP_PPSCTL
ALL_DMA_REG	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 3-291. 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>	-

The called functions	--
Input parameter{in}	
<b>mac_debug</b>	enet debug status only one parameter can be selected which is shown as below
<i>ENET_MAC_RECEIVE R_NOT_IDLE</i>	MAC receiver is not in idle state
<i>ENET_RX_ASYNC HROUS_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
Output parameter{out}	
-	-
Return value	
<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 3-292. Function enet\_address\_filter\_enable**

Function name	enet_address_filter_enable
Function prototype	void enet_address_filter_enable(enet_macaddress_enum mac_addr);

<b>Function descriptions</b>	enable the MAC address filter
<b>Precondition</b>	-
<b>The called functions</b>	--
<b>Input parameter{in}</b>	
<b>mac_addr</b>	select which MAC address will be enable refer to <a href="#">Table 3-263. Enum enet_macaddress_enum</a> . only one parameter can be selected which is shown as below
ENET_MAC_ADDRES S1	enable MAC address 1 filter
ENET_MAC_ADDRES S2	enable MAC address 2 filter
ENET_MAC_ADDRES S3	enable MAC address 3 filter
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-293. Function enet\_address\_filter\_disable**

<b>Function name</b>	enet_address_filter_disable
<b>Function prototype</b>	void enet_address_filter_disable(enet_macaddress_enum mac_addr);
<b>Function descriptions</b>	disable the MAC address filter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mac_addr</b>	select which MAC address will be enable, refer to <a href="#">Table 3-263. Enum enet_macaddress_enum</a> . only one parameter can be selected which is shown as below
ENET_MAC_ADDRES S1	enable MAC address 1 filter
ENET_MAC_ADDRES S2	enable MAC address 2 filter
ENET_MAC_ADDRES S3	enable MAC address 3 filter
<b>Output parameter{out}</b>	

-	-
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 3-294. 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>	-
<b>Input parameter{in}</b>	
<b>mac_addr</b>	select which MAC address will be enable, refer to <a href="#">Table 3-263. Enum enet_macaddress_enum</a> . only one parameter can be selected which is shown as below
ENET_MAC_ADDRES S1	enable MAC address 1 filter
ENET_MAC_ADDRES S2	enable MAC address 2 filter
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	mask ENET_MAC_ADDR1H [15:8] bits

<i>SK_BYTE5</i>	
<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
<i>ENET_ADDRESS_FILTER_SA</i>	The MAC address is used to compared with the SA field of the received frame
<i>ENET_ADDRESS_FILTER_DA</i>	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 3-295. 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 3-296. 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>	refer to <a href="#">Table 3-261. Enum enet_phydirection_enum</a> , only one parameter can be selected which is shown as below
ENET_PHY_WRITE	write data to phy register
ENET_PHY_READ	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 3-297. 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>	
-	-



Return value	
ErrStatus	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 3-298. Function enet\_phyloopback\_disable**

Function name	enet_phyloopback_disable
Function prototype	ErrStatus enet_phyloopback_disable(void);
Function descriptions	disable the loopback function of PHY chip
Precondition	-
The called functions	enet_phy_write_read
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	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 3-299. Function enet\_forward\_feature\_enable**

Function name	enet_forward_feature_enable
Function prototype	void enet_forward_feature_enable(uint32_t feature);
Function descriptions	enable ENET forward feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	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_ERRFRAMES</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 3-300. 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>	-
<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_ERRFRAMES</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_UNDEERSZ_GOODFRAMES);
```

## enet\_fliter\_feature\_enable

The description of enet\_fliter\_feature\_enable is shown as below:

**Table 3-301. 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>	-
<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
ENET_SRC_FILTER	filter source address function
ENET_SRC_FILTER_INVERSE	inverse source address filtering result function
ENET_DEST_FILTER_INVERSE	inverse DA filtering result function
ENET_MULTICAST_FILTER_PASS	pass all multicast frames function
ENET_MULTICAST_FILTER_HASH_MODE	HASH multicast filter function
ENET_UNICAST_FILTER_HASH_MODE	HASH unicast filter function
ENET_FILTER_MODE_EITHER	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 3-302. Function enet\_fliter\_feature\_disable**

<b>Function name</b>	enet_fliter_feature_disable
<b>Function prototype</b>	void enet_fliter_feature_disable(uint32_t feature);
<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
ENET_SRC_FILTER	filter source address function
ENET_SRC_FILTER_INVERSE	inverse source address filtering result function
ENET_DEST_FILTER_INVERSE	inverse DA filtering result function
ENET_MULTICAST_FILTER_PASS	pass all multicast frames function
ENET_MULTICAST_FILTER_HASH_MODE	HASH multicast filter function
ENET_UNICAST_FILTER_HASH_MODE	HASH unicast filter function
ENET_FILTER_MODE_EITHER	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 3-303. 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>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	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 3-304. 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>	-
<b>Input parameter{in}</b>	
<b>detect</b>	pause frame detect type only one parameter can be selected which is shown as below
<i>ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDDETECT</i>	besides the unique multicast address, MAC can also use the MAC0 address to detecting pause frame
<i>ENET_UNIQUE_PAUSEDDETECT</i>	only the unique multicast address for pause frame which is specified in IEEE802.3 can be detected
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-305. 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 3-306. Function enet\_flowcontrol\_threshold\_config**

<b>Function name</b>	enet_flowcontrol_threshold_config
<b>Function prototype</b>	void enet_flowcontrol_threshold_config(uint32_t deactivate, uint32_t active);
<b>Function descriptions</b>	configure the threshold of the flow control(deactivate and active threshold)
<b>Precondition</b>	-
<b>The called functions</b>	-

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
<i>ENET_DEACTIVE_THRESHOLD_256BYTES</i>	threshold level is 256 bytes
<i>ENET_DEACTIVE_THRESHOLD_512BYTES</i>	threshold level is 512 bytes
<i>ENET_DEACTIVE_THRESHOLD_768BYTES</i>	threshold level is 768 bytes
<i>ENET_DEACTIVE_THRESHOLD_1024BYTES</i>	threshold level is 1024 bytes
<i>ENET_DEACTIVE_THRESHOLD_1280BYTES</i>	threshold level is 1280 bytes
<i>ENET_DEACTIVE_THRESHOLD_1536BYTES</i>	threshold level is 1536 bytes
<i>ENET_DEACTIVE_THRESHOLD_1792BYTES</i>	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_THRESHOLD_256BYTES</i>	threshold level is 256 bytes
<i>ENET_ACTIVE_THRESHOLD_512BYTES</i>	threshold level is 512 bytes
<i>ENET_ACTIVE_THRESHOLD_768BYTES</i>	threshold level is 768 bytes
<i>ENET_ACTIVE_THRESHOLD_1024BYTES</i>	threshold level is 1024 bytes
<i>ENET_ACTIVE_THRESHOLD_1280BYTES</i>	threshold level is 1280 bytes
<i>ENET_ACTIVE_THRESHOLD_1536BYTES</i>	threshold level is 1536 bytes
<i>ENET_ACTIVE_THRESHOLD_1792BYTES</i>	threshold level is 1792 bytes
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the threshold of the flow control */
```

```
enet_flowcontrol_threshold_config(ENET_DEACTIVE_THRESHOLD_256BYTES, ENET_ACTIVE_THRESHOLD_256BYTES);
```

## enet\_flowcontrol\_feature\_enable

The description of enet\_flowcontrol\_feature\_enable is shown as below:

**Table 3-307. 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
ENET_ZERO_QUANTA_PAUSE	the automatic zero-quanta generation function
ENET_TX_FLOWCONTROL	the flow control operation in the MAC
ENET_RX_FLOWCONTROL	decoding function for the received pause frame and process it
ENET_BACK_PRESSURE	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 3-308. 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-quanta 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 3-309. 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>	state of dma process, the value range shows below: 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
--	---

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 3-310. 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>	
-	-
<b>Return value</b>	
-	-

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 3-311. 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 3-312. 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 3-313. Function `enet_current_desc_address_get`**

<b>Function name</b>	<code>enet_current_desc_address_get</code>
<b>Function prototype</b>	<code>uint32_t enet_current_desc_address_get(enet_desc_reg_enum addr_get);</code>
<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, refer to <a href="#">Table 3-253. Enum <code>enet_desc_reg_enum</code></a> only one parameter can be selected which is shown as below
<code>ENET_RX_DESC_TABLE</code>	the start address of the receive descriptor table
<code>ENET_RX_CURRENT_DESC</code>	the start descriptor address of the current receive descriptor read by the RxDMA controller
<code>ENET_RX_CURRENT_BUFFER</code>	the current receive buffer address being read by the RxDMA controller
<code>ENET_TX_DESC_TABLE</code>	the start address of the transmit descriptor table
<code>ENET_TX_CURRENT_DESC</code>	the start descriptor address of the current transmit descriptor read by the TxDMA controller
<code>ENET_TX_CURRENT_BUFFER</code>	the current transmit buffer address being read by the TxDMA controller
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<code>uint32_t</code>	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 3-314. Function `enet_desc_information_get`**

<b>Function name</b>	<code>enet_desc_information_get</code>
<b>Function prototype</b>	<code>uint32_t enet_desc_information_get(enet_descriptors_struct *desc, enet_descstate_enum info_get);</code>
<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="#">Table 3-246. Structure enet_descutors struct</a>
<b>Input parameter{in}</b>	
<b>info_get</b>	the descriptor information type which is selected, refer to <a href="#">Table 3-264. Enum enet_descstate enum</a> only one parameter can be selected which is shown as below
<b>TXDESC_COLLISION_COUNT</b>	the number of collisions occurred before the frame was transmitted
<b>TXDESC_BUFFER_1_ADDR</b>	the buffer1 address of the Tx frame
<b>RXDESC_FRAME_LENGTH</b>	the byte length of the received frame that was transferred to the buffer
<b>RXDESC_BUFFER_1_SIZE</b>	receive buffer 1 size
<b>RXDESC_BUFFER_2_SIZE</b>	receive buffer 2 size
<b>RXDESC_BUFFER_1_ADDR</b>	the buffer1 address of the Rx 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 3-315. 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 3-316. 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="#">Table 3-246. 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
Output parameter{out}	
-	-
Return value	
<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 3-317. 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="#">Table 3-246. 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	
ENET_TDES0_VFRM	VLAN frame
ENET_TDES0_FRMF	frame flushed
ENET_TDES0_TCHM	the second address chained mode
ENET_TDES0_TERM	transmit end of ring mode
ENET_TDES0_TTSN	transmit timestamp function enable
ENET_TDES0_DPAD	disable adding pad
ENET_TDES0_DCRC	disable CRC
ENET_TDES0_FSG	first segment
ENET_TDES0_LSG	last segment
ENET_TDES0_INTC	interrupt on completion
ENET_TDES0_DAV	DAV bit
When type of parameter <b>desc</b> is RX	
ENET_RDES0_DAV	descriptor available
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-318. Function `enet_desc_flag_clear`**

<b>Function name</b>	<code>enet_desc_flag_clear</code>
<b>Function prototype</b>	<code>void enet_desc_flag_clear(enet_descriptors_struct *desc, uint32_t desc_flag);</code>
<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="#">Table 3-246. Structure <code>enet_descriptors_struct</code></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	
<code>ENET_TDES0_VFRM</code>	VLAN frame
<code>ENET_TDES0_FRMF</code>	frame flushed
<code>ENET_TDES0_TCHM</code>	the second address chained mode
<code>ENET_TDES0_TERM</code>	transmit end of ring mode
<code>ENET_TDES0_TTSN</code>	transmit timestamp function enable
<code>ENET_TDES0_DPAD</code>	disable adding pad
<code>ENET_TDES0_DCRC</code>	disable CRC
<code>ENET_TDES0_FSG</code>	first segment
<code>ENET_TDES0_LSG</code>	last segment
<code>ENET_TDES0_INTC</code>	interrupt on completion
<code>ENET_TDES0_DAV</code>	DAV bit
When type of parameter <b>desc</b> is RX	
<code>ENET_RDES0_DAV</code>	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 3-319. Function `enet_rx_desc_immediate_receive_complete_interrupt`**

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

<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="#">Table 3-246. 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 3-320. 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="#">Table 3-246. 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 3-321. 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>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* drop current receive frame */
```

```
enet_rxframe_drop( );
```

## enet\_dma\_feature\_enable

The description of enet\_dma\_feature\_enable is shown as below:

**Table 3-322. Function enet\_dma\_feature\_enable**

<b>Function name</b>	enet_dma_feature_enable
<b>Function prototype</b>	void enet_dma_feature_enable(uint32_t feature);
<b>Function descriptions</b>	enable DMA feature
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>feature</b>	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
<b>Output parameter{out}</b>	
-	-

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 3-323. Function enet\_dma\_feature\_disable**

Function name	enet_dma_feature_disable
Function prototype	void enet_dma_feature_disable(uint32_t feature);
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
ENET_NO_FLUSH_RXFRAME	RxDMA does not flushes frames function
ENET_SECONDFRAME_OPT	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 3-324. Function enet\_rx\_desc\_enhanced\_status\_get**

Function name	enet_rx_desc_enhanced_status_get
Function prototype	uint32_t enet_rx_desc_enhanced_status_get(enet_descriptors_struct *desc, uint32_t desc_status);
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="#">Table 3-246. Structure enet_descriptors_struct</a>
Input parameter{in}	
<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</i> <i>R</i>	IP frame header error
<i>ENET_RDES4_IPPLDE</i> <i>RR</i>	IP frame payload error
<i>ENET_RDES4_IPCKS</i> <i>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</i> <i>F</i>	PTP on ethernet frame
<i>ENET_RDES4_PTPVF</i>	PTP version format
Output parameter{out}	
-	-
Return value	
-	-

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 3-325. 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>	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

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 3-326. 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>	
-	-
<b>Return value</b>	
-	-

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 3-327. 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
<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 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 3-328. 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 3-329. 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 3-330. 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>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-331. Function enet\_ptp\_normal\_descriptors\_chain\_init**

<b>Function name</b>	enet_ptp_normal_descriptors_chain_init
<b>Function prototype</b>	void enet_ptp_normal_descriptors_chain_init(enet_dmadirection_enum direction, enet_descriptors_struct *desc_ptptab);
<b>Function descriptions</b>	initialize the DMA Tx/Rx descriptors's parameters in normal 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>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="#">Table 3-246. Structure enet_descriptors_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-332. 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="#">Table 3-246. 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 3-333. 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 3-334. 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 3-335. 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>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-336. Function enet\_wum\_filter\_config**

<b>Function name</b>	enet_wum_filter_config
<b>Function prototype</b>	void enet_wum_filter_config(uint32_t pdata[]);
<b>Function descriptions</b>	set the remote wakeup frame registers
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pdata</b>	pointer to buffer data which is written to remote wakeup frame registers (8 words total)
<b>Output parameter{out}</b>	

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

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 3-337. 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 3-338. Function `enet_wum_feature_disable`**

<b>Function name</b>	<code>enet_wum_feature_disable</code>
<b>Function prototype</b>	<code>void enet_wum_feature_disable(uint32_t feature)</code>
<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
<code>ENET_WUM_MAGIC_PACKET_FRAME</code>	enable a wakeup event due to magic packet reception
<code>ENET_WUM_WAKEUP_FRAME</code>	enable a wakeup event due to wakeup frame reception
<code>ENET_WUM_GLOBAL_UNICAST</code>	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 3-339. Function `enet_msc_counters_reset`**

<b>Function name</b>	<code>enet_msc_counters_reset</code>
<b>Function prototype</b>	<code>void enet_msc_counters_reset(void);</code>
<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>	
-	-
<b>Return value</b>	
-	-

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 3-340. 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_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 3-341. 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_RESET_ON_READ	reset on read
ENET_MSC_COUNTER_FREEZE	MSC counter freeze

<i>RS_FREEZE</i>	
<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 3-342. 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 <a href="#">Table 3-265. Enum enet_msc_preset_enum</a> 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 3-343. 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, refer to <a href="#">Table 3-254. Enum enet_msc_counter_enum</a> only one parameter can be selected which is shown as below
ENET_MSC_TX_SCCNT	MSC transmitted good frames after a single collision counter
ENET_MSC_TX_MSCCNT	MSC transmitted good frames after more than a single collision counter
ENET_MSC_TX_TGFCNT	MSC transmitted good frames counter
ENET_MSC_RX_RFCECNT	MSC received frames with CRC error counter
ENET_MSC_RX_RFAECNT	MSC received frames with alignment error counter
ENET_MSC_RX_RGUFCNT	MSC received good unicast frames counter
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	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\_subsecond\_2\_nanosecond

The description of enet\_ptp\_subsecond\_2\_nanosecond is shown as below:

**Table 3-344. Function enet\_ptp\_subsecond\_2\_nanosecond**

<b>Function name</b>	enet_ptp_subsecond_2_nanosecond
<b>Function prototype</b>	uint32_t enet_ptp_subsecond_2_nanosecond(uint32_t subsecond);
<b>Function descriptions</b>	change subsecond to nanosecond
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>subsecond</b>	subsecond value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	the nanosecond value

Example:

```
/* change subsecond to nanosecond */
```

```
uint32_t reval;
```

```
reval = enet_ptp_subsecond_2_nanosecond (2);
```

### enet\_ptp\_nanosecond\_2\_subsecond

The description of enet\_ptp\_nanosecond\_2\_subsecond is shown as below:

**Table 3-345. Function enet\_ptp\_nanosecond\_2\_subsecond**

<b>Function name</b>	enet_ptp_nanosecond_2_subsecond
<b>Function prototype</b>	uint32_t enet_ptp_nanosecond_2_subsecond(uint32_t nanosecond);
<b>Function descriptions</b>	change nanosecond to subsecond
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>nanosecond</b>	Nanosecond value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	the subsecond value

Example:

```
/* change nanosecond to subsecond */
```

```
uint32_t reval;
```

```
reval = enet_ptp_nanosecond_2_subsecond (2);
```

### enet\_ptp\_feature\_enable

The description of enet\_ptp\_feature\_enable is shown as below:

**Table 3-346. 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>	-

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

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 3-347. 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>	-
Input parameter{in}	
<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>MP_INT</i>	
<i>ENET_ALL_RX_TIMES TAMP</i>	all received frames are taken snapshot
<i>ENET_NONTYPE_FRA ME_SNAPSHOT</i>	take snapshot when received non type frame
<i>ENET_IPV6_FRAME_S NAPSHOT</i>	take snapshot for IPv6 frame
<i>ENET_IPV4_FRAME_S NAPSHOT</i>	take snapshot for IPv4 frame
<i>ENET_PTP_FRAME_U SE_MACADDRESS_FI LTER</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 3-348. 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_ORDINA RY</i>	type of ordinary clock node type for timestamp
<i>ENET_CKNT_BOUND ARY</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_T O_PEER</i>	type of peer-to-peer transparent clock node type for timestamp
<i>ENET_PTP_ADDEND_</i>	addend register update

<i>UPDATE</i>	
<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>	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 3-349. Function `enet_ptp_subsecond_increment_config`**

<b>Function name</b>	<code>enet_ptp_subsecond_increment_config</code>
<b>Function prototype</b>	<code>void enet_ptp_subsecond_increment_config(uint32_t subsecond);</code>
<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 3-350. Function `enet_ptp_timestamp_addend_config`**

<b>Function name</b>	<code>enet_ptp_timestamp_addend_config</code>
<b>Function prototype</b>	<code>void enet_ptp_timestamp_addend_config(uint32_t add);</code>
<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 3-351. Function `enet_ptp_timestamp_update_config`**

<b>Function name</b>	<code>enet_ptp_timestamp_update_config</code>
<b>Function prototype</b>	<code>void enet_ptp_timestamp_update_config(uint32_t sign, uint32_t second, uint32_t subsecond);</code>
<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
<code>ENET_PTP_ADD_TO_TIME</code>	update value is added to system time
<code>ENET_PTP_SUBTRACT_FROM_TIME</code>	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 3-352. Function `enet_ptp_expected_time_config`**

<b>Function name</b>	<code>enet_ptp_expected_time_config</code>
<b>Function prototype</b>	<code>void enet_ptp_expected_time_config(uint32_t second, uint32_t nanosecond);</code>
<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>	
-	-
<b>Return value</b>	
-	-

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 3-353. 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>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<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="#">Table 3-247. Structure enet_ptp_systime_struct</a>
<b>Return value</b>	
-	-

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 3-354. 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>	-



The called functions	-
Input parameter{in}	
freq	
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_32768HZ	PPS output 32768Hz frequency
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PPS output frequency as 1Hz */
```

```
enet_ptp_pps_output_frequency_config(ENET_PPISOFC_1HZ);
```

## enet\_ptp\_start

The enet\_ptp\_start is shown as below:

**Table 3-355. enet\_ptp\_start**

Function name	enet_ptp_start
---------------	----------------

<b>Function prototype</b>	void enet_ptp_start(int32_t updatemethod, uint32_t init_sec, uint32_t init_subsec, uint32_t carry_cfg, uint32_t accuracy_cfg);
<b>Function descriptions</b>	configure and start PTP timestamp counter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>updatemethod</b>	method for updating
<i>ENET_PTP_FINEMODE</i>	fine correction method
<i>ENET_PTP_COARSEMODE</i>	coarse correction method
<b>Input parameter{in}</b>	
<b>init_sec</b>	second value for initializing system time
<b>Input parameter{in}</b>	
<b>init_subsec</b>	subsecond value for initializing system time
<b>Input parameter{in}</b>	
<b>carry_cfg</b>	the value to be added to the accumulator register (in fine method is used)
<b>Input parameter{in}</b>	
<b>accuracy_cfg</b>	the value to be added to the subsecond value of system time
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* gconfigure and start PTP timestamp counter*/
```

```
enet_ptp_start(ENET_PTP_FINEMODE, 10, 10, 10, 10);
```

## enet\_ptp\_finecorrection\_adjfreq

The enet\_ptp\_finecorrection\_adjfreq is shown as below:

**Table 3-356. enet\_ptp\_finecorrection\_adjfreq**

<b>Function name</b>	enet_ptp_finecorrection_adjfreq
<b>Function prototype</b>	void enet_ptp_finecorrection_adjfreq(int32_t carry_cfg);
<b>Function descriptions</b>	adjust frequency in fine method by configure addend register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>carry_cfg</b>	the value to be added to the accumulator register
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* adjust frequency in fine method by configure addend register */
```

```
enet_ptp_finecorrection_adjfreq(10);
```

### enet\_ptp\_coarsecorrection\_systime\_update

The enet\_ptp\_coarsecorrection\_systime\_update is shown as below:

**Table 3-357. enet\_ptp\_coarsecorrection\_systime\_update**

<b>Function name</b>	enet_ptp_coarsecorrection_systime_update
<b>Function prototype</b>	void enet_ptp_coarsecorrection_systime_update(enet_ptp_systime_struct *systime_struct);
<b>Function descriptions</b>	update system time in coarse method
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>systime_struct</b>	the descriptor pointer which users want to configure, the structure members can refer to <a href="#">Table 3-247. Structure enet_ptp_systime_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* update system time in coarse method */
```

```
enet_ptp_systime_struct systime_struct;
```

```
enet_ptp_coarsecorrection_systime_update (&systime_struct);
```

### enet\_ptp\_finecorrection\_settime

The enet\_ptp\_finecorrection\_settime is shown as below:

**Table 3-358. enet\_ptp\_finecorrection\_settime**

<b>Function name</b>	enet_ptp_finecorrection_settime
<b>Function prototype</b>	void enet_ptp_finecorrection_settime(enet_ptp_systime_struct *systime_struct);
<b>Function descriptions</b>	set system time in fine method
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>systime_struct</b>	the descriptor pointer which users want to configure, the structure members

	can refer to <a href="#">Table 3-247. Structure enet_ptp_systime_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set system time in fine method */
enet_ptp_systime_struct systime_struct;
enet_ptp_finecorrection_settime (&systime_struct);
```

## enet\_ptp\_flag\_get

The enet\_ptp\_flag\_get is shown as below:

**Table 3-359. enet\_ptp\_flag\_get**

<b>Function name</b>	enet_ptp_flag_get
<b>Function prototype</b>	FlagStatus enet_ptp_flag_get(uint32_t flag);
<b>Function descriptions</b>	get the ptp flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	ptp flag status to be checked
ENET_PTP_ADDEND_UPDATE	addend register update
ENET_PTP_SYSTIME_UPDATE	timestamp update
ENET_PTP_SYSTIME_INIT	timestamp initialize
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the ptp flag status */
FlagStatus status = enet_ptp_flag_get(ENET_PTP_ADDEND_UPDATE);
```

## enet\_initpara\_reset

The description of enet\_initpara\_reset is shown as below:

Table 3-360. Function enet\_initpara\_reset

Function name	enet_initpara_reset
Function prototype	void enet_initpara_reset(void);
Function descriptions	reset the ENET initpara struct, call it before using enet_initpara_config()
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the ENET initpara struct */
enet_initpara_reset();
```

## 3.12. EXMC

The external memory controller EXMC, is used as a translator for MCU to access a variety of external memory. The EXMC registers are listed in chapter [3.12.1](#), the EXMC firmware functions are introduced in chapter [3.12.2](#).

### 3.12.1. Descriptions of Peripheral registers

EXMC registers are listed in the table shown as below:

Table 3-361. EXMC Registers

Registers	Descriptions
EXMC_SNCTL	SRAM/NOR Flash control registers
EXMC_SNTCFG	SRAM/NOR Flash timing configuration registers
EXMC_SNWTCFG	SRAM/NOR Flash write timing configuration registers
EXMC_NPCTL	NAND flash/PC card control registers
EXMC_NPINTEN	NAND flash/PC card interrupt enable registers
EXMC_NPCTCFG	NAND flash/PC card common space timing configuration registers
EXMC_NPATCFG	NAND flash/PC card attribute space timing configuration registers
EXMC_PIOTCFG3	PC card I/O space timing configuration register
EXMC_NECC	NAND flash ECC registers

### 3.12.2. Descriptions of Peripheral functions

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

**Table 3-362. EXMC firmware function**

Function name	Function description
exmc_norsram_deinit	deinitialize EXMC NOR/SRAM region x
exmc_norsram_struct_para_init	initialize exmc_norsram_parameter_struct with the default values
exmc_norsram_init	initialize EXMC NOR/SRAM region x
exmc_norsram_enable	enable EXMC NOR/PSRAM region x
exmc_norsram_disable	disable EXMC NOR/PSRAM region x
exmc_norsram_page_size_config	configure CRAM page size
exmc_nand_deinit	deinitialize EXMC NAND bank x
exmc_nand_struct_para_init	initialize exmc_norsram_parameter_struct with the default values
exmc_nand_init	initialize EXMC NAND bank x
exmc_nand_enable	enable EXMC NAND bank x
exmc_nand_disable	disable EXMC NAND bank x
exmc_nand_ecc_config	enable or disable the EXMC NAND ECC function
exmc_ecc_get	get the EXMC ECC value
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_interrupt_enable	enable EXMC interrupt
exmc_interrupt_disable	disable EXMC interrupt
exmc_flag_get	get EXMC flag status
exmc_flag_clear	clear EXMC flag status
exmc_interrupt_flag_get	get EXMC interrupt flag
exmc_interrupt_flag_clear	clear EXMC interrupt flag

## Structure exmc\_norsram\_timing\_parameter\_struct

**Table 3-363. Structure exmc\_norsram\_timing\_parameter\_struct**

Member name	Function description
asyn_access_mode	asynchronous access mode
syn_data_latency	configure the data latency, synchronous access mode valid
syn_clk_division	configure the clock divide ratio, synchronous access mode valid
bus_latency	configure the bus latency
asyn_data_setuptime	configure the data setup time, asynchronous access mode valid
asyn_address_holdtime	configure the address hold time, asynchronous access mode valid
asyn_address_setu	configure the data setup time, asynchronous access mode valid

Member name	Function description
ptime	

## Structure exmc\_norsram\_parameter\_struct

**Table 3-364. 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 bust mode
memory_write	enable or disable the write operation
nwait_config	NWAIT signal configuration, only work in synchronous mode
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, the structure members can refer to <a href="#">Table 3-363. Structure exmc_norsram_timing_parameter_struct</a>
write_timing	timing parameters for write when the extended mode is used, the structure members can refer to <a href="#">Table 3-363. Structure exmc_norsram_timing_parameter_struct</a>

## Structure exmc\_nand\_pccard\_timing\_parameter\_struct

**Table 3-365. Structure exmc\_nand\_pccard\_timing\_parameter\_struct**

Member name	Function description
databus_hiztime	configure the databus HiZ time for write operation
holdtime	configure the address hold time(or the data hold time for write operation)
waittime	configure the minimum wait time
setuptime	configure the address setup time

## Structure exmc\_nand\_parameter\_struct

**Table 3-366. Structure exmc\_nand\_parameter\_struct**

Member name	Function description
nand_bank	select the bank of NAND
ecc_size	the page size for the ECC calculation
atr_latency	configure the latency of ALE low to RB low

Member name	Function description
ctr_latency	configure the latency of CLE low to RB low
ecc_logic	enable or disable the ECC calculation logic
databus_width	the NAND flash databus width
wait_feature	enable or disable the wait feature
common_space_timing	the timing parameters for NAND flash common space, the structure members can refer to <a href="#">Table 3-365. Structure <i>exmc_nand_pccard_timing_parameter_struct</i></a>
attribute_space_timing	the timing parameters for NAND flash attribute space, the structure members can refer to <a href="#">Table 3-365. Structure <i>exmc_nand_pccard_timing_parameter_struct</i></a>

### Structure `exmc_pccard_parameter_struct`

**Table 3-367. Structure `exmc_pccard_parameter_struct`**

Member name	Function description
atr_latency	configure the latency of ALE low to RB low
ctr_latency	configure the latency of CLE low to RB low
wait_feature	enable or disable the wait feature
common_space_timing	the timing parameters for PC card common space, the structure members can refer to <a href="#">Table 3-365. Structure <i>exmc_nand_pccard_timing_parameter_struct</i></a>
attribute_space_timing	the timing parameters for PC card attribute space, the structure members can refer to <a href="#">Table 3-365. Structure <i>exmc_nand_pccard_timing_parameter_struct</i></a>
io_space_timing	the timing parameters for PC card IO space, the structure members can refer to <a href="#">Table 3-365. Structure <i>exmc_nand_pccard_timing_parameter_struct</i></a>

### `exmc_norsram_deinit`

The description of `exmc_norsram_deinit` is shown as below:

**Table 3-368. Function `exmc_norsram_deinit`**

Function name	<code>exmc_norsram_deinit</code>
Function prototype	<code>void exmc_norsram_deinit(uint32_t exmc_norsram_region);</code>
Function descriptions	deinitialize EXMC NOR/SRAM region
Precondition	-
The called functions	-
Input parameter{in}	
<code>exmc_norsram_region</code>	EXMC NOR/SRAM region
<code>EXMC_BANK0_NORSRAM_REGIONx</code>	x=0,1,2,3
Output parameter{out}	



-	-
Return value	
-	-

Example:

```
/* deinitialize the EXMC NOR/SRAM region1 of bank0 */
```

```
exmc_norsram_deinit(EXMC_BANK0_NORSRAM_REGION1);
```

### exmc\_norsram\_struct\_para\_init

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

**Table 3-369. 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 3-364. 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 3-370. 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>	-

Input parameter{in}	
<b>exmc_norsram_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-364</a> . <a href="#">Structure exmc_norsram_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize EXMC NOR/SRAM bank */

exmc_norsram_parameter_struct lcd_init_struct;

exmc_norsram_timing_parameter_struct lcd_timing_init_struct;

/* configure timing parameter */

lcd_timing_init_struct.asyn_access_mode = EXMC_ACCESS_MODE_A;

lcd_timing_init_struct.syn_data_latency = EXMC_DATA_LAT_2_CLK;

lcd_timing_init_struct.syn_clk_division = EXMC_SYN_CLOCK_RATIO_DISABLE;

lcd_timing_init_struct.bus_latency = 1;

lcd_timing_init_struct.asyn_data_setup_time = 5;

lcd_timing_init_struct.asyn_address_hold_time = 2;

lcd_timing_init_struct.asyn_address_setup_time = 2;

/* configure EXMC bus parameters */

lcd_init_struct.norsram_region = EXMC_BANK0_NORSRAM_REGION1;

lcd_init_struct.write_mode = EXMC_ASYNC_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 3-371. 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_NORSRAM_REGIONx</i>	x=0,1,2,3
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the EXMC NOR/SRAM region1 of bank0 */
```

```
exmc_norsram_enable(EXMC_BANK0_NORSRAM_REGION1);
```

### exmc\_norsram\_disable

The description of exmc\_norsram\_disable is shown as below:

**Table 3-372. Function exmc\_norsram\_disable**

<b>Function name</b>	exmc_norsram_disable
<b>Function prototype</b>	void exmc_norsram_disable(uint32_t exmc_norsram_region);
<b>Function descriptions</b>	disable EXMC NOR/PSRAM region
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_norsram_region</b>	EXMC NOR/SRAM region

<b>n</b>	
EXMC_BANK0_NORSRAM_REGIONx	x=0,1,2,3
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the EXMC NOR/SRAM region1 of bank0 */
```

```
exmc_norsram_disable(EXMC_BANK0_NORSRAM_REGION1);
```

### exmc\_norsram\_page\_size\_config

The description of exmc\_norsram\_page\_size\_config is shown as below:

**Table 3-373. 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\_deinit

The description of exmc\_nand\_deinit is shown as below:

**Table 3-374. 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>	-
<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:

```
/* 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 3-375. Function exmc\_nand\_struct\_para\_init**

<b>Function name</b>	exmc_nand_struct_para_init
<b>Function prototype</b>	void exmc_nand_struct_para_init(exmc_nand_parameter_struct* exmc_nand_init_struct);
<b>Function descriptions</b>	initialize the struct exmc_nand_parameter_struct
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_nand_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-366. Structure exmc_nand_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the struct nand_init_struct */
exmc_nand_parameter_struct nand_init_struct;
```

exmc\_nand\_struct\_para\_init (&nand\_init\_struct);

## exmc\_nand\_init

The description of exmc\_nand\_init is shown as below:

**Table 3-376. Function exmc\_nand\_init**

<b>Function name</b>	exmc_nand_init
<b>Function prototype</b>	void exmc_nand_init(exmc_nand_parameter_struct* exmc_nand_init_struct);
<b>Function descriptions</b>	initialize EXMC NAND bank
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_nand_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-366. 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 3-377. 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 3-378. 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\_nand\_ecc\_config

The description of exmc\_nand\_ecc\_config is shown as below:

**Table 3-379. Function exmc\_nand\_ecc\_config**

<b>Function name</b>	exmc_nand_ecc_config
<b>Function prototype</b>	void exmc_nand_ecc_config(uint32_t exmc_nand_bank, ControlStatus newvalue);
<b>Function descriptions</b>	enable or disable the EXMC NAND ECC function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_nand_bank</b>	specifie the NAND bank
<i>EXMC_BANKx_NAND</i>	x=1,2
<b>Input parameter{in}</b>	
<b>newvalue</b>	ENABLE or DISABLE
<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 3-380. 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
<i>EXMC_BANKx_NAND</i>	x=1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	



uint32_t	the error correction code(ECC) value
----------	--------------------------------------

Example:

```
/* get the EXMC ECC value */
```

```
uint32_t ecc_value;
```

```
ecc_value = exmc_ecc_get(EXMC_BANK1_NAND);
```

### exmc\_pccard\_deinit

The description of exmc\_pccard\_deinit is shown as below:

**Table 3-381. Function exmc\_pccard\_deinit**

Function name	exmc_pccard_deinit
Function prototype	void exmc_pccard_deinit(void);
Function descriptions	deinitialize EXMC PC card bank
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

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 3-382. Function exmc\_pccard\_struct\_para\_init**

Function name	exmc_pccard_struct_para_init
Function prototype	void exmc_pccard_struct_para_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);
Function descriptions	initialize the struct exmc_pccard_parameter_struct
Precondition	-
The called functions	-
Input parameter{in}	
exmc_pccard_init_struct	Structure for initialization, the structure members can refer to <a href="#">Table 3-367. Structure exmc_pccard_parameter_struct</a>
Output parameter{out}	

-	-
Return value	
-	-

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 3-383. 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 3-367. 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 3-384. Function exmc\_pccard\_enable**

<b>Function name</b>	exmc_pccard_enable
<b>Function prototype</b>	void exmc_pccard_enable(void);
<b>Function descriptions</b>	enable EXMC PC card bank
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable EXMC PC card bank */

exmc_pccard_enable();
```

## exmc\_pccard\_disable

The description of exmc\_pccard\_disable is shown as below:

**Table 3-385. Function exmc\_pccard\_disable**

<b>Function name</b>	exmc_pccard_disable
<b>Function prototype</b>	void exmc_pccard_disable(void);
<b>Function descriptions</b>	disable EXMC PC card bank
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable EXMC PC card bank */  
  
exmc_pccard_disable();
```

## exmc\_interrupt\_enable

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

**Table 3-386. 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
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCA RD	the PC Card bank
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify get which interrupt flag
EXMC_NAND_PCCAR D_INT_FLAG_RISE	rising edge interrupt and corresponding flag
EXMC_NAND_PCCAR D_INT_FLAG_LEVEL	high-level interrupt and corresponding flag
EXMC_NAND_PCCAR D_INT_FLAG_FALL	falling edge interrupt and corresponding flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable EXMC rising edge interrupt*/  
  
exmc_interrupt_enable(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

## exmc\_interrupt\_disable

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

**Table 3-387. 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
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCA RD	the PC Card bank
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify get which interrupt flag
EXMC_NAND_PCCAR D_INT_FLAG_RISE	rising edge interrupt and corresponding flag
EXMC_NAND_PCCAR D_INT_FLAG_LEVEL	high-level interrupt and corresponding flag
EXMC_NAND_PCCAR D_INT_FLAG_FALL	falling edge interrupt and corresponding flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable EXMC rising edge interrupt */
```

```
exmc_interrupt_disable(EXMC_BANK1_NAND,  
EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

### exmc\_flag\_get

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

**Table 3-388. 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
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCA	the PC Card bank

<i>RD</i>	
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>EXMC_NAND_PCCAR</i> <i>D_FLAG_RISE</i>	interrupt rising edge status
<i>EXMC_NAND_PCCAR</i> <i>D_FLAG_LEVEL</i>	interrupt high-level status
<i>EXMC_NAND_PCCAR</i> <i>D_FLAG_FALL</i>	interrupt falling edge status
<i>EXMC_NAND_PCCAR</i> <i>D_FLAG_FIFOE</i>	FIFO empty status
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* check rising edge status is set or not*/
```

```
if(RESET != exmc_flag_get(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_FLAG_RISE));
```

## exmc\_flag\_clear

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

**Table 3-389. 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
<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
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>EXMC_NAND_PCCAR</i> <i>D_FLAG_RISE</i>	interrupt rising edge status
<i>EXMC_NAND_PCCAR</i> <i>D_FLAG_LEVEL</i>	interrupt high-level status
<i>EXMC_NAND_PCCAR</i>	interrupt falling edge status

<i>D_FLAG_FALL</i>	
<i>EXMC_NAND_PCCAR</i> <i>D_FLAG_FIFOE</i>	FIFO empty status
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear rising edge status */
```

```
exmc_flag_clear(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_FLAG_RISE);
```

### exmc\_interrupt\_flag\_get

The description of `exmc_interrupt_flag_get` is shown as below:

**Table 3-390. Function `exmc_interrupt_flag_get`**

<b>Function name</b>	<code>exmc_interrupt_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus exmc_interrupt_flag_get(uint32_t exmc_bank, uint32_t interrupt);</code>
<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
<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
<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 corresponding flag
<i>EXMC_NAND_PCCAR</i> <i>D_INT_FLAG_LEVEL</i>	high-level interrupt and corresponding flag
<i>EXMC_NAND_PCCAR</i> <i>D_INT_FLAG_FALL</i>	falling edge interrupt and corresponding flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* check rising edge interrupt flag is set or not*/
```

```
if(RESET!=exmc_interrupt_flag_get(EXMC_BANK1_NAND,EXMC_NAND_PCCARD_INT_FLAG_RISE));
```

### exmc\_interrupt\_flag\_clear

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

**Table 3-391. 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
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCA RD	the PC Card bank
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify get which interrupt flag
EXMC_NAND_PCCARD_INT_FLAG_RISE	rising edge interrupt and corresponding flag
EXMC_NAND_PCCARD_INT_FLAG_LEVEL	high-level interrupt and corresponding flag
EXMC_NAND_PCCARD_INT_FLAG_FALL	falling edge interrupt and corresponding flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear rising edge interrupt flag */
```

```
exmc_interrupt_flag_clear(EXMC_BANK1_NAND,EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

## 3.13. EXTI

EXTI is the interrupt / event controller in the MCU. It contains up to 22 independent edge detectors and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter [3.13.1](#), the EXTI firmware functions are introduced in chapter [3.13.2](#).



### 3.13.1. Descriptions of Peripheral registers

EXTI registers are listed in the table shown as below:

**Table 3-392. EXTI Registers**

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

### 3.13.2. Descriptions of Peripheral functions

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

**Table 3-393. EXTI firmware function**

Function name	Function description
exti_deinit	deinitialize the EXTI
exti_init	initialize the EXTI line x
exti_interrupt_enable	enable the interrupts from EXTI line x
exti_interrupt_disable	disable the interrupts from EXTI line x
exti_event_enable	enable the events from EXTI line x
exti_event_disable	disable the events from EXTI line x
exti_software_interrupt_enable	enable the software interrupt event from EXTI line x
exti_software_interrupt_disable	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 3-394. 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

## Enum exti\_mode\_enum

Table 3-395. Enum exti\_mode\_enum

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

## Enum exti\_trig\_type\_enum

Table 3-396. 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 3-397. 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 3-398. Function exti\_init**

Function name	exti_init
Function prototype	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
Function descriptions	initialize the EXTI
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-394. Enum exti_line_enum</a>
Input parameter{in}	
mode	EXTI mode, refer to <a href="#">Table 3-395. Enum exti_mode_enum</a>
Input parameter{in}	
trig_type	trigger type, refer to <a href="#">Table 3-396. 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 3-399. Function exti\_interrupt\_enable**

Function name	enable the interrupts from EXTI line x
Function prototype	void exti_interrupt_enable(exti_line_enum linex);
Function descriptions	enable the interrupts from EXTI line x
Precondition	-

The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-394. Enum exti_line_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

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 3-400. Function exti\_interrupt\_disable**

Function name	exti_interrupt_disable
Function prototype	void exti_interrupt_disable(exti_line_enum linex);
Function descriptions	disable the interrupts from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-394. Enum exti_line_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

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 3-401. Function exti\_event\_enable**

Function name	exti_event_enable
Function prototype	void exti_event_enable(exti_line_enum linex);
Function descriptions	enable the events from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	

<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-394. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the events from EXTI line 0 */
```

```
exti_event_enable(EXTI_0);
```

### exti\_event\_disable

The description of exti\_event\_disable is shown as below:

**Table 3-402. Function exti\_event\_disable**

<b>Function name</b>	exti_event_disable
<b>Function prototype</b>	void exti_event_disable(exti_line_enum linex);
<b>Function descriptions</b>	disable the events from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-394. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the events from EXTI line 0 */
```

```
exti_event_disable(EXTI_0);
```

### exti\_software\_interrupt\_enable

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

**Table 3-403. Function exti\_software\_interrupt\_enable**

<b>Function name</b>	exti_software_interrupt_enable
<b>Function prototype</b>	void exti_software_interrupt_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable the EXTI software interrupt event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-394. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* enable EXTI line 0 software interrupt event */
```

```
exti_software_interrupt_enable(EXTI_0);
```

### exti\_software\_interrupt\_disable

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

**Table 3-404. 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 EXTI software interrupt event
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-394. Enum exti_line_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable EXTI line 0 software interrupt event */
```

```
exti_software_interrupt_disable(EXTI_0);
```

### exti\_flag\_get

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

**Table 3-405. 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 3-394. Enum exti_line_enum</a>
Output parameter{out}	
-	-
Return value	

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

Example:

```
/* get EXTI line 0 flag status */
```

```
FlagStatus state = exti_flag_get(EXTI_0);
```

### exti\_flag\_clear

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

**Table 3-406. Function exti\_flag\_clear**

<b>Function name</b>	exti_flag_clear
<b>Function prototype</b>	void exti_flag_clear(exti_line_enum linex);
<b>Function descriptions</b>	clear EXTI line x interrupt pending flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-394. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear EXTI line 0 flag status */
```

```
exti_flag_clear(EXTI_0);
```

### exti\_interrupt\_flag\_get

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

**Table 3-407. Function exti\_interrupt\_flag\_get**

<b>Function name</b>	exti_interrupt_flag_get
<b>Function prototype</b>	FlagStatus exti_interrupt_flag_get(exti_line_enum linex);
<b>Function descriptions</b>	get EXTI line x interrupt pending flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-394. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get EXTI line 0 interrupt flag status */
```

```
FlagStatus state = exti_interrupt_flag_get(EXTI_0);
```

## exti\_interrupt\_flag\_clear

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

**Table 3-408. 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 3-394. 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);
```

## 3.14. FMC

There is flash controller and option byte for GD32E50x series. The FMC registers are listed in chapter [3.14.1](#) the FMC firmware functions are introduced in chapter [3.14.2](#).

### 3.14.1. Descriptions of Peripheral registers

FMC registers are listed in the table shown as below: :

**Table 3-409. FMC Registers**

Registers	Descriptions
FMC_WS	FMC wait state register
FMC_KEY	FMC unlock key register
FMC_OBKEY	FMC option bytes unlock key register
FMC_STAT	FMC status register
FMC_CTL	FMC control register
FMC_ADDR	FMC address register



Registers	Descriptions
FMC_OBSTAT	FMC option bytes status register
FMC_WP	FMC erase/program protection register
FMC_PID	FMC product ID register

### 3.14.2. Descriptions of Peripheral functions

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

**Table 3-410. FMC firmware function**

Function name	Function description
fmc_unlock	unlock the main FMC operation
fmc_lock	lock the main FMC operation
fmc_wsnt_set	set the FMC wait state
fmc_prefetch_enable	enable pre-fetch
fmc_prefetch_disable	disable pre-fetch
fmc_ibus_enable	enable IBUS cache
fmc_ibus_disable	disable IBUS cache
fmc_ibus_reset	reset IBUS cache
fmc_dbus_enable	enable DBUS cache
fmc_dbus_disable	disable DBUS cache
fmc_dbus_reset	reset DBUS cache
fmc_page_erase	FMC erase page
fmc_mass_erase	FMC erase whole chip
fmc_word_program	FMC program a word at the corresponding address
ob_unlock	unlock the option bytes operation
ob_lock	lock the option bytes operation
ob_erase	erase the option bytes
ob_write_protection_enable	enable write protection
ob_security_protection_config	configure the option bytes security protection
ob_user_write	program option bytes USER
ob_data_program	program option bytes DATA
ob_user_get	get the value of option bytes USER
ob_data_get	get the value of option bytes DATA
ob_write_protection_get	get the value of option bytes write protection
ob_security_protection_flag_get	get option bytes security protection state
fmc_flag_get	get FMC flag status
fmc_flag_clear	clear the FMC flag
fmc_interrupt_enable	enable FMC interrupt
fmc_interrupt_disable	disable FMC interrupt
fmc_interrupt_flag_get	get FMC interrupt flag state
fmc_interrupt_flag_clear	clear FMC interrupt flag

## Enum `fmc_state_enum`

**Table 3-411. `fmc_state_enum`**

enum name	enum description
FMC_READY	the operation has been completed
FMC_BUSY	the operation is in progress
FMC_PGERR	program error
FMC_PGAERR	program alignment error
FMC_WPERR	erase/program protection error
FMC_TOERR	timeout error
FMC_OB_HSPC	high security protection

## `fmc_unlock`

The description of `fmc_unlock` is shown as below:

**Table 3-412. Function `fmc_unlock`**

Function name	<code>fmc_unlock</code>
Function prototype	<code>void fmc_unlock (void);</code>
Function descriptions	unlock the main FMC operation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* unlock the main FMC operation */
```

```
fmc_unlock();
```

## `fmc_lock`

The description of `fmc_lock` is shown as below:

**Table 3-413. Function `fmc_lock`**

Function name	<code>fmc_lock</code>
Function prototype	<code>void fmc_lock(void);</code>
Function descriptions	lock the main FMC operation
Precondition	-
The called functions	-
Input parameter{in}	

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

Example:

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

```
fmc_lock();
```

### fmc\_wscnt\_set

The description of fmc\_wscnt\_set is shown as below:

**Table 3-414. Function fmc\_wscnt\_set**

<b>Function name</b>	fmc_wscnt_set
<b>Function prototype</b>	void fmc_wscnt_set(uint32_t wscnt);
<b>Function descriptions</b>	set the wait state counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>wscnt</b>	wait state counter value
<i>FMC_WAIT_STATE_0</i>	0 wait state added
<i>FMC_WAIT_STATE_1</i>	1 wait state added
<i>FMC_WAIT_STATE_2</i>	2 wait state added
<i>FMC_WAIT_STATE_3</i>	3 wait state added
<i>FMC_WAIT_STATE_4</i>	4 wait state added
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set 1wait state */
```

```
fmc_wscnt_set(WSCNT_1);
```

### fmc\_prefetch\_enable

The description of fmc\_prefetch\_enable is shown as below:

**Table 3-415. Function fmc\_prefetch\_enable**

<b>Function name</b>	fmc_prefetch_enable
<b>Function prototype</b>	void fmc_prefetch_enable(void);
<b>Function descriptions</b>	enable pre-fetch

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

Example:

```
/* enable pre-fetch */
fmc_prefetch_enable();
```

### fmc\_prefetch\_disable

The description of fmc\_prefetch\_disable is shown as below:

**Table 3-416. Function fmc\_prefetch\_disable**

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

Example:

```
/* disable pre-fetch */
fmc_prefetch_disable();
```

### fmc\_ibus\_enable

The description of fmc\_ibus\_enable is shown as below:

**Table 3-417. Function fmc\_ibus\_enable**

<b>Function name</b>	fmc_ibus_enable
<b>Function prototype</b>	void fmc_ibus_enable(void);
<b>Function descriptions</b>	enable IBUS cache
<b>Precondition</b>	-
<b>The called functions</b>	-

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

Example:

```
/* enable IBUS cache */
```

```
fmc_ibus_enable();
```

### fmc\_ibus\_disable

The description of fmc\_ibus\_disable is shown as below:

**Table 3-418. Function fmc\_ibus\_disable**

Function name	fmc_ibus_disable
Function prototype	void fmc_ibus_disable(void);
Function descriptions	disable IBUS cache
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable IBUS cache */
```

```
fmc_ibus_disable();
```

### fmc\_ibus\_reset

The description of fmc\_ibus\_reset is shown as below:

**Table 3-419. Function fmc\_ibus\_reset**

Function name	fmc_ibus_reset
Function prototype	void fmc_ibus_reset(void);
Function descriptions	reset IBUS cache
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset IBUS cache */
```

```
fmc_ibus_reset();
```

## fmc\_dbus\_enable

The description of fmc\_dbus\_enable is shown as below:

**Table 3-420. Function fmc\_dbus\_enable**

Function name	fmc_dbus_enable
Function prototype	void fmc_dbus_enable(void);
Function descriptions	enable DBUS cache
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DBUS cache */
```

```
fmc_dbus_enable();
```

## fmc\_dbus\_disable

The description of fmc\_dbus\_disable is shown as below:

**Table 3-421. Function fmc\_dbus\_disable**

Function name	fmc_dbus_disable
Function prototype	void fmc_dbus_disable(void);
Function descriptions	disable DBUS cache
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* disable DBUS cache */
```

```
fmc_dbus_disable();
```

### fmc\_dbus\_reset

The description of fmc\_dbus\_reset is shown as below:

**Table 3-422. Function fmc\_dbus\_reset**

Function name	fmc_dbus_reset
Function prototype	void fmc_dbus_reset(void);
Function descriptions	reset DBUS cache
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset DBUS cache */
```

```
fmc_dbus_reset();
```

### fmc\_page\_erase

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

**Table 3-423. Function fmc\_page\_erase**

Function name	fmc_page_erase
Function prototype	fmc_state_enum fmc_page_erase(uint32_t page_address);
Function descriptions	FMC erase page
Precondition	fmc_unlock
The called functions	-
Input parameter{in}	
page_address	the page address to be erased
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to <a href="#">Table 3-411. fmc_state_enum</a>

Example:

```
/* erase page */

fmc_unlock();

fmc_state_enum state = fmc_page_erase(0x08004000);
```

### fmc\_mass\_erase

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

**Table 3-424. 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>	FMC erase whole chip
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, refer to <a href="#">Table 3-411. fmc_state_enum</a>

Example:

```
/* erase whole chip */

fmc_unlock();

fmc_state_enum state = fmc_mass_erase();
```

### fmc\_word\_program

The description of fmc\_word\_program is shown as below:

**Table 3-425. Function fmc\_word\_program**

<b>Function name</b>	fmc_word_program
<b>Function prototype</b>	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
<b>Function descriptions</b>	program a word at the corresponding address
<b>Precondition</b>	fmc_unlock, fmc_page_erase / fmc_mass_erase
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
address	the address to program
<b>Input parameter{in}</b>	
data	the data to program
<b>Output parameter{out}</b>	



-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Table 3-411. fmc_state_enum</a>

Example:

```
/* program a word at the corresponding address */

fmc_unlock();

fmc_page_erase(0x08004000);

fmc_state_enum state = fmc_word_program(0x08004000, 0xaabbccdd);
```

## ob\_unlock

The description of ob\_unlock is shown as below:

**Table 3-426. Function ob\_unlock**

<b>Function name</b>	ob_unlock
<b>Function prototype</b>	void ob_unlock(void);
<b>Function descriptions</b>	unlock the option bytes operation
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* unlock the option bytes operation */

fmc_unlock();

ob_unlock();
```

## ob\_lock

The description of ob\_lock is shown as below:

**Table 3-427. Function ob\_lock**

<b>Function name</b>	ob_lock
<b>Function prototype</b>	void ob_lock(void);
<b>Function descriptions</b>	lock the option byts operation
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	-

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

Example:

```
/* lock the option bytes operation */
```

```
fmc_unlock();
```

```
ob_lock();
```

## ob\_erase

The description of ob\_erase is shown as below:

**Table 3-428. Function ob\_erase**

Function name	ob_erase
Function prototype	fmc_state_enum ob_erase(void);
Function descriptions	erase the FMC option bytes
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to <a href="#">Table 3-411. fmc_state_enum</a>

Example:

```
/* erase the FMC option bytes */
```

```
fmc_state_enum state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```
state = ob_erase();
```

## ob\_write\_protection\_enable

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

**Table 3-429. Function ob\_write\_protection\_enable**

Function name	ob_write_protection_enable
---------------	----------------------------

<b>Function prototype</b>	fmc_state_enum 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>	-
<b>Input parameter{in}</b>	
<b>ob_wp</b>	enable write protection
<i>OB_WP_NONE</i>	disable all write protection
<i>OB_WPx</i>	write protect specify sector x
<i>OB_WP_ALL</i>	write protect all sector
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Table 3-411. fmc_state_enum</a>

Example:

```
/* enable write protection */

fmc_state_enum state;

fmc_unlock();

ob_unlock();

state = ob_write_protection_enable(OB_WP7);
```

## ob\_security\_protection\_config

The description of ob\_security\_protection\_config is shown as below:

**Table 3-430. Function ob\_security\_protection\_config**

<b>Function name</b>	ob_security_protection_config
<b>Function prototype</b>	fmc_state_enum ob_security_protection_config (uint8_t ob_spc);
<b>Function descriptions</b>	configure security protection
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ob_spc</b>	specify security protection
<i>FMC_NSPC</i>	no security protection
<i>FMC_LSPC</i>	low security protection
<i>FMC_HSPC</i>	high security protection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Table 3-411. fmc_state_enum</a>

Example:

```
/* enable low security protection */

fmc_state enum state;

ob_unlock();

state = ob_security_protection_config(FMC_LSPC);
```

## ob\_user\_write

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

**Table 3-431. Function ob\_user\_write**

<b>Function name</b>	ob_user_write
<b>Function prototype</b>	fmc_state_enum ob_user_write(uint8_t ob_fwdgt, uint8_t ob_deepsleep, uint8_t ob_stdby, uint8_t ob_boot);
<b>Function descriptions</b>	program option bytes USER
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ob_fwdgt</b>	option bytes watchdog value
OB_FWDGT_SW	software free watchdog
OB_FWDGT_HW	hardware free watchdog
<b>Input parameter{in}</b>	
<b>ob_deepsleep</b>	option bytes deepsleep reset value
OB_DEEPSLEEP_NRS T	no reset when entering deepsleep mode
OB_DEEPSLEEP_RST	generate a reset instead of entering deepsleep mode
<b>Input parameter{in}</b>	
<b>ob_stdby</b>	option bytes standby reset value
OB_STDBY_NRST	no reset when entering standby mode
OB_STDBY_RST	generate a reset instead of entering standby mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Table 3-411. fmc_state_enum</a>

Example:

```
/* configure user option byte */

fmc_state enum state;

fmc_unlock();

ob_unlock();
```

```
state = ob_user_write(OB_FWDGT_HW,OB_DEEPSLEEP_RST,
OB_STDBY_RST);
```

## ob\_data\_program

The description of ob\_data\_program is shown as below:

**Table 3-432. Function ob\_data\_program**

<b>Function name</b>	ob_data_program
<b>Function prototype</b>	fmc_state_enum ob_data_program(uint16_t ob_data);
<b>Function descriptions</b>	program option bytes DATA
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>data</b>	the byte to be programmed
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Table 3-411. fmc_state_enum</a>

Example:

```
/* program option bytes data */
fmc_state_enum state;
fmc_unlock();
ob_unlock();
state = ob_data_program(0x1fff804, 0x56);
```

## ob\_user\_get

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

**Table 3-433. 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 value of option bytes USER
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

<b>uint8_t</b>	the FMC user option bytes values(0x0 – 0xFF)
----------------	--

Example:

```
/* get the FMC user option bytes */
```

```
uint8_t user;
```

```
user = ob_user_get();
```

### ob\_data\_get

The description of ob\_data\_get is shown as below:

**Table 3-434. Function ob\_data\_get**

<b>Function name</b>	ob_data_get
<b>Function prototype</b>	uint16_t ob_data_get(void);
<b>Function descriptions</b>	get the value of option bytes DATA
<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 data option bytes values(0x0 – 0xFFFF)

Example:

```
/* get the FMC data option bytes */
```

```
uint16_t data;
```

```
data = ob_data_get();
```

### ob\_write\_protection\_get

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

**Table 3-435. Function ob\_write\_protection\_get**

<b>Function name</b>	ob_write_protection_get
<b>Function prototype</b>	uint32_t ob_write_protection_get(void);
<b>Function descriptions</b>	get the value of option bytes write protection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

Return value	
<b>uint32_t</b>	the FMC write protection option bytes value(0x0– 0xFFFFFFFF)

Example:

```
/* get the FMC option bytes write protection */
```

```
uint32_t wp;
```

```
wp = ob_write_protection_get();
```

### ob\_security\_protection\_flag\_get

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

**Table 3-436. Function ob\_security\_protection\_flag\_get**

<b>Function name</b>	ob_security_protection_flag_get
<b>Function prototype</b>	FlagStatus ob_security_protection_flag_get(void);
<b>Function descriptions</b>	get the FMC option bytes security protection state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the FMC option bytes security protection */
```

```
FlagStatus flag;
```

```
flag = ob_security_protection_flag_get();
```

### fmc\_flag\_get

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

**Table 3-437. 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>	get FMC flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	FMC flag
<i>FMC_FLAG_BUSY</i>	FMC busy flag

<i>FMC_FLAG_PGERR</i>	FMC program error flag
<i>FMC_FLAG_PGAERR</i>	FMC program alignment error flag
<i>FMC_FLAG_WPERR</i>	FMC erase/program protection error flag
<i>FMC_FLAG_END</i>	FMC end of operation flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get FMC end of operation flag */
```

```
FlagStatus flag;
```

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

## fmc\_flag\_clear

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

**Table 3-438. Function fmc\_flag\_clear**

<b>Function name</b>	fmc_flag_clear
<b>Function prototype</b>	void fmc_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear the FMC flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	FMC flag
<i>FMC_FLAG_PGERR</i>	FMC program error flag
<i>FMC_FLAG_PGAERR</i>	FMC program alignment error flag
<i>FMC_FLAG_WPERR</i>	FMC erase/program protection error flag
<i>FMC_FLAG_END</i>	FMC end of operation flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear FMC program error flag */
```

```
fmc_flag_clear(FMC_FLAG_PGERR);
```

## fmc\_interrupt\_enable

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



Table 3-439. Function `fmc_interrupt_enable`

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

Example:

```
/* enable FMC end of operation interrupt */
```

```
fmc_unlock();
```

```
fmc_interrupt_enable(FMC_INT_END);
```

### `fmc_interrupt_disable`

The description of `fmc_interrupt_disable` is shown as below:

Table 3-440. Function `fmc_interrupt_disable`

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

Example:

```
/* disable FMC end of operation interrupt */
```

```
fmc_unlock();
```

```
fmc_interrupt_disable(FMC_INT_END);
```

## fmc\_interrupt\_flag\_get

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

**Table 3-441. Function fmc\_interrupt\_flag\_get**

<b>Function name</b>	fmc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus fmc_interrupt_flag_get(fmc_interrupt_flag_enum flag);
<b>Function descriptions</b>	get FMC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	FMC interrupt flag
<i>FMC_INT_FLAG_PGE</i> <i>RR</i>	FMC operation error interrupt flag
<i>FMC_INT_FLAG_PGA</i> <i>ERR</i>	FMC program alignment error interrupt flag
<i>FMC_INT_FLAG_WPE</i> <i>RR</i>	FMC erase/program protection error interrupt flag
<i>FMC_INT_FLAG_END</i> <i>K0_PGERR</i>	FMC end of operation interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get FMC operation error interrupt flag bit */
```

```
FlagStatus flag;
```

```
flag = fmc_interrupt_flag_get (FMC_INT_FLAG_PGERR);
```

## fmc\_interrupt\_flag\_clear

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

**Table 3-442. Function fmc\_interrupt\_flag\_clear**

<b>Function name</b>	fmc_interrupt_flag_clear
<b>Function prototype</b>	void fmc_interrupt_flag_clear (fmc_interrupt_flag_enum flag);
<b>Function descriptions</b>	clear FMC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	FMC flag

<i>FMC_INT_FLAG_PGE</i> <i>RR</i>	FMC operation error interrupt flag
<i>FMC_INT_FLAG_PGA</i> <i>ERR</i>	FMC program alignment error interrupt flag
<i>FMC_INT_FLAG_WPE</i> <i>RR</i>	FMC erase/program protection error interrupt flag
<i>FMC_INT_FLAG_END</i> <i>K0_PGERR</i>	FMC end of operation interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear FMC operation error interrupt flag */
fmc_interrupt_flag_clear (FMC_INT_FLAG_PGERR);
```

## 3.15. FWDGT

The free watchdog timer (FWDGT) is a hardware timing circuitry that can be used to detect system failures due to software malfunctions. It's suitable for the situation that requires an independent environment and lower timing accuracy. The FWDGT registers are listed in chapter [3.15.1](#) the FWDGT firmware functions are introduced in chapter [3.15.2](#).

### 3.15.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

**Table 3-443. FWDGT Registers**

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

### 3.15.2. Descriptions of Peripheral functions

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

**Table 3-444. 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

Function name	Function description
fwdgt_enable	start the free watchdog timer counter
fwdgt_prescaler_value_config	configure the FWDGT counter prescaler value
fwdgt_reload_value_config	configure the FWDGT counter reload value
fwdgt_config	configure counter reload value, and prescaler divider value
fwdgt_counter_reload	reload the counter of FWDGT
fwdgt_flag_get	get flag state of FWDGT

### fwdgt\_write\_enable

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

**Table 3-445. 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 3-446. 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 3-447. 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 3-448. Function fwdgt\_prescaler\_value\_config**

<b>Function name</b>	fwdgt_prescaler_value_config
<b>Function prototype</b>	ErrStatus fwdgt_prescaler_value_config(uint16_t prescaler_value);
<b>Function descriptions</b>	configure the FWDGT counter prescaler value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>prescaler_value</b>	specify prescaler value
<i>FWDGT_PSC_DIV4</i>	FWDGT prescaler set to 4
<i>FWDGT_PSC_DIV8</i>	FWDGT prescaler set to 8
<i>FWDGT_PSC_DIV16</i>	FWDGT prescaler set to 16
<i>FWDGT_PSC_DIV32</i>	FWDGT prescaler set to 32
<i>FWDGT_PSC_DIV64</i>	FWDGT prescaler set to 64

<i>FWDGT_PSC_DIV128</i>	FWDGT prescaler set to 128
<i>FWDGT_PSC_DIV256</i>	FWDGT prescaler set to 256
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the FWDGT counter prescaler value */

fwdgt_prescaler_value_config (FWDGT_PSC_DIV8);
```

### fwdgt\_reload\_value\_config

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

**Table 3-449. Function fwdgt\_reload\_value\_config**

<b>Function name</b>	fwdgt_reload_value_config
<b>Function prototype</b>	ErrStatus fwdgt_reload_value_config(uint16_t reload_value);
<b>Function descriptions</b>	configure the FWDGT counter reload value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>reload_value</b>	specify reload value(0x0000 - 0x0FFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the FWDGT counter reload value */

fwdgt_reload_value_config(625);
```

### fwdgt\_config

The description of fwdgt\_config is shown as below:

**Table 3-450. 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)

Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
<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\_counter\_reload

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

**Table 3-451. 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>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reload FWDGT counter */
fwdgt_counter_reload ( );
```

### fwdgt\_flag\_get

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

**Table 3-452. Function fwdgt\_flag\_get**

<b>Function name</b>	fwdgt_flag_get
<b>Function prototype</b>	FlagStatus fwdgt_flag_get(uint16_t flag);
<b>Function descriptions</b>	get flag state of FWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flag to get
<i>FWDGT_FLAG_PUD</i>	a write operation to FWDGT_PSC register is on going
<i>FWDGT_FLAG_RUD</i>	a write operation to FWDGT_RLD register is on going
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```

/* test if a prescaler value update is on going */

FlagStatus status;

status = fwdgt_flag_get (FWDGT_FLAG_PUD);

if(status == RESET)
{
    ...
}
else
{
    ...
}

```

## 3.16. GPIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter [3.16.1](#), the GPIO firmware functions are introduced in chapter [3.16.2](#).

### 3.16.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

**Table 3-453. GPIO Registers**

Registers	Descriptions
GPIOx_CTL0	GPIO port control register 0



Registers	Descriptions
GPIOx_CTL1	GPIO port control register 1
GPIOx_ISTAT	GPIO port input status register
GPIOx_OCTL	GPIO port output control register
GPIOx_BOP	GPIO port bit operate register
GPIOx_BC	GPIO port bit clear register
GPIOx_LOCK	GPIO port configuration lock register
GPIOx_SPD	GPIO port bit speed register
AFIO_EC	AFIO event control register
AFIO_PCF0	AFIO port configuration register 0
AFIO_EXTISS0	AFIO port EXTI sources selection register 0
AFIO_EXTISS1	AFIO port EXTI sources selection register 1
AFIO_EXTISS2	AFIO port EXTI sources selection register 2
AFIO_EXTISS3	AFIO port EXTI sources selection register 3
AFIO_PCF1	AFIO port AFIO port configuration register 1
AFIO_CPSCCTL	IO compensation control register
AFIO_PCFA	AFIO port configuration register A
AFIO_PCFB	AFIO port configuration register B
AFIO_PCFC	AFIO port configuration register C
AFIO_PCFD	AFIO port configuration register D
AFIO_PCFE	AFIO port configuration register E
AFIO_PCFG	AFIO port configuration register G

### 3.16.2. Descriptions of Peripheral functions

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

**Table 3-454. GPIO firmware function**

Function name	Function description
gpio_deinit	reset GPIO port
gpio_afio_deinit	reset alternate function I/O(AFIO)
gpio_init	GPIO parameter initialization
gpio_bit_set	set GPIO pin
gpio_bit_reset	reset GPIO pin
gpio_bit_write	write data to the specified GPIO pin
gpio_port_write	write data to the specified GPIO port
gpio_input_bit_get	get GPIO pin input status
gpio_input_port_get	get GPIO port input status
gpio_output_bit_get	get GPIO pin output status
gpio_output_port_get	get GPIO port output status
gpio_pin_remap_config	configure GPIO pin remap
gpio_afio_port_config	configure AFIO port alternate function
gpio_ethernet_phy_select	select ethernet MII or RMII PHY (for GD32E50X_CL devices)

Function name	Function description
gpio_exti_source_select	select GPIO pin exti sources
gpio_event_output_config	configure GPIO pin event output
gpio_event_output_enable	enable GPIO pin event output
gpio_event_output_disable	disable GPIO pin event output
gpio_pin_lock	lock GPIO pin
gpio_compensation_config	configure the I/O compensation cell
gpio_compensation_flag_get	check the I/O compensation cell is ready or not

## gpio\_deinit

The description of gpio\_deinit is shown as below:

**Table 3-455. 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)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset GPIOA */
```

```
gpio_deinit (GPIOA);
```

## gpio\_afio\_deinit

The description of gpio\_afio\_deinit is shown as below:

**Table 3-456. Function gpio\_afio\_deinit**

Function name	gpio_afio_deinit
Function prototype	void gpio_afio_deinit(void);
Function descriptions	reset alternate function I/O(AFIO)
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* reset alternate function */
```

```
gpio_afio_deinit();
```

## gpio\_init

The description of gpio\_init is shown as below:

**Table 3-457. Function gpio\_init**

<b>Function name</b>	gpio_init
<b>Function prototype</b>	void gpio_init(uint32_t gpio_periph, uint32_t mode, uint32_t speed, uint32_t pin);
<b>Function descriptions</b>	GPIO parameter initialization
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
GPIOx	GPIOx (x=A,B,C,D,E,F,G)
<b>Input parameter{in}</b>	
<b>gpio_mode</b>	gpio pin mode
GPIO_MODE_AIN	analog input mode
GPIO_MODE_IN_FLOATING	floating input mode
GPIO_MODE_IPD	pull-down input mode
GPIO_MODE_IPU	pull-up input mode
GPIO_MODE_OUT_OD	GPIO output with open-drain
GPIO_MODE_OUT_PP	GPIO output with push-pull
GPIO_MODE_AF_OD	AFIO output with open-drain
GPIO_MODE_AF_PP	AFIO output with push-pull
<b>Input parameter{in}</b>	
<b>speed</b>	gpio output max speed value
GPIO_OSPEED_10MHZ	output max speed 10MHz
GPIO_OSPEED_2MHZ	output max speed 2MHz
GPIO_OSPEED_50MHZ	output max speed 50MHz
GPIO_OSPEED_MAX	output max speed more than 50MHz

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

Example:

```
/* config PA0 as analog input mode*/
```

```
gpio_init (GPIOA, GPIO_MODE_AIN, GPIO_OSPEED_50MHZ, GPIO_PIN_0);
```

### gpio\_bit\_set

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

**Table 3-458. Function gpio\_bit\_set**

<b>Function name</b>	gpio_bit_set
<b>Function prototype</b>	void gpio_bit_set(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	set GPIO pin
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx (x=A,B,C,D,E,F,G)
Input parameter{in}	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set PA0*/
```

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

### gpio\_bit\_reset

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

**Table 3-459. Function gpio\_bit\_reset**

<b>Function name</b>	gpio_bit_reset
<b>Function prototype</b>	void gpio_bit_reset(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	reset GPIO pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx (x=A,B,C,D,E,F,G)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x (x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

## gpio\_bit\_write

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

**Table 3-460. Function gpio\_bit\_write**

<b>Function name</b>	gpio_bit_write
<b>Function prototype</b>	void gpio_bit_write(uint32_t gpio_periph,uint32_t pin,bit_status bit_value);
<b>Function descriptions</b>	write data to the specified GPIO pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx (x=A,B,C,D,E,F,G)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Input parameter{in}</b>	
<b>bit_value</b>	SET or RESET
<i>RESET</i>	clear the port pin
<i>SET</i>	set the port pin

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write 1 to PA0 */
```

```
gpio_bit_write (GPIOA, GPIO_PIN_0, SET);
```

## gpio\_port\_write

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

**Table 3-461. Function gpio\_port\_write**

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

Example:

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

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

## gpio\_input\_bit\_get

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

**Table 3-462. Function gpio\_input\_bit\_get**

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

<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx (x=A,B,C,D,E,F,G)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x (x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<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 3-463. Function gpio\_input\_port\_get**

<b>Function name</b>	gpio_input_port_get
<b>Function prototype</b>	uint16_t gpio_input_port_get(uint32_t gpio_periph);
<b>Function descriptions</b>	get GPIO port input status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx (x=A,B,C,D,E,F,G)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	0x0000-0xFFFF

Example:

```
/* get input value of Port A */
```

```
uint16_t port_state;
```

```
port_state = gpio_input_port_get(GPIOA);
```

### gpio\_output\_bit\_get

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

Table 3-464. Function gpio\_output\_bit\_get

Function name	gpio_output_bit_get
Function prototype	FlagStatus gpio_output_bit_get(uint32_t gpio_periph,uint32_t pin);
Function descriptions	get GPIO pin output status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x=A,B,C,D,E,F,G)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get output status of PA0 */
```

```
FlagStatus bit_state;
```

```
bit_state = gpio_output_bit_get (GPIOA, GPIO_PIN_0);
```

### gpio\_output\_port\_get

The description of gpio\_output\_port\_get is shown as below:

Table 3-465. Function gpio\_output\_port\_get

Function name	gpio_output_port_get
Function prototype	uint16_t gpio_output_port_get(uint32_t gpio_periph);
Function descriptions	get GPIO port output status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x=A,B,C,D,E,F,G)
Output parameter{out}	
-	-
Return value	
Uint16_t	0x0000-0xFFFF

Example:

```
/* get output value of Port A */
```



```
uint16_t port_state;
```

```
port_state = gpio_output_port_get (GPIOA);
```

### gpio\_pin\_remap\_config

The description of gpio\_pin\_remap\_config is shown as below:

**Table 3-466. Function gpio\_pin\_remap\_config**

<b>Function name</b>	gpio_pin_remap_config
<b>Function prototype</b>	void gpio_pin_remap_config(uint32_t remap, ControlStatus newvalue);
<b>Function descriptions</b>	configure GPIO pin remap
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_remap</b>	select the pin to remap
GPIO_SPI0_REMAP	SPI0 remapping
GPIO_I2C0_REMAP	I2C0 remapping
GPIO_USART0_REMAP	USART0 remapping
GPIO_USART1_REMAP	USART1 remapping
GPIO_USART2_PARTIAL_REMAP	USART2 partial remapping
GPIO_USART2_FULL_REMAP	USART2 full remapping
GPIO_TIMER0_PARTIAL_REMAP	TIMER0 partial remapping
GPIO_TIMER0_FULL_REMAP	TIMER0 full remapping
GPIO_TIMER1_PARTIAL_REMAP0	TIMER1 partial remapping
GPIO_TIMER1_PARTIAL_REMAP1	TIMER1 partial remapping
GPIO_TIMER1_FULL_REMAP	TIMER1 full remapping
GPIO_TIMER2_PARTIAL_REMAP	TIMER2 partial remapping
GPIO_TIMER2_FULL_REMAP	TIMER2 full remapping
GPIO_TIMER3_REMAP	TIMER3 remapping
GPIO_PD01_REMAP	PD01 remapping
GPIO_TIMER4CH3_IR	TIMER4 channel3 internal remapping

<i>EMAP</i>	
<i>GPIO_ADC0_ETRGR_T_REMAP</i>	ADC0 external trigger routine conversion remapping(only for GD32E50X_HD devices)
<i>GPIO_ADC1_ETRGR_T_REMAP</i>	ADC1 external trigger routine conversion remapping(only for GD32E50X_HD devices)
<i>GPIO_ENET_REMAP</i>	ENET remapping(only for GD32E50X_CL devices)
<i>GPIO_SWJ_NONJTRST_REMAP</i>	full SWJ(JTAG-DP + SW-DP),but without NJTRST
<i>GPIO_SWJ_SWDPENABLE_REMAP</i>	JTAG-DP disabled and SW-DP enabled
<i>GPIO_SWJ_DISABLE_REMAP</i>	JTAG-DP disabled and SW-DP disabled
<i>GPIO_SPI2_REMAP</i>	SPI2 remapping
<i>GPIO_TIMER1ITR0_REMAP</i>	TIMER1 internal trigger 0 remapping(only for GD32E50X_CL devices)
<i>GPIO_PTP_PPS_REMAP</i>	ethernet PTP PPS remapping(only for GD32E50X_CL devices)
<i>GPIO_TIMER8_REMAP</i>	TIMER8 remapping
<i>GPIO_TIMER9_REMAP</i>	TIMER9 remapping
<i>GPIO_TIMER10_REMAP</i>	TIMER10 remapping
<i>GPIO_TIMER12_REMAP</i>	TIMER12 remapping
<i>GPIO_TIMER13_REMAP</i>	TIMER13 remapping
<i>GPIO_EXMC_NADV_REMAP</i>	EXMC_NADV connect/disconnect
<i>GPIO_CTC_REMAP0</i>	CTC remapping(PD15)
<i>GPIO_CTC_REMAP1</i>	CTC remapping(PF0)
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*enable SPI0 remapping*/
```

gpio\_pin\_remap\_config (GPIO\_SPI0\_REMAP, ENABLE);

## gpio\_afio\_port\_config

The description of gpio\_afio\_port\_config is shown as below:

**Table 3-467. Function gpio\_afio\_port\_config**

<b>Function name</b>	gpio_afio_port_config
<b>Function prototype</b>	void gpio_afio_port_config(uint32_t afio_function, ControlStatus newvalue);
<b>Function descriptions</b>	configure AFIO port alternate function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
afio_function	configure the port alternate function(SHRTIMER not support on GD32E50X_EPRT devices)
AFIO_PA2_CMP1_CFG	configure PA2 alternate function to CMP1
AFIO_PA3_USBHS_CFG	configure PA3 alternate function to USBHS
AFIO_PA5_USBHS_CFG	configure PA5 alternate function to USBHS
AFIO_PA8_I2C2_CFG	configure PA8 alternate function to I2C2
AFIO_PA8_SHRTIMER_CFG	configure PA8 alternate function to SHRTIMER
AFIO_PA9_I2C2_CFG	configure PA9 alternate function to I2C2
AFIO_PA9_SHRTIMER_CFG	configure PA9 alternate function to SHRTIMER
AFIO_PA10_CMP5_CFG	configure PA10 alternate function to CMP5
AFIO_PA10_SHRTIMER_CFG	configure PA10 alternate function to SHRTIMER
AFIO_PA11_USART5_CFG	configure PA11 alternate function to USART5
AFIO_PA11_SHRTIMER_CFG	configure PA11 alternate function to SHRTIMER
AFIO_PA12_CMP1_CFG	configure PA12 alternate function to CMP1
AFIO_PA12_USART5_CFG	configure PA12 alternate function to USART5
AFIO_PA12_SHRTIMER_CFG	configure PA12 alternate function to SHRTIMER
AFIO_PA15_SHRTIMER	configure PA15 alternate function to SHRTIMER

MER_CFG	
AFIO_PB0_USBHS_CFG	configure PB0 alternate function to USBHS
AFIO_PB1_CMP3_CFG	configure PB1 alternate function to CMP3
AFIO_PB1_USBHS_CFG	configure PB1 alternate function to USBHS
AFIO_PB1_SHRTIMER_CFG	configure PB1 alternate function to SHRTIMER
AFIO_PB2_USBHS_CFG	configure PB2 alternate function to USBHS
AFIO_PB2_SHRTIMER_CFG	configure PB2 alternate function to SHRTIMER
AFIO_PB3_SHRTIMER_CFG	configure PB3 alternate function to SHRTIMER
AFIO_PB4_I2S2_CFG	configure PB4 alternate function to I2S2
AFIO_PB4_I2C2_CFG	configure PB4 alternate function to I2C2
AFIO_PB4_SHRTIMER_CFG	configure PB4 alternate function to SHRTIMER
AFIO_PB5_I2C2_CFG	configure PB5 alternate function to I2C2
AFIO_PB5_USBHS_CFG	configure PB5 alternate function to USBHS
AFIO_PB5_SHRTIMER_CFG	configure PB5 alternate function to SHRTIMER
AFIO_PB6_SHRTIMER_CFG	configure PB6 alternate function to SHRTIMER
AFIO_PB7_SHRTIMER_CFG	configure PB7 alternate function to SHRTIMER
AFIO_PB8_I2C2_CFG	configure PB8 alternate function to I2C2
AFIO_PB8_SHRTIMER_CFG	configure PB8 alternate function to SHRTIMER
AFIO_PB9_CMP1_CFG	configure PB9 alternate function to CMP1
AFIO_PB9_SHRTIMER_CFG	configure PB9 alternate function to SHRTIMER
AFIO_PB10_USBHS_CFG	configure PB10 alternate function to USBHS
AFIO_PB10_SHRTIMER_CFG	configure PB10 alternate function to SHRTIMER

AFIO_PB11_USBHS_CFG	configure PB11 alternate function to USBHS
AFIO_PB11_SHRTIMER_CFG	configure PB11 alternate function to SHRTIMER
AFIO_PB12_USBHS_CFG	configure PB12 alternate function to USBHS
AFIO_PB12_SHRTIMER_CFG	configure PB12 alternate function to SHRTIMER
AFIO_PB13_USBHS_CFG	configure PB13 alternate function to USBHS
AFIO_PB13_SHRTIMER_CFG	configure PB13 alternate function to SHRTIMER
AFIO_PB14_I2S1_CFG	configure PB14 alternate function to I2S1
AFIO_PB14_SHRTIMER_CFG	configure PB14 alternate function to SHRTIMER
AFIO_PB15_SHRTIMER_CFG	configure PB15 alternate function to SHRTIMER
AFIO_PC0_USBHS_CFG	configure PC0 alternate function to USBHS
AFIO_PC2_I2S1_CFG	configure PC2 alternate function to I2S1
AFIO_PC2_USBHS_CFG	configure PC2 alternate function to USBHS
AFIO_PC3_USBHS_CFG	configure PC3 alternate function to USBHS
AFIO_PC6_CMP5_CFG	configure PC6 alternate function to CMP5
AFIO_PC6_USART5_CFG	configure PC6 alternate function to USART5
AFIO_PC6_SHRTIMER_CFG	configure PC6 alternate function to SHRTIMER
AFIO_PC7_USART5_CFG	configure PC7 alternate function to USART5
AFIO_PC7_SHRTIMER_CFG	configure PC7 alternate function to SHRTIMER
AFIO_PC8_USART5_CFG	configure PC8 alternate function to USART5
AFIO_PC8_SHRTIMER_CFG	configure PC8 alternate function to SHRTIMER
AFIO_PC9_I2C2_CFG	configure PC9 alternate function to I2C2
AFIO_PC9_SHRTIMER	configure PC9 alternate function to SHRTIMER

ER_CFG	
AFIO_PC10_I2C2_CFG	configure PC10 alternate function to I2C2
AFIO_PC11_I2S2_CFG	configure PC11 alternate function to I2S2
AFIO_PC11_SHRTIMER_CFG	configure PC11 alternate function to SHRTIMER
AFIO_PC12_SHRTIMER_CFG	configure PC12 alternate function to SHRTIMER
AFIO_PD4_SHRTIMER_CFG	configure PD4 alternate function to SHRTIMER
AFIO_PD5_SHRTIMER_CFG	configure PD5 alternate function to SHRTIMER
AFIO_PE0_SHRTIMER_CFG	configure PE0 alternate function to SHRTIMER
AFIO_PE1_SHRTIMER_CFG	configure PE1 alternate function to SHRTIMER
AFIO_PE8_CMP1_CFG	configure PE8 alternate function to CMP1
AFIO_PE9_CMP3_CFG	configure PE9 alternate function to CMP3
AFIO_PE10_CMP5_CFG	configure PE10 alternate function to CMP5
AFIO_PE11_CMP5_CFG	configure PE11 alternate function to CMP5
AFIO_PE12_CMP3_CFG	configure PE12 alternate function to CMP3
AFIO_PE13_CMP1_CFG	configure PE13 alternate function to CMP1
AFIO_PG6_SHRTIMER_CFG	configure PG6 alternate function to SHRTIMER_C
AFIO_PG7_USART5_CFG	configure PG7 alternate function to USART5
AFIO_PG7_SHRTIMER_CFG	configure PG7 alternate function to SHRTIMER
AFIO_PG9_USART5_CFG	configure PG9 alternate function to USART5
AFIO_PG10_SHRTIMER_CFG	configure PG10 alternate function to SHRTIMER
AFIO_PG11_SHRTIMER_CFG	configure PG11 alternate function to SHRTIMER
AFIO_PG12_SHRTIMER_CFG	configure PG12 alternate function to SHRTIMER

AFIO_PG13_SHRTIMER_CFG	configure PG13 alternate function to SHRTIMER
AFIO_PG14_USART5_CFG	configure PG14 alternate function to USART5
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure PG14 alternate function to USART5*/
```

```
gpio_afio_port_config (AFIO_PG14_USART5_CFG, ENABLE);
```

## gpio\_ethernet\_phy\_select

The description of gpio\_ethernet\_phy\_select is shown as below:

**Table 3-468. Function gpio\_ethernet\_phy\_select**

<b>Function name</b>	gpio_ethernet_phy_select
<b>Function prototype</b>	void gpio_ethernet_phy_select(uint32_t enet_sel);
<b>Function descriptions</b>	select ethernet MII or RMII PHY (for GD32E50X_CL devices)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>enet_sel</b>	ethernet MII or RMII PHY selection
<i>GPIO_ENET_PHY_MII</i>	configure ethernet MAC for connection with an MII PHY
<i>GPIO_ENET_PHY_RMII</i>	configure ethernet MAC for connection with an RMII PHY
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ethernet MAC for connection with an RMII PHY */
```

```
gpio_ethernet_phy_select (GPIO_ENET_PHY_RMII);
```

## gpio\_exti\_source\_select

The description of gpio\_exti\_source\_select is shown as below:

**Table 3-469. Function gpio\_exti\_source\_select**

<b>Function name</b>	gpio_exti_source_select
<b>Function prototype</b>	void gpio_exti_source_select(uint8_t output_port, uint8_t output_pin);
<b>Function descriptions</b>	select GPIO pin exti sources
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>output_port</b>	gpio event output port
<b>GPIO_PORT_SOURCE_GPIOx</b>	output port source (x= A,B,C,D,E)
<b>Input parameter{in}</b>	
<b>output_pin</b>	gpio event output pin
<b>GPIO_PIN_SOURCE_x</b>	pin number (x=0..15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config PA0 as EXTI source*/
```

```
gpio_exti_source_select (GPIO_PORT_SOURCE_GPIOA, GPIO_PIN_SOURCE_0);
```

## gpio\_event\_output\_config

The description of gpio\_event\_output\_config is shown as below:

**Table 3-470. Function gpio\_event\_output\_config**

<b>Function name</b>	gpio_event_output_config
<b>Function prototype</b>	void gpio_event_output_config(uint8_t output_port, uint8_t output_pin);
<b>Function descriptions</b>	configure GPIO pin event output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>output_port</b>	gpio event output port
<b>GPIO_EVENT_PORT_GPIOx</b>	event output port x (x= A,B,C,D,E)
<b>Input parameter{in}</b>	
<b>output_pin</b>	gpio event output pin
<b>GPIO_EVENT_PIN_x</b>	pin number (x=0..15)
<b>Output parameter{out}</b>	



-	-
Return value	
-	-

Example:

```
/* Config PA0 as the output of event */
```

```
gpio_event_output_config (GPIO_EVENT_PORT_GPIOA, GPIO_EVENT_PIN_0);
```

### gpio\_event\_output\_enable

The description of gpio\_event\_output\_enable is shown as below:

**Table 3-471. Function gpio\_event\_output\_enable**

Function name	gpio_event_output_enable
Function prototype	void gpio_event_output_enable(void);
Function descriptions	enable GPIO pin event output
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable GPIO pin event output */
```

```
gpio_event_output_enable(void);
```

### gpio\_event\_output\_disable

The description of gpio\_event\_output\_disable is shown as below:

**Table 3-472. Function gpio\_event\_output\_disable**

Function name	gpio_event_output_disable
Function prototype	void gpio_event_output_disable(void);
Function descriptions	disable GPIO pin event output
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* disable GPIO pin event output */
```

```
gpio_event_output_disable(void);
```

## gpio\_pin\_lock

The description of gpio\_pin\_lock is shown as below:

**Table 3-473. 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
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx (x=A,B,C,D,E,F,G)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* lock PA0 */
```

```
gpio_pin_lock (GPIOA, GPIO_PIN_0);
```

## gpio\_compensation\_config

The description of gpio\_compensation\_config is shown as below:

**Table 3-474. Function gpio\_compensation\_config**

<b>Function name</b>	gpio_compensation_config
<b>Function prototype</b>	void gpio_compensation_config(uint32_t compensation);
<b>Function descriptions</b>	configure the I/O compensation cell
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>compensation</b>	specifies the I/O compensation cell mode

<code>GPIO_COMPENSATION_ENABLE</code>	I/O compensation cell is enabled
<code>GPIO_COMPENSATION_DISABLE</code>	I/O compensation cell is disabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enabled I/O compensation cell */
gpio_compensation_config (GPIO_COMPENSATION_ENABLE);
```

### gpio\_compensation\_flag\_get

The description of `gpio_compensation_flag_get` is shown as below:

**Table 3-475. Function `gpio_compensation_flag_get`**

<b>Function name</b>	<code>gpio_compensation_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus gpio_compensation_flag_get(void);</code>
<b>Function descriptions</b>	check the I/O compensation cell is ready or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* check the I/O compensation cell state */
FlagStatus cell_state;
cell_state = gpio_compensation_flag_get (void);
```

## 3.17. SHRTIMER

SHRTIMER has a high-precision counting clock and can be used for high-precision timing. It can generate 10 high precision and flexible digital signals to control motor or be used for power management applications. The 10 digital signals can be output independently or coupled into 5 pairs of complementary signals. The SHRTIMER registers are listed in chapter [3.17.1](#), the SHRTIMER firmware functions are introduced in chapter [3.17.2](#). The GD32EPRT

series does not have SHRTIMER module.

### 3.17.1. Descriptions of Peripheral registers

SHRTIMER registers are listed in the table shown as below:

**Table 3-476. SHRTIMER Register**

Registers	Descriptions
<b>Master Timer Registers</b>	
SHRTIMER_MTCTL0	Master_TIMER control register 0
SHRTIMER_MTINTF	Master_TIMER interrupt flag register
SHRTIMER_MTINTC	Master_TIMER interrupt flag clear register
SHRTIMER_MTDMAINTEN	Master_TIMER DMA and interrupt enable register
SHRTIMER_MTCNT	Master_TIMER counter register
SHRTIMER_MTCAR	Master_TIMER counter auto reload register
SHRTIMER_MTCREP	Master_TIMER counter repetition register
SHRTIMER_MTCMP0V	Master_TIMER compare 0 value register
SHRTIMER_MTCMP1V	Master_TIMER compare 1 value register
SHRTIMER_MTCMP2V	Master_TIMER compare 2 value register
SHRTIMER_MTCMP3V	Master_TIMER compare 3 value register
SHRTIMER_MTACTL	Master_TIMER additional control register
<b>Slave Timer Registers</b>	
SHRTIMER_STXCTL0	Slave_TIMERx control register 0
SHRTIMER_STXINTF	Slave_TIMERx interrupt flag register
SHRTIMER_STXINTC	Slave_TIMERx interrupt flag clear register
SHRTIMER_STXDMAINTEN	Slave_TIMERx DMA and interrupt enable register
SHRTIMER_STXCNT	Slave_TIMERx counter register
SHRTIMER_STXCAR	Slave_TIMERx counter auto reload register
SHRTIMER_STXCREP	Slave_TIMERx counter repetition register
SHRTIMER_STXCMP0V	Slave_TIMERx compare 0 value register
SHRTIMER_STXCMP0CP	Slave_TIMERx compare 0 composite register
SHRTIMER_STXCMP1V	Slave_TIMERx compare 1 value register
SHRTIMER_STXCMP2V	Slave_TIMERx compare 2 value register
SHRTIMER_STXCMP3V	Slave_TIMERx compare 3 value register
SHRTIMER_STXCAP0V	Slave_TIMERx capture 0 value register
SHRTIMER_STXCAP1V	Slave_TIMERx capture 1 value register
SHRTIMER_STXDTCTL	Slave_TIMERx dead-time control register
SHRTIMER_STXCH0SET	Slave_TIMERx channel 0 set request register
SHRTIMER_STXCH0RST	Slave_TIMERx channel 0 reset request register
SHRTIMER_STXCH1SET	Slave_TIMERx channel 1 set request register
SHRTIMER_STXCH1RST	Slave_TIMERx channel 1 reset request register
SHRTIMER_STXEXEVFCFG 0	Slave_TIMERx external event filter configuration register 0

Registers	Descriptions
SHRTIMER_STXEXEVFCFG 1	Slave_TIMERx external event filter configuration register 1
SHRTIMER_STXCNTRST	Slave_TIMERx counter reset register
SHRTIMER_STXCSCCTL	Slave_TIMERx carrier-signal control register
SHRTIMER_STXCAP0TRG	Slave_TIMERx capture 0 trigger register
SHRTIMER_STXCAP1TRG	Slave_TIMERx capture 1 trigger register
SHRTIMER_STXCHOCTL	Slave_TIMERx channel output control register
SHRTIMER_STXFLTCTL	Slave_TIMERx fault control register
SHRTIMER_STXACTL	Slave_TIMERx additional control register
<b>Common Registers</b>	
SHRTIMER_CTL0	SHRTIMER control register 0
SHRTIMER_CTL1	SHRTIMER control register 1
SHRTIMER_INTF	SHRTIMER interrupt flag register
SHRTIMER_INTC	SHRTIMER interrupt flag clear register
SHRTIMER_INTEN	SHRTIMER interrupt enable register
SHRTIMER_CHOUTEN	SHRTIMER channel output enable register
SHRTIMER_CHOUTDIS	SHRTIMER channel output disable register
SHRTIMER_CHOUTDISF	SHRTIMER channel output disable flag register
SHRTIMER_BMCTL	SHRTIMER bunch mode control register
SHRTIMER_BMSTRG	SHRTIMER bunch mode start trigger register
SHRTIMER_BMCMPV	SHRTIMER bunch mode compare value register
SHRTIMER_BMCAR	SHRTIMER bunch mode counter auto reload register
SHRTIMER_EXEVCFG0	SHRTIMER external event configuration register 0
SHRTIMER_EXEVCFG1	SHRTIMER external event configuration register 1
SHRTIMER_EXEVDCTL	SHRTIMER external event digital filter control register
SHRTIMER_ADCTRIGS0	SHRTIMER trigger source 0 to ADC register
SHRTIMER_ADCTRIGS1	SHRTIMER trigger source 1 to ADC register
SHRTIMER_ADCTRIGS2	SHRTIMER trigger source 2 to ADC register
SHRTIMER_ADCTRIGS3	SHRTIMER trigger source 3 to ADC register
SHRTIMER_DLLCCTL	SHRTIMER DLL calibration control register
SHRTIMER_FLTINCFG0	SHRTIMER fault input configuration register 0
SHRTIMER_FLTINCFG1	SHRTIMER fault input configuration register 1
SHRTIMER_DMAUPMTR	SHRTIMER DMA update Master_TIMER register
SHRTIMER_DMAUPST0R	SHRTIMER DMA update Slave_TIMER0 register
SHRTIMER_DMAUPST1R	SHRTIMER DMA update Slave_TIMER1 register
SHRTIMER_DMAUPST2R	SHRTIMER DMA update Slave_TIMER2 register
SHRTIMER_DMAUPST3R	SHRTIMER DMA update Slave_TIMER3 register
SHRTIMER_DMAUPST4R	SHRTIMER DMA update Slave_TIMER4 register
SHRTIMER_DMATB	SHRTIMER DMA transfer buffer register

### 3.17.2. Descriptions of Peripheral functions

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

**Table 3-477. SHRTIMER firmware function**

Function name	Function description
shrtimer_deinit	deinit a SHRTIMER
shrtimer_dll_calibration_start	configure and start DLL calibration
shrtimer_baseinit_struct_para_init	initialize SHRTIMER time base parameters struct with a default value
shrtimer_timers_base_init	initialize Master_TIMER and Slave_TIMER timerbase
shrtimer_timers_counter_enable	enable a counter
shrtimer_timers_counter_disable	disable a counter
shrtimer_timers_update_event_enable	enable the Master_TIMER or Slave_TIMER update event
shrtimer_timers_update_event_disable	disable the Master_TIMER or Slave_TIMER update event
shrtimer_software_update	trigger the Master_TIMER and Slave_TIMER registers update by software
shrtimer_software_counter_reset	the Master_TIMER and Slave_TIMER counter reset by software
shrtimer_timerinit_struct_para_init	initialize waveform mode initialization parameters struct with a default value
shrtimer_timers_waveform_init	initialize a timer to work in waveform mode
shrtimer_timercfg_struct_para_init	initialize Slave_TIMER general behavior configuration struct with a default value
shrtimer_slavetimer_waveform_config	configure the general behavior of a Slave_TIMER which work in waveform mode
shrtimer_comparecfg_struct_para_init	initialize compare unit configuration struct with a default value
shrtimer_slavetimer_waveform_compare_config	configure the compare unit of a Slave_TIMER which work in waveform mode
shrtimer_channel_outputcfg_struct_para_init	initialize channel output configuration struct with a default value
shrtimer_slavetimer_waveform_channel_config	configure the channel output of a Slave_TIMER work in waveform mode
shrtimer_slavetimer_waveform_channel_software_request	software generates channel "set request" or "reset request"
shrtimer_slavetimer_waveform_channel_output_level_get	get Slave_TIMER channel output level
shrtimer_slavetimer_waveform_channel_state_get	get Slave_TIMER channel run state
shrtimer_deadtimecfg_struct_para_init	initialize dead time configuration struct with a default value

Function name	Function description
shrtimer_slavetimer_deadtime_config	configure the dead time for Slave_TIMER
shrtimer_carriersignalcfg_struct_para_init	initialize carrier signal configuration struct with a default value
shrtimer_slavetimer_carriersignal_config	configure the carrier signal mode for Slave_TIMER
shrtimer_output_channel_enable	enable a output channel
shrtimer_output_channel_disable	disable a output channel
shrtimer_mastertimer_compare_value_config	configure the compare value in Master_TIMER
shrtimer_mastertimer_compare_value_get	get the compare value in Master_TIMER
shrtimer_slavetimer_compare_value_config	configure the compare value in Slave_TIMER
shrtimer_slavetimer_compare_value_get	get the compare value in Slave_TIMER
shrtimer_timers_counter_value_config	configure the counter value in Master_TIMER and Slave_TIMER
shrtimer_timers_counter_value_get	get the counter value in Master_TIMER and Slave_TIMER
shrtimer_timers_autoreload_value_config	configure the counter auto reload value in Master_TIMER and Slave_TIMER
shrtimer_timers_autoreload_value_get	get the counter auto reload value in Master_TIMER and Slave_TIMER
shrtimer_timers_repetition_value_config	configure the counter repetition value in Master_TIMER and Slave_TIMER
shrtimer_timers_repetition_value_get	get the counter repetition value in Master_TIMER and Slave_TIMER
shrtimer_exefilter_struct_para_init	initialize external event filtering for Slave_TIMER configuration struct with a default value
shrtimer_slavetimer_exeevent_filtering_config	configure the external event filtering for Slave_TIMER (blanking, windowing)
shrtimer_exeeventcfg_struct_para_init	initialize external event configuration struct with a default value
shrtimer_exeevent_config	configure the an external event
shrtimer_exeevent_prescaler	configure external event digital filter clock division
shrtimer_synccfg_struct_para_init	initialize synchronization configuration struct with a default value
shrtimer_synchronization_config	configure the synchronization input/output of the SHRTIMER
shrtimer_faultcfg_struct_para_init	configure the synchronization input/output of the SHRTIMER
shrtimer_fault_config	configure the fault input
shrtimer_fault_prescaler_config	configure the fault input digital filter clock division
shrtimer_fault_input_enable	fault input enable

Function name	Function description
shrtimer_fault_input_disable	fault input disable
shrtimer_timers_dma_enable	enable the Master_TIMER and Slave_TIMER DMA request
shrtimer_timers_dma_disable	disable the Master_TIMER and Slave_TIMER DMA request
shrtimer_dmamode_config	configure the DMA mode for Master_TIMER or Slave_TIMER
shrtimer_bunchmode_struct_para_init	initialize bunch mode configuration struct with a default value
shrtimer_bunchmode_config	configure bunch mode for the SHRTIMER
shrtimer_bunchmode_enable	enable the bunch mode
shrtimer_bunchmode_disable	disable the bunch mode
shrtimer_bunchmode_flag_get	get bunch mode operating flag
shrtimer_bunchmode_software_start	bunch mode started by software
shrtimer_slavetimer_capture_config	configure the capture source in Slave_TIMER
shrtimer_slavetimer_capture_software	capture triggered by software in Slave_TIMER
shrtimer_slavetimer_capture_value_read	read the capture value
shrtimer_adctrigcfg_struct_para_init	initialize ADC trigger configuration struct with a default value
shrtimer_adc_trigger_config	configure the trigger source to ADC and the update source
shrtimer_timers_flag_get	get the Master_TIMER and Slave_TIMER flag
shrtimer_timers_flag_clear	clear the Master_TIMER and Slave_TIMER flag
shrtimer_common_flag_get	get the common flag
shrtimer_common_flag_clear	clear the common flag
shrtimer_timers_interrupt_enable	enable the Master_TIMER and Slave_TIMER interrupt
shrtimer_timers_interrupt_disable	disable the Master_TIMER and Slave_TIMER interrupt
shrtimer_timers_interrupt_flag_get	clear the Master_TIMER and Slave_TIMER interrupt flag
shrtimer_timers_interrupt_flag_clear	clear the Master_TIMER and Slave_TIMER interrupt flag
shrtimer_common_interrupt_enable	enable the common interrupt
shrtimer_common_interrupt_disable	disable common interrupt
shrtimer_common_interrupt_flag_get	clear the common interrupt flag
shrtimer_common_interrupt_flag_clear	clear the common interrupt flag

### Structure shrtimer\_baseinit\_parameter\_struct

**Table 3-478. Structure shrtimer\_baseinit\_parameter\_struct**

member name	Function description
period	period value, min value: 3 tSHRTIMER_CK clock, max value: 0xFFFF - (1 tSHRTIMER_CK)
repetitioncounter	the counter repetition value, 0x00~0xFF
prescaler	prescaler value, refer to: counter clock division
counter_mode	counter operating mode, refer to: counter operating mode



## Structure shrtimer\_timerinit\_parameter\_struct

**Table 3-479. Structure shrtimer\_timerinit\_parameter\_struct**

member name	Function description
half_mode	specifies whether or not half mode is enabled, refer to: half mode enabling status
start_sync	specifies whether or not timer is started by a rising edge on the synchronization input, refer to: synchronous input start timer
reset_sync	specifies whether or not timer is reset by a rising edge on the synchronization input, refer to: synchronous input reset timer
dac_trigger	indicates whether or not the a DAC synchronization event is generated, refer to: trigger source to DAC
shadow	Indicates whether or not the a DAC synchronization event is generated, refer to: trigger source to DAC
update_selection	the update occurs with respect to DMA mode or STxUPINy (Slave_TIMERx only), refer to: update event selection
cnt_bunch	the timer behaves during a bunch mode operation, refer to: timer behaves during a bunch mode operation
repetition_update	the timer behaves during a bunch mode operation, refer to: timer behaves during a bunch mode operation

## Structure shrtimer\_timercfg\_parameter\_struct

**Table 3-480. Structure shrtimer\_timercfg\_parameter\_struct**

member name	Function description
balanced_mode	specifies whether or not the balanced mode is enabled, refer to: set balanced mode
fault_enable	specifies whether or not the fault channels are enabled for the Slave_TIMER, refer to: fault channel enabled for a Slave_TIMER
fault_protect	specifies whether the write protection function is enable or not, refer to: protect fault enable
deadtime_enable	specifies whether or not dead time insertion is enabled for the timer, refer to: dead time enable
delayed_idle	the delayed IDLE mode, refer to: set delayed IDLE state mode
update_source	the source triggering the Slave_TIMER registers update, refer to: update is done synchronously with any other Slave_TIMER or Master_TIMER update
cnt_reset	the source triggering the Slave_TIMER counter reset, refer to: Slave_TIMER counter reset
reset_update	specifies whether or not registers update is triggered when the timer counter is reset, refer to: update event generated by reset event

### Structure shrtimer\_comparecfg\_parameter\_struct

**Table 3-481. Structure shrtimer\_comparecfg\_parameter\_struct**

member name	Function description
compare_value	compare value, min value: 3 tSHRTIMER_CK clock, max value: 0xFFFF - (1 tSHRTIMER_CK)
delayed_mode	defining whether the compare register is behaving in regular mode or in delayed mode, refer to: compare 3 or 1 delayed mode
timeout_value	compare value for compare 0 or 2 when compare 3 or 1 is delayed mode with time out is selected , timeout_value + compare_value must be less than 0xFFFF

### Structure shrtimer\_exefilter\_parameter\_struct

**Table 3-482. Structure shrtimer\_exefilter\_parameter\_struct**

member name	Function description
filter_mode	the external event filter mode for Slave_TIMER, refer to: external event filter mode
memorized	specifies whether or not the signal is memorized, refer to: external event memorized enable

### Structure shrtimer\_deadtimecfg\_parameter\_struct

**Table 3-483. Structure shrtimer\_deadtimecfg\_parameter\_struct**

member name	Function description
prescaler	dead time generator clock division, refer to: dead time prescaler
rising_value	rising edge dead-time value, 0x0000~0xFFFF
rising_sign	the sign of rising edge dead-time value, refer to: dead time rising sign
rising_protect	dead time rising edge protection for value and sign, refer to: dead time rising edge protection for value and sign
risingsign_protect	dead time rising edge protection for sign, refer to: dead time rising edge protection only for sign
falling_value	falling edge dead-time value, 0x0000~0xFFFF
falling_sign	the sign of falling edge dead-time value, refer to: dead time falling sign
falling_protect	dead time falling edge protection for value and sign, refer to: dead time falling edge protection for value and sign
fallingsign_protect	dead time falling edge protection for sign, refer to: dead time falling edge protection only for sign

### Structure shrtimer\_carriersignalcfg\_parameter\_struct

**Table 3-484. Structure shrtimer\_carriersignalcfg\_parameter\_struct**

member name	Function description
period	carrier signal period: tCSPRD, 0x0~0xF. tCSPRD = (period + 1) * 16 *

	tSHRTIMER_CK
duty_cycle	carrier signal duty cycle, 0x0~0x7, duty cycle = duty_cycle/8
first_pulse	first carrier-signal pulse width: tCSFSTPW, 0x0~0xF. tCSFSTPW = (first_pulse+1) * 16 * tSHRTIMER_CK

## Structure shrtimer\_synccfg\_parameter\_struct

**Table 3-485. Structure shrtimer\_synccfg\_parameter\_struct**

member name	Function description
input_source	the external synchronization input source, refer to: the synchronization input source
output_source	the source and event to be sent on the external synchronization outputs, refer to: the synchronization output source
output_polarity	the polarity and length of the pulse to be sent on the external synchronization outputs, refer to: the pulse on the synchronization output pad SHRTIMER_SCOUT

## Structure shrtimer\_bunchmode\_parameter\_struct

**Table 3-486. Structure shrtimer\_bunchmode\_parameter\_struct**

member name	Function description
mode	the bunch mode operating mode, refer to: continuous mode in bunch mode
clock_source	specifies the burst mode clock source, refer to: bunch mode clock source
prescaler	the bunch mode prescaler, refer to: bunch mode clock division
shadow	specifies whether or not preload is enabled for SHRTIMER_BMCMPV and SHRTIMER_BMCAR registers, refer to: bunch mode shadow enable
trigger	the event triggering the bunch operation, refer to: the event triggers bunch mode operation
idle_duration	the duration of the IDLE, 0x0000~0xFFFF
period	the bunch mode period which is the sum of the IDLE and RUN duration, 0x0001~0xFFFF

## Structure shrtimer\_exeventcfg\_parameter\_struct

**Table 3-487. Structure shrtimer\_exeventcfg\_parameter\_struct**

member name	Function description
source	the source of the external event, refer to: external event source
polarity	the active level of external event 0 when EXEVyEG[1:0] = 2'b00, refer to: external event polarity
edge	the sensitivity of the external event, external event edge sensitivity
digital_filter	external event filter control, 0x0~0xF

## Structure `shrtimer_faultcfg_parameter_struct`

**Table 3-488. Structure `shrtimer_faultcfg_parameter_struct`**

member name	Function description
source	the source of the fault input, refer to: fault input source
polarity	the polarity of the fault input, refer to: fault input polarity
filter	fault input filter control, 0x0~0xF
control	fault input enable or disable, refer to: enable or disable fault
protect	protect fault input configuration, refer to: protect fault input configuration

## Structure `shrtimer_adctrigcfg_parameter_struct`

**Table 3-489. Structure `shrtimer_adctrigcfg_parameter_struct`**

member name	Function description
update_source	the source triggering the update of the SHRTIMER_ADCTRIGSy register, refer to: SHRTIMER_ADCTRIG update source
trigger	the event triggering the ADC conversion, refer to: ADC trigger event

## Structure `shrtimer_channel_outputcfg_parameter_struct`

**Table 3-490. Structure `shrtimer_channel_outputcfg_parameter_struct`**

member name	Function description
polarity	configure channel output polarity, refer to: channel output polarity
set_request	configure the event generates channel 'set request', refer to channel set request
reset_request	configure the event generates channel 'reset request', refer to: channel reset request
idle_bunch	specifies whether channel output can be IDLE state in bunch mode, refer to: channel IDLE state enable in bunch mode
idle_state	specifies channel output idle state, refer to channel output idle state
fault_state	specifies the output level when in FAULT state, refer to: channel output in fault state
carrier_mode	specifies whether or not the carrier-signal mode is enabled, refer to: channel carrier-signal mode enable
deadtime_bunch	specifies whether or not deadtime is inserted before output entering the IDLE state in bunch mode, refer to: channel dead-time insert in bunch mode

## `shrtimer_deinit`

The description of `shrtimer_deinit` is shown as below:

**Table 3-491. Function `shrtimer_deinit`**

Function name	<code>shrtimer_deinit</code>
Function prototype	<code>void shrtimer_deinit(uint32_t shrtimer_periph);</code>

<b>Function descriptions</b>	deinit SHRTIMER
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<b>SHRTIMERx(x=0)</b>	SHRTIMER selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset SHRTIMER */

shrtimer_deinit(SHRTIMER0);
```

### shrtimer\_dll\_calibration\_start

The description of shrtimer\_dll\_calibration\_start is shown as below:

**Table 3-492. Function shrtimer\_dll\_calibration\_start**

<b>Function name</b>	shrtimer_dll_calibration_start
<b>Function prototype</b>	void shrtimer_dll_calibration_start(uint32_t shrtimer_periph);
<b>Function descriptions</b>	configure and start DLL calibration
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<b>SHRTIMERx(x=0)</b>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>calform</b>	specify the calibration form
<b>SHRTIMER_CALIBRATION_ONCE</b>	DLL calibration start once
<b>SHRTIMER_CALIBRATION_1048576_PERIOD</b>	DLL periodic calibration, the length of the DLL calibration cycle is 1048576 * tSHRTIMER_CK
<b>SHRTIMER_CALIBRATION_131072_PERIOD</b>	DLL periodic calibration, the length of the DLL calibration cycle is 131072 * tSHRTIMER_CK
<b>SHRTIMER_CALIBRATION_16384_PERIOD</b>	DLL periodic calibration, the length of the DLL calibration cycle is 16384 * tSHRTIMER_CK
<b>SHRTIMER_CALIBRATION_2048_PERIOD</b>	DLL periodic calibration, the length of the DLL calibration cycle is 2048 * tSHRTIMER_CK
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* configure and start DLL calibration */

shrtimer_dll_calibration_start(SHRTIMER0, SHRTIMER_CALIBRATION_ONCE);
```

### shrtimer\_baseinit\_struct\_para\_init

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

**Table 3-493. Function shrtimer\_baseinit\_struct\_para\_init**

Function name	shrtimer_baseinit_struct_para_init
Function prototype	void shrtimer_baseinit_struct_para_init(shrtimer_baseinit_parameter_struct* baseinit);
Function descriptions	initialize SHRTIMER time base parameters struct with the default value
Precondition	-
The called functions	-
Input parameter{in}	
baseinit	shrtimer_baseinit_parameter_struct, the structure members can refer to <a href="#">Table 3-478. Structure shrtimer_baseinit_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize SHRTIMER time base parameters struct with the default value */

shrtimer_baseinit_parameter_struct baseinit;

shrtimer_baseinit_struct_para_init(&baseinit);
```

### shrtimer\_timers\_base\_init

The description of shrtimer\_timers\_base\_init is shown as below:

**Table 3-494. Function shrtimer\_timers\_base\_init**

Function name	shrtimer_timers_base_init
Function prototype	void shrtimer_timers_base_init(uint32_t shrtimer_periph);
Function descriptions	initialize Master_TIMER and Slave_TIMER timerbase
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection

Input parameter{in}	
<b>timer_id</b>	master timer and slave timer index
<i>SHRTIMER_MASTER_TIMER</i>	the master timer
<i>SHRTIMER_SLAVE_TIMERx</i>	the slave timer selection(x=0..4)
Input parameter{in}	
<b>baseinit</b>	shrtimer_baseinit_parameter_struct, the structure members can refer to <a href="#">Table 3-478. Structure shrtimer_baseinit_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize Master_TIMER and Slave_TIMER timerbase */

shrtimer_baseinit_parameter_struct baseinit_para;

shrtimer_baseinit_struct_para_init(&baseinit_para);

baseinit_para.period = 384;

baseinit_para.prescaler = SHRTIMER_PRESCALER_MUL64;

baseinit_para.repetitioncounter = 0;

baseinit_para.counter_mode = SHRTIMER_COUNTER_MODE_CONTINUOUS;

shrtimer_timers_base_init(SHRTIMER0, SHRTIMER_SLAVE_TIMER0, &baseinit_para);

```

### shrtimer\_timers\_counter\_enable

The description of shrtimer\_timers\_counter\_enable is shown as below:

**Table 3-495. Function shrtimer\_timers\_counter\_enable**

<b>Function name</b>	shrtimer_timers_counter_enable
<b>Function prototype</b>	void shrtimer_timers_counter_enable(uint32_t shrtimer_periph, uint32_t cntid);
<b>Function descriptions</b>	enable a counter
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
Input parameter{in}	
<b>cntid</b>	specify the counter to configure

<i>SHRTIMER_MT_COUNTER</i>	the counter of Master_TIMER
<i>SHRTIMER_STx_COUNTER(x=0..4)</i>	the counter of Slave_TIMERx(x=0..4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable a counter */
```

```
void shrtimer_timers_counter_enable(SHRTIMER0, SHRTIMER_MT_COUNTER);
```

### shrtimer\_timers\_counter\_disable

The description of shrtimer\_timers\_counter\_disable is shown as below:

**Table 3-496. Function shrtimer\_timers\_counter\_enable**

<b>Function name</b>	shrtimer_timers_counter_disable
<b>Function prototype</b>	void shrtimer_timers_counter_disable(uint32_t shrtimer_periph, uint32_t cntid);
<b>Function descriptions</b>	disable a counter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>cntid</b>	specify the counter to configure
<i>SHRTIMER_MT_COUNTER</i>	the counter of Master_TIMER
<i>SHRTIMER_STx_COUNTER(x=0..4)</i>	the counter of Slave_TIMERx(x=0..4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable a counter */
```

```
shrtimer_timers_counter_disable(SHRTIMER0, SHRTIMER_MT_COUNTER);
```



## shrtimer\_timers\_update\_event\_enable

The description of shrtimer\_timers\_update\_event\_enable is shown as below:

**Table 3-497. Function shrtimer\_timers\_update\_event\_enable**

<b>Function name</b>	shrtimer_timers_update_event_enable
<b>Function prototype</b>	void shrtimer_timers_update_event_enable(uint32_t shrtimer_periph, uint32_t timer_id);
<b>Function descriptions</b>	enable the Master_TIMER or Slave_TIMER update event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	SHRTIMER peripheral
SHRTIMER_MASTER_TIMER	the counter of Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	the counter of Slave_TIMERx(x=0..4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the Master_TIMER or Slave_TIMER update event */
```

```
shrtimer_timers_update_event_enable(SHRTIMER0, SHRTIMER_MASTER_TIMER);
```

## shrtimer\_timers\_update\_event\_disable

The description of shrtimer\_timers\_update\_event\_disable is shown as below:

**Table 3-498. Function shrtimer\_timers\_update\_event\_disable**

<b>Function name</b>	shrtimer_timers_update_event_disable
<b>Function prototype</b>	void shrtimer_timers_update_event_disable(uint32_t shrtimer_periph, uint32_t timer_id);
<b>Function descriptions</b>	disable the Master_TIMER or Slave_TIMER update event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
<b>Input parameter{in}</b>	

<b>timer_id</b>	SHRTIMER peripheral
<i>SHRTIMER_MASTER_TIMER</i>	the counter of Master_TIMER
<i>SHRTIMER_SLAVE_TIMERx(x=0..4)</i>	the counter of Slave_TIMERx(x=0..4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the Master_TIMER or Slave_TIMER update event */
```

```
shrtimer_timers_update_event_disable(SHRTIMER0, SHRTIMER_MASTER_TIMER);
```

## shrtimer\_software\_update

The description of shrtimer\_software\_update is shown as below:

**Table 3-499. Function shrtimer\_software\_update**

<b>Function name</b>	shrtimer_software_update
<b>Function prototype</b>	void shrtimer_software_update(uint32_t shrtimer_periph, uint32_t timersrc);
<b>Function descriptions</b>	update the Master_TIMER or Slave_TIMER by software
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timersrc</b>	software update timer selection
<i>SHRTIMER_UPDATE_SW_MT</i>	Master_TIMER software update
<i>SHRTIMER_UPDATE_SW_STx(x=0..4)</i>	Slave_TIMERx software update (x=0..4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* update the Master_TIMER or Slave_TIMER by software */
```

```
void shrtimer_software_update(SHRTIMER0, SHRTIMER_UPDATE_SW_MT);
```

## shrtimer\_software\_counter\_reset

The description of shrtimer\_software\_counter\_reset is shown as below:

**Table 3-500. Function shrtimer\_software\_counter\_reset**

<b>Function name</b>	shrtimer_software_counter_reset
<b>Function prototype</b>	void shrtimer_software_counter_reset(uint32_t shrtimer_periph, uint32_t timerrst);
<b>Function descriptions</b>	reset the Master_TIMER or Slave_TIMER counter by software
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timersrc</b>	software reset timer selection
SHRTIMER_COUNTER_RESET_SW_MT	Master_TIMER software reset
SHRTIMER_COUNTER_RESET_SW_STx (x=0..4)	Slave_TIMERx software reset(x=0..4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

/\* triggers the Master\_TIMER or Slave\_TIMER registers update by software \*/

```
void shrtimer_software_counter_reset(SHRTIMER0,
SHRTIMER_COUNTER_RESET_SW_MT);
```

## shrtimer\_timerinit\_struct\_para\_init

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

**Table 3-501. Function shrtimer\_timerinit\_struct\_para\_init**

<b>Function name</b>	shrtimer_timerinit_struct_para_init
<b>Function prototype</b>	void shrtimer_timerinit_struct_para_init(shrtimer_timerinit_parameter_struct* timerinit);
<b>Function descriptions</b>	initialize waveform mode initialization parameters struct with a default value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timerinit</b>	shrtimer_timerinit_parameter_struct, the structure members can refer to

	<a href="#">Table 3-479. Structure <i>shrtimer_timerinit_parameter_struct</i></a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize waveform mode initialization parameters struct with a default value */
```

```
shrtimer_timerinit_parameter_struct timerinit_para;
```

```
shrtimer_timerinit_struct_para_init(&timerinit_para);
```

### shrtimer\_timers\_waveform\_init

The description of shrtimer\_timers\_waveform\_init is shown as below:

**Table 3-502. Function shrtimer\_timers\_waveform\_init**

<b>Function name</b>	shrtimer_timers_waveform_init
<b>Function prototype</b>	void shrtimer_timers_waveform_init(uint32_t shrtimer_periph, uint32_t timer_id, shrtimer_timerinit_parameter_struct* timerinitpara);
<b>Function descriptions</b>	initialize a timer to work in waveform mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
Input parameter{in}	
<b>timer_id</b>	SHRTIMER peripheral
<i>SHRTIMER_MASTER_TIMER</i>	the counter of Master_TIMER
<i>SHRTIMER_SLAVE_TIMERx(x=0..4)</i>	the counter of Slave_TIMERx(x=0..4)
Input parameter{in}	
<b>timerinit</b>	shrtimer_timerinit_parameter_struct, the structure members can refer to <a href="#">Table 3-479. Structure <i>shrtimer_timerinit_parameter_struct</i></a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize a timer to work in waveform mode */
```

```
shrtimer_timerinit_parameter_struct timerinit_para;
```

```

shrtimer_timerinit_struct_para_init(&timerinit_para);

timerinit_para.cnt_bunch = SHRTIMER_TIMERBUNCHNMODE_MAINTAINCLOCK;

timerinit_para.DAC_trigger = SHRTIMER_DAC_TRIGGER_NONE;

timerinit_para.half_mode = SHRTIMER_HALFMODE_DISABLED;

timerinit_para.repetition_update = SHRTIMER_UPDATEONREPETITION_DISABLED;

timerinit_para.reset_sync = SHRTIMER_SYNCRESET_DISABLED;

timerinit_para.shadow = SHRTIMER_SHADOW_DISABLED;

timerinit_para.start_sync = SHRTIMER_SYNISTART_DISABLED;

timerinit_para.update_selection =
SHRTIMER_MT_ST_UPDATE_SELECTION_INDEPENDENT;

shrtimer_timers_waveform_init(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,
&timerinit_para);

```

### shrtimer\_timercfg\_struct\_para\_init

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

**Table 3-503. Function shrtimer\_timercfg\_struct\_para\_init**

Function name	shrtimer_timercfg_struct_para_init
Function prototype	void shrtimer_timercfg_struct_para_init(shrtimer_timercfg_parameter_struct* timercfg);
Function descriptions	initialize Slave_TIMER general behavior configuration struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
timercfg	shrtimer_timercfg_parameter_struct, the structure members can refer to <a href="#">Table 3-480. Structure shrtimer_timercfg_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize Slave_TIMER general behavior configuration struct with a default value */

shrtimer_timercfg_parameter_struct timercfg_para;

shrtimer_timercfg_struct_para_init(timercfg_para);

```

## shrtimer\_slavetimer\_waveform\_config

The description of shrtimer\_slavetimer\_waveform\_config is shown as below:

**Table 3-504. Function shrtimer\_slavetimer\_waveform\_config**

<b>Function name</b>	shrtimer_slavetimer_waveform_config
<b>Function prototype</b>	void shrtimer_slavetimer_waveform_config(uint32_t shrtimer_periph, uint32_t timer_id, shrtimer_timercfg_parameter_struct * timercfg);
<b>Function descriptions</b>	initialize Slave_TIMER general behavior configuration struct with a default value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	SHRTIMER peripheral
<i>SHRTIMER_MASTER_TIMER</i>	the counter of Master_TIMER
<i>SHRTIMER_SLAVE_TIMERx(x=0..4)</i>	the counter of Slave_TIMERx(x=0..4)
<b>Input parameter{in}</b>	
<b>timercfg</b>	shrtimer_timercfg_parameter_struct, the structure members can refer to <a href="#">Table 3-480. Structure shrtimer_timercfg_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* initialize Slave_TIMER general behavior configuration struct with a default value */
shrtimer_timercfg_parameter_struct timercfg_para;
shrtimer_timercfg_struct_para_init(&timercfg_para);
timercfg_para.balanced_mode = SHRTIMER_STXBALANCEDMODE_DISABLED;
timercfg_para.cnt_reset = SHRTIMER_STXCNT_RESET_NONE;
timercfg_para.deadtime_enable = SHRTIMER_STXDEADTIME_DISABLED;
timercfg_para.delayed_idle = SHRTIMER_STXDELAYED_IDLE_DISABLED;
timercfg_para.fault_enable = SHRTIMER_STXFAULTENABLE_NONE;
timercfg_para.fault_protect = SHRTIMER_STXFAULT_PROTECT_READWRITE;

```

```

timercfg_para.reset_update = SHRTIMER_STXUPDATEONRESET_DISABLED;

timercfg_para.update_source = SHRTIMER_STXUPDATETRIGGER_NONE;

shrtimer_slavetimer_waveform_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,
&timercfg_para);

```

### shrtimer\_comparecfg\_struct\_para\_init

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

**Table 3-505. Function shrtimer\_comparecfg\_struct\_para\_init**

Function name	shrtimer_comparecfg_struct_para_init
Function prototype	void shrtimer_comparecfg_struct_para_init(shrtimer_comparecfg_parameter_struct* comparecfg);
Function descriptions	initialize compare unit configuration struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
comparecfg	timer parameter initialization struct, the structure members can refer to <a href="#">Table 3-481. Structure shrtimer_comparecfg_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize compare unit configuration struct with a default value */

shrtimer_comparecfg_parameter_struct comparecfg_para;

shrtimer_comparecfg_struct_para_init(comparecfg_para);

```

### shrtimer\_slavetimer\_waveform\_compare\_config

The description of shrtimer\_slavetimer\_waveform\_compare\_config is shown as below:

**Table 3-506. Function shrtimer\_slavetimer\_waveform\_compare\_config**

Function name	shrtimer_slavetimer_waveform_compare_config
Function prototype	void shrtimer_slavetimer_waveform_compare_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t comparex, shrtimer_comparecfg_parameter_struct* cmpcfg);
Function descriptions	configure the compare unit of a Slave_TIMER which work in waveform mode
Precondition	-
The called functions	-

Input parameter{in}	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
Input parameter{in}	
<b>timer_id</b>	SHRTIMER peripheral
<i>SHRTIMER_MASTER_TIMER</i>	the counter of Master_TIMER
<i>SHRTIMER_SLAVE_TIMERx(x=0..4)</i>	the counter of Slave_TIMERx(x=0..4)
Input parameter{in}	
<b>comparex</b>	compare unit selection
<i>SHRTIMER_COMPAREy (y=0..4)</i>	the compare unit y(y=0..4)
Input parameter{in}	
<b>comparecfg</b>	timer parameter initialization struct, the structure members can refer to <a href="#">Table 3-481. Structure shrtimer_comparecfg_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the compare unit of a Slave_TIMER which work in waveform mode */
shrtimer_comparecfg_parameter_struct comparecfg_para;
shrtimer_comparecfg_struct_para_init(&comparecfg_para);
comparecfg_para.compare_value = 192;
shrtimer_slavetimer_waveform_compare_config(SHRTIMER0,
SHRTIMER_SLAVE_TIMER0, SHRTIMER_COMPARE0, &comparecfg_para);
```

## shrtimer\_channel\_outputcfg\_struct\_para\_init

The description of shrtimer\_channel\_outputcfg\_struct\_para\_init is shown as below:

**Table 3-507. Function shrtimer\_channel\_outputcfg\_struct\_para\_init**

<b>Function name</b>	shrtimer_channel_outputcfg_struct_para_init
<b>Function prototype</b>	void shrtimer_channel_outputcfg_struct_para_init(shrtimer_channel_outputcfg_parameter_struct * channelcfg);
<b>Function descriptions</b>	initialize channel output configuration struct with a default value
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	



<b>channelcfg</b>	shrtimer_channel_outputcfg_parameter_struct, the structure members can refer to <a href="#">Table 3-490. Structure <i>shrtimer_channel_outputcfg_parameter_struct</i></a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize channel output configuration struct with a default value */
```

```
shrtimer_channel_outputcfg_parameter_struct outcfg_para;
```

```
shrtimer_channel_outputcfg_struct_para_init(&outcfg_para);
```

### shrtimer\_slavetimer\_waveform\_channel\_config

The description of shrtimer\_slavetimer\_waveform\_channel\_config is shown as below:

**Table 3-508. Function shrtimer\_slavetimer\_waveform\_channel\_config**

<b>Function name</b>	shrtimer_slavetimer_waveform_channel_config
<b>Function prototype</b>	void shrtimer_slavetimer_waveform_channel_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t channel, shrtimer_channel_outputcfg_parameter_struct * channelcfg);
<b>Function descriptions</b>	configure the channel of a Slave_TIMER work in waveform mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	SHRTIMER timer index
<i>SHRTIMER_MASTER_TIMER</i>	the counter of Master_TIMER
<i>SHRTIMER_SLAVE_TIMERx(x=0..4)</i>	the counter of Slave_TIMERx(x=0..4)
<b>Input parameter{in}</b>	
<b>channel</b>	SHRTIMER timer channel index
<i>SHRTIMER_STx_CHy(x=0..4,y=0,1)</i>	SHRTIMER timer channel selection
<b>Input parameter{in}</b>	
<b>channelcfg</b>	shrtimer_channel_outputcfg_parameter_struct, the structure members can refer to <a href="#">Table 3-490. Structure <i>shrtimer_channel_outputcfg_parameter_struct</i></a>
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```

/* configure the channel of a Slave_TIMER work in waveform mode */

shrtimer_channel_outputcfg_parameter_struct outcfg_para;

shrtimer_channel_outputcfg_struct_para_init(&outcfg_para);

outcfg_para.carrier_mode = SHRTIMER_CHANNEL_CARRIER_DISABLED;

outcfg_para.deadtime_bunch = SHRTIMER_CHANNEL_BUNCH_ENTRY_REGULAR;

outcfg_para.fault_state = SHRTIMER_CHANNEL_FAULTSTATE_NONE;

outcfg_para.idle_bunch = SHRTIMER_CHANNEL_BUNCH_IDLE_DISABLE;

outcfg_para.idle_state = SHRTIMER_CHANNEL_IDLESTATE_INACTIVE;

outcfg_para.polarity = SHRTIMER_CHANNEL_POLARITY_HIGH;

outcfg_para.reset_request = SHRTIMER_CHANNEL_RESET_CMP1;

outcfg_para.set_request = SHRTIMER_CHANNEL_SET_CMP0;

shrtimer_slavetimer_waveform_channel_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,
SHRTIMER_ST0_CH0, &outcfg_para);

```

### shrtimer\_slavetimer\_waveform\_channel\_software\_request

The description of shrtimer\_slavetimer\_waveform\_channel\_software\_request is shown as below:

**Table 3-509. Function shrtimer\_slavetimer\_waveform\_channel\_software\_request**

<b>Function name</b>	shrtimer_slavetimer_waveform_channel_software_request
<b>Function prototype</b>	void shrtimer_slavetimer_waveform_channel_software_request(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t channel, uint32_t request)
<b>Function descriptions</b>	software generates channel "set request" or "reset request"
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	SHRTIMER timer index
SHRTIMER_MASTER_TIMER	the counter of Master_TIMER
SHRTIMER_SLAVE_TIMER	the counter of Slave_TIMERx(x=0..4)

<i>MERx(x=0..4)</i>	
<b>Input parameter{in}</b>	
<b>channel</b>	SHRTIMER timer channel index
<i>SHRTIMER_STx_CHy( x=0..4,y=0,1)</i>	SHRTIMER timer channel selection
<b>Input parameter{in}</b>	
<b>request</b>	request type selection
<i>SHRTIMER_CHANNEL _SOFTWARE_SET</i>	software event cannot generate request
<i>SHRTIMER_CHANNEL _SOFTWARE_RESET</i>	software event can generate request
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* software generates channel "set request" or "reset request" */
```

```
shrtimer_slavetimer_waveform_channel_software_request (SHRTIMER0,  
SHRTIMER_SLAVE_TIMER0, SHRTIMER_ST0_CH0,  
SHRTIMER_CHANNEL_SOFTWARE_SET);
```

### shrtimer\_slavetimer\_waveform\_channel\_output\_level\_get

The description of shrtimer\_slavetimer\_waveform\_channel\_output\_level\_get is shown as below:

**Table 3-510. Function shrtimer\_slavetimer\_waveform\_channel\_output\_level\_get**

<b>Function name</b>	shrtimer_slavetimer_waveform_channel_output_level_get
<b>Function prototype</b>	uint32_t shrtimer_slavetimer_waveform_channel_output_level_get(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t channel);
<b>Function descriptions</b>	get Slave_TIMER channel output level
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	SHRTIMER timer index
<i>SHRTIMER_MASTER_ TIMER</i>	the counter of Master_TIMER
<i>SHRTIMER_SLAVE_TI MERx(x=0..4)</i>	the counter of Slave_TIMERx(x=0..4)

Input parameter{in}	
<b>channel</b>	SHRTIMER timer channel index
<i>SHRTIMER_STx_CHy(x=0..4,y=0,1)</i>	SHRTIMER timer channel selection
Output parameter{out}	
-	-
Return value	
<b>uint32_t</b>	channel output level

Example:

```
/* get Slave_TIMER channel output level */
```

```
uint32_t output_level;
```

```
output_level = shrtimer_slavetimer_waveform_channel_output_level_get(uint32_t
shrtimer_periph, uint32_t timer_id, uint32_t channel)
```

### shrtimer\_slavetimer\_waveform\_channel\_state\_get

The description of shrtimer\_slavetimer\_waveform\_channel\_state\_get is shown as below:

**Table 3-511. Function shrtimer\_slavetimer\_waveform\_channel\_state\_get**

<b>Function name</b>	shrtimer_slavetimer_waveform_channel_state_get
<b>Function prototype</b>	uint32_t shrtimer_slavetimer_waveform_channel_state_get(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t channel)
<b>Function descriptions</b>	get Slave_TIMER channel run state
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
Input parameter{in}	
<b>timer_id</b>	SHRTIMER timer index
<i>SHRTIMER_MASTER_TIMER</i>	the counter of Master_TIMER
<i>SHRTIMER_SLAVE_TIMERx(x=0..4)</i>	the counter of Slave_TIMERx(x=0..4)
Input parameter{in}	
<b>channel</b>	SHRTIMER timer channel index
<i>SHRTIMER_STx_CHy(x=0..4,y=0,1)</i>	SHRTIMER timer channel selection
Output parameter{out}	
-	-
Return value	
<b>uint32_t</b>	SHRTIMER_CHANNEL_STATE_IDLE or

	SHRTIMER_CHANNEL_STATE_RUN or SHRTIMER_CHANNEL_STATE_FAULT
--	---

Example:

```
/* get Slave_TIMER channel run state */

uint32_t output_state;

output_state = shrtimer_slavetimer_waveform_channel_state_get (SHRTIMER0,
SHRTIMER_SLAVE_TIMER0, SHRTIMER_ST0_CH0);
```

### shrtimer\_deadtimercfg\_struct\_para\_init

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

**Table 3-512. Function shrtimer\_channel\_outputcfg\_struct\_para\_init**

<b>Function name</b>	shrtimer_deadtimercfg_struct_para_init
<b>Function prototype</b>	void shrtimer_deadtimercfg_struct_para_init(shrtimer_deadtimercfg_parameter_struct * dtcfg);
<b>Function descriptions</b>	initialize dead time configuration struct with a default value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dtcfg</b>	shrtimer_deadtimercfg_parameter_struct, the structure members can refer to <a href="#">Table 3-483. Structure shrtimer_deadtimercfg_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize dead time configuration struct with a default value */

shrtimer_deadtimercfg_parameter_struct deadtimecfg_para;

shrtimer_deadtimercfg_struct_para_init(&deadtimecfg_para);
```

### shrtimer\_slavetimer\_deadtime\_config

The description of shrtimer\_slavetimer\_deadtime\_config is shown as below:

**Table 3-513. Function shrtimer\_slavetimer\_waveform\_channel\_software\_request**

<b>Function name</b>	shrtimer_slavetimer_deadtime_config
<b>Function prototype</b>	void shrtimer_slavetimer_deadtime_config(uint32_t shrtimer_periph, uint32_t timer_id, shrtimer_deadtimercfg_parameter_struct* dtcfg)
<b>Function descriptions</b>	configure the dead time for Slave_TIMER

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<b>SHRTIMERx(x=0)</b>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	SHRTIMER timer index
<b>SHRTIMER_SLAVE_TIMERx(x=0..4)</b>	the counter of Slave_TIMERx(x=0..4)
<b>Input parameter{in}</b>	
<b>dctcfg</b>	shrtimer_deadtimecfg_parameter_struct, the structure members can refer to <a href="#">Table 3-483. Structure shrtimer_deadtimecfg_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure the dead time for Slave_TIMER */

shrtimer_deadtimecfg_parameter_struct deadtimecfg_para;

shrtimer_deadtimecfg_struct_para_init(&deadtimecfg_para);

deadtimecfg_para.fallingsign_protect =
SHRTIMER_DEADTIME_FALLINGSIGN_PROTECT_DISABLE;

deadtimecfg_para.falling_protect =
SHRTIMER_DEADTIME_FALLING_PROTECT_DISABLE;

deadtimecfg_para.falling_sign = SHRTIMER_DEADTIME_FALLINGSIGN_POSITIVE;

deadtimecfg_para.falling_value = 0x0040;

deadtimecfg_para.prescaler = SHRTIMER_DEADTIME_PRESCALER_MUL8;

deadtimecfg_para.risingsign_protect =
SHRTIMER_DEADTIME_RISINGSIGN_PROTECT_DISABLE;

deadtimecfg_para.rising_protect = SHRTIMER_DEADTIME_RISING_PROTECT_DISABLE;

deadtimecfg_para.rising_sign = SHRTIMER_DEADTIME_RISINGSIGN_POSITIVE;

deadtimecfg_para.rising_value = 0x0040;

shrtimer_slavetimer_deadtime_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,
&deadtimecfg_para);

```

## shrtimer\_carriersignalcfg\_struct\_para\_init

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

**Table 3-514. Function shrtimer\_channel\_outputcfg\_struct\_para\_init**

<b>Function name</b>	shrtimer_carriersignalcfg_struct_para_init
<b>Function prototype</b>	void shrtimer_carriersignalcfg_struct_para_init(shrtimer_carriersignalcfg_parameter_struct* carriercfg);
<b>Function descriptions</b>	initialize carrier signal configuration struct with a default value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>carriercfg</b>	shrtimer_carriersignalcfg_parameter_struct, the structure members can refer to <a href="#">Table 3-484. Structure shrtimer_carriersignalcfg_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize carrier signal configuration struct with a default value */
```

```
shrtimer_carriersignalcfg_parameter_struct carriercfg;
```

```
shrtimer_carriersignalcfg_struct_para_init(&carriercfg);
```

## shrtimer\_slavetimer\_carriersignal\_config

The description of shrtimer\_slavetimer\_carriersignal\_config is shown as below:

**Table 3-515. Function shrtimer\_slavetimer\_carriersignal\_config**

<b>Function name</b>	shrtimer_slavetimer_carriersignal_config
<b>Function prototype</b>	void shrtimer_slavetimer_carriersignal_config(uint32_t shrtimer_periph, uint32_t timer_id, shrtimer_carriersignalcfg_parameter_struct* carriercfg);
<b>Function descriptions</b>	configure the carrier signal mode for Slave_TIMER
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<b>SHRTIMERx(x=0)</b>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	SHRTIMER timer index
<b>SHRTIMER_SLAVE_TIMERx(x=0..4)</b>	the counter of Slave_TIMERx(x=0..4)

Input parameter{in}	
<b>carriercfg</b>	shrtimer_carriersignalcfg_parameter_struct, the structure members can refer to <a href="#">Table 3-484. Structure shrtimer_carriersignalcfg_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the carrier signal mode for Slave_TIMER */
shrtimer_carriersignalcfg_parameter_struct carriercfg;
shrtimer_carriersignalcfg_struct_para_init(&carriercfg);
shrtimer_slavetimer_carriersignal_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,
&carriercfg);
```

## shrtimer\_output\_channel\_enable

The description of shrtimer\_output\_channel\_enable is shown as below:

**Table 3-516. Function shrtimer\_output\_channel\_enable**

Function name	shrtimer_output_channel_enable
Function prototype	void shrtimer_output_channel_enable(uint32_t shrtimer_periph, uint32_t chid);
Function descriptions	enable a output channel
Precondition	-
The called functions	-
Input parameter{in}	
<b>shrtimer_periph</b>	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
<b>ch_id</b>	SHRTIMER timer channel index
SHRTIMER_STx_CHy(x=0..4;y=0,1)	timer channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable a output channel */
void shrtimer_output_channel_enable(SHRTIMER0, SHRTIMER_ST0_CH0);
```



## shrtimer\_output\_channel\_disable

The description of shrtimer\_output\_channel\_disable is shown as below:

**Table 3-517. Function shrtimer\_output\_channel\_disable**

<b>Function name</b>	shrtimer_output_channel_disable
<b>Function prototype</b>	void shrtimer_output_channel_disable(uint32_t shrtimer_periph, uint32_t chid);
<b>Function descriptions</b>	disable a output channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>ch_id</b>	SHRTIMER timer channel index
<i>SHRTIMER_STx_CHy(x=0..4;y=0,1)</i>	timer channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable a output channel */
```

```
void shrtimer_output_channel_disable(SHRTIMER0, SHRTIMER_ST0_CH0);
```

## shrtimer\_slavetimer\_compare\_value\_config

The description of shrtimer\_slavetimer\_compare\_value\_config is shown as below:

**Table 3-518. Function shrtimer\_slavetimer\_waveform\_compare\_config**

<b>Function name</b>	shrtimer_slavetimer_compare_value_config
<b>Function prototype</b>	void shrtimer_slavetimer_compare_value_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t comparex, uint32_t cmpvalue);
<b>Function descriptions</b>	configure the compare value in Slave_TIMER
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	SHRTIMER peripheral
<i>SHRTIMER_SLAVE_TI</i>	the counter of Slave_TIMERx(x=0..4)

<i>MERx(x=0..4)</i>	
<b>Input parameter{in}</b>	
<b>comparex</b>	compare unit selection
<i>SHRTIMER_COMPAR Ey (y=0..4)</i>	the compare unit y(y=0..4)
<b>Input parameter{in}</b>	
<b>cmpvalue</b>	the compare value from 3 tSHRTIMER_CK clock to 0xFFFF - (1 tSHRTIMER_CK)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the compare value in Slave_TIMER */
```

```
shrtimer_slavetimer_compare_value_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,  
SHRTIMER_COMPARE0, 3);
```

### shrtimer\_slavetimer\_compare\_value\_get

The description of shrtimer\_slavetimer\_compare\_value\_get is shown as below:

**Table 3-519. Function shrtimer\_slavetimer\_compare\_value\_get**

<b>Function name</b>	shrtimer_slavetimer_compare_value_get
<b>Function prototype</b>	uint32_t shrtimer_slavetimer_compare_value_get(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t comparex)
<b>Function descriptions</b>	get the compare value in Slave_TIMER
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	SHRTIMER peripheral
<i>SHRTIMER_SLAVE_TIMERx(x=0..4)</i>	the counter of Slave_TIMERx(x=0..4)
<b>Input parameter{in}</b>	
<b>comparex</b>	compare unit selection
<i>SHRTIMER_COMPAR Ey (y=0..4)</i>	the compare unit y(y=0..4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

uint32_t	the compare value
----------	-------------------

Example:

```
/* get the compare value in Slave_TIMER */

uint32_t cmpvalue;

cmpvalue = shrtimer_slavetimer_compare_value_get (SHRTIMER0,
SHRTIMER_SLAVE_TIMER0, SHRTIMER_COMPARE0);
```

### shrtimer\_mastertimer\_compare\_value\_config

The description of shrtimer\_mastertimer\_compare\_value\_config is shown as below:

**Table 3-520. Function shrtimer\_mastertimer\_compare\_value\_config**

<b>Function name</b>	shrtimer_mastertimer_compare_value_config
<b>Function prototype</b>	void shrtimer_mastertimer_compare_value_config(uint32_t shrtimer_periph, uint32_t comparex, uint32_t cmpvalue);
<b>Function descriptions</b>	configure the compare value in Master_TIMER
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>comparex</b>	compare unit selection
<i>SHRTIMER_COMPAR Ey (y=0..4)</i>	the compare unit y(y=0..4)
<b>Input parameter{in}</b>	
<b>cmpvalue</b>	the compare value from 3 tSHRTIMER_CK clock to 0xFFFF - (1 tSHRTIMER_CK)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the compare value in Master_TIMER */

shrtimer_mastertimer_compare_value_config(SHRTIMER0, SHRTIMER_COMPARE0, 3);
```

### shrtimer\_mastertimer\_compare\_value\_get

The description of shrtimer\_mastertimer\_compare\_value\_get is shown as below:

Table 3-521. Function `shrtimer_slavetimer_compare_value_get`

<b>Function name</b>	<code>shrtimer_mastertimer_compare_value_get</code>
<b>Function prototype</b>	<code>uint32_t shrtimer_mastertimer_compare_value_get(uint32_t shrtimer_periph, uint32_t comparex);</code>
<b>Function descriptions</b>	get the compare value in Master_TIMER
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>comparex</b>	compare unit selection
<i>SHRTIMER_COMPAR Ey (y=0..4)</i>	the compare unit y(y=0..4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	the compare value

Example:

```
/* get the compare value in Master_TIMER */
```

```
uint32_t cmpvalue;
```

```
cmpvalue = shrtimer_mastertimer_compare_value_get(SHRTIMER0,
SHRTIMER_COMPARE0)
```

### **shrtimer\_timers\_counter\_value\_config**

The description of `shrtimer_timers_counter_value_config` is shown as below:

Table 3-522. Function `shrtimer_timers_counter_value_config`

<b>Function name</b>	<code>shrtimer_timers_counter_value_config</code>
<b>Function prototype</b>	<code>void shrtimer_timers_counter_value_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t cntvalue);</code>
<b>Function descriptions</b>	configure the counter value in Master_TIMER and Slave_TIMER
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	SHRTIMER peripheral
<i>SHRTIMER_MASTER_TIMER</i>	the counter of Master_TIMER

<i>SHRTIMER_SLAVE_TIMERx(x=0..4)</i>	the counter of Slave_TIMERx(x=0..4)
<b>Input parameter{in}</b>	
<b>cntvalue</b>	the value ranges from 0 to 0xffff
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the counter value in Master_TIMER and Slave_TIMER */
```

```
shrtimer_timers_counter_value_config(SHRTIMER0, SHRTIMER_MASTER_TIMER, 100);
```

### shrtimer\_timers\_counter\_value\_get

The description of shrtimer\_timers\_counter\_value\_get is shown as below:

**Table 3-523. Function shrtimer\_timers\_counter\_value\_get**

<b>Function name</b>	shrtimer_timers_counter_value_get
<b>Function prototype</b>	uint32_t shrtimer_timers_counter_value_get(uint32_t shrtimer_periph, uint32_t timer_id);
<b>Function descriptions</b>	get the counter value in Master_TIMER and Slave_TIMER
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	SHRTIMER peripheral
<i>SHRTIMER_MASTER_TIMER</i>	the counter of Master_TIMER
<i>SHRTIMER_SLAVE_TIMERx(x=0..4)</i>	the counter of Slave_TIMERx(x=0..4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	the counter value

Example:

```
/* get the counter value in Master_TIMER and Slave_TIMER */
```

```
uint32_t value;
```

```
value = shrtimer_timers_counter_value_get(SHRTIMER0, SHRTIMER_MASTER_TIMER);
```

## shrtimer\_timers\_autoreload\_value\_config

The description of shrtimer\_timers\_autoreload\_value\_config is shown as below:

**Table 3-524. Function shrtimer\_timers\_autoreload\_value\_config**

<b>Function name</b>	shrtimer_timers_autoreload_value_config
<b>Function prototype</b>	void shrtimer_timers_autoreload_value_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t carlvalue);
<b>Function descriptions</b>	configure the counter auto reload value in Master_TIMER and Slave_TIMER
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	SHRTIMER peripheral
<i>SHRTIMER_MASTER_TIMER</i>	the counter of Master_TIMER
<i>SHRTIMER_SLAVE_TIMERx(x=0..4)</i>	the counter of Slave_TIMERx(x=0..4)
<b>Input parameter{in}</b>	
<b>carlvalue</b>	the value ranges from 0 to 0xffff
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the counter auto reload value in Master_TIMER and Slave_TIMER */
```

```
shrtimer_timers_autoreload_value_config(SHRTIMER0, SHRTIMER_MASTER_TIMER, 100);
```

## shrtimer\_timers\_autoreload\_value\_get

The description of shrtimer\_timers\_autoreload\_value\_get is shown as below:

**Table 3-525. Function shrtimer\_timers\_autoreload\_value\_get**

<b>Function name</b>	shrtimer_timers_autoreload_value_get
<b>Function prototype</b>	uint32_t shrtimer_timers_autoreload_value_get(uint32_t shrtimer_periph, uint32_t timer_id)
<b>Function descriptions</b>	get the counter auto reload value in Master_TIMER and Slave_TIMER
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
Input parameter{in}	
<b>timer_id</b>	SHRTIMER peripheral
<i>SHRTIMER_MASTER_TIMER</i>	the counter of Master_TIMER
<i>SHRTIMER_SLAVE_TIMERx(x=0..4)</i>	the counter of Slave_TIMERx(x=0..4)
Output parameter{out}	
-	-
Return value	
<b>uint32_t</b>	the counter value

Example:

```
/* get the counter auto reload value in Master_TIMER and Slave_TIMER */
```

```
uint32_t value;
```

```
value = shrtimer_timers_autoreload_value_get(SHRTIMER0,
SHRTIMER_MASTER_TIMER);
```

### shrtimer\_timers\_repetition\_value\_config

The description of shrtimer\_timers\_repetition\_value\_config is shown as below:

**Table 3-526. Function shrtimer\_timers\_repetition\_value\_config**

<b>Function name</b>	shrtimer_timers_repetition_value_config
<b>Function prototype</b>	void shrtimer_timers_repetition_value_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t replvalue);
<b>Function descriptions</b>	configure the counter repetition value in Master_TIMER and Slave_TIMER
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
Input parameter{in}	
<b>timer_id</b>	SHRTIMER peripheral
<i>SHRTIMER_MASTER_TIMER</i>	the counter of Master_TIMER
<i>SHRTIMER_SLAVE_TIMERx(x=0..4)</i>	the counter of Slave_TIMERx(x=0..4)
Input parameter{in}	
<b>replvalue</b>	the value ranges from 0 to 0xffff
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* configure the counter repetition value in Master_TIMER and Slave_TIMER */
shrtimer_timers_repetition_value_config(SHRTIMER0, SHRTIMER_MASTER_TIMER, 100);
```

### shrtimer\_timers\_repetition\_value\_get

The description of shrtimer\_timers\_repetition\_value\_get is shown as below:

**Table 3-527. Function shrtimer\_timers\_repetition\_value\_get**

<b>Function name</b>	shrtimer_timers_repetition_value_get
<b>Function prototype</b>	uint32_t shrtimer_timers_repetition_value_get(uint32_t shrtimer_periph, uint32_t timer_id);
<b>Function descriptions</b>	get the counter repetition value in Master_TIMER and Slave_TIMER
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	SHRTIMER peripheral
<i>SHRTIMER_MASTER_TIMER</i>	the counter of Master_TIMER
<i>SHRTIMER_SLAVE_TIMERx(x=0..4)</i>	the counter of Slave_TIMERx(x=0..4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	the counter value

Example:

```
/* get the counter repetition value in Master_TIMER and Slave_TIMER */
uint32_t value;
value = shrtimer_timers_repetition_value_get (SHRTIMER0, SHRTIMER_MASTER_TIMER);
```

### shrtimer\_exefilter\_struct\_para\_init

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



**Table 3-528. Function shrtimer\_exefilter\_struct\_para\_init**

<b>Function name</b>	shrtimer_exefilter_struct_para_init
<b>Function prototype</b>	void shrtimer_exefilter_struct_para_init(shrtimer_exefilter_parameter_struct * exefilter);
<b>Function descriptions</b>	initialize external event filtering for Slave_TIMER configuration struct with a default value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exefilter</b>	shrtimer_exefilter_parameter_struct * exefilter, the structure members can refer to <a href="#">Table 3-482. Structure shrtimer_exefilter_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize external event filtering for Slave_TIMER configuration struct with a default value */
shrtimer_exefilter_parameter_struct exefilter;
shrtimer_exefilter_struct_para_init(&exefilter);
```

## shrtimer\_slavetimer\_exeevent\_filtering\_config

The description of shrtimer\_slavetimer\_exeevent\_filtering\_config is shown as below:

**Table 3-529. Function shrtimer\_slavetimer\_exeevent\_filtering\_config**

<b>Function name</b>	shrtimer_slavetimer_exeevent_filtering_config
<b>Function prototype</b>	void shrtimer_slavetimer_exeevent_filtering_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t event_id, shrtimer_exefilter_parameter_struct *exefilter);
<b>Function descriptions</b>	configure the external event filtering for Slave_TIMER (blanking, windowing)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<b>SHRTIMERx(x=0)</b>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	SHRTIMER timer index
<b>SHRTIMER_SLAVE_TIMERx(x=0..4)</b>	the counter of Slave_TIMERx(x=0..4)
<b>Input parameter{in}</b>	
<b>event_id</b>	SHRTIMER event index

SHRTIMER_EXEVENT _NONE	the counter of Slave_TIMERx <i>undefined event channel</i>
SHRTIMER_EXEVENT _y(y=0..9)	extern event y(y=0..9)
<b>Input parameter{in}</b>	
<b>exevfilter</b>	shrtimer_exevfilter_parameter_struct * exevfilter, the structure members can refer to <a href="#">Table 3-482. Structure shrtimer_exevfilter_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure the external event filtering for Slave_TIMER (blanking, windowing) */

shrtimer_exevfilter_parameter_struct exevfilter;

shrtimer_exevfilter_struct_para_init(&exevfilter);

exevfilter.filter_mode = SHRTIMER_EXEVFILTER_BLANKINGCMP1;

exevfilter.memorized = SHRTIMER_EXEVMEMORIZED_DISABLE;

shrtimer_slavetimer_exeevent_filtering_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,
SHRTIMER_EXEVENT_5, &exevfilter);

```

### shrtimer\_exeeventcfg\_struct\_para\_init

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

**Table 3-530. Function shrtimer\_exeeventcfg\_struct\_para\_init**

<b>Function name</b>	shrtimer_exeeventcfg_struct_para_init
<b>Function prototype</b>	void shrtimer_exeeventcfg_struct_para_init(shrtimer_exeeventcfg_parameter_struct * exevcfg);
<b>Function descriptions</b>	initialize external event configuration struct with a default value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exevcfg</b>	shrtimer_exeeventcfg_parameter_struct, the structure members can refer to <a href="#">Table 3-487. Structure shrtimer_exeeventcfg_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize external event configuration struct with a default value */
```

```
shrtimer_exeventcfg_parameter_struct exevcfg;
```

```
shrtimer_exeventcfg_struct_para_init(&exevcfg);
```

## shrtimer\_exevent\_config

The description of shrtimer\_exevent\_config is shown as below:

**Table 3-531. Function shrtimer\_exevent\_config**

<b>Function name</b>	shrtimer_exevent_config
<b>Function prototype</b>	void shrtimer_exevent_config(uint32_t shrtimer_periph, uint32_t event_id, shrtimer_exeventcfg_parameter_struct* exevcfg);
<b>Function descriptions</b>	configure the an external event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>event_id</b>	SHRTIMER event index
<i>SHRTIMER_EXEVENT_NONE</i>	the counter of Slave_TIMERx <i>undefined event channel</i>
<i>SHRTIMER_EXEVENT_y(y=0..9)</i>	extern event y(y=0..9)
<b>Input parameter{in}</b>	
<b>exevcfg</b>	shrtimer_exeventcfg_parameter_struct, the structure members can refer to <a href="#">Table 3-487. Structure shrtimer_exeventcfg_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the an external event */
```

```
shrtimer_exeventcfg_parameter_struct exevcfg;
```

```
shrtimer_exeventcfg_struct_para_init(&exevcfg);
```

```
exevcfg.digital_filter = 0x5;
```

```
exevcfg.edge = SHRTIMER_EXEV_EDGE_RISING;
```

```
exevcfg.polarity = SHRTIMER_EXEV_EDGE_LEVEL;
```

```
exevcfg.source = SHRTIMER_EXEV_SRC0;
```

```
shrtimer_exeevent_config(SHRTIMER0, SHRTIMER_EXEVENT_5, &exevcfg);
```

## shrtimer\_exeevent\_prescaler

The description of shrtimer\_exeevent\_prescaler is shown as below:

**Table 3-532. Function shrtimer\_exeevent\_prescaler**

<b>Function name</b>	shrtimer_exeevent_prescaler
<b>Function prototype</b>	void shrtimer_exeevent_prescaler(uint32_t shrtimer_periph, uint32_t prescaler);
<b>Function descriptions</b>	configure external event digital filter clock division
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<b>SHRTIMERx(x=0)</b>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>prescaler</b>	clock division value
<b>SHRTIMER_EXEV_PRESCALER_DIVx(x=1,2,4,8)</b>	$f_{SHRTIMER\_EXEV\_FCK} = f_{SHRTIMER\_CK}/x$ (x=1,2,4,8)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure external event digital filter clock division */
```

```
shrtimer_exeevent_prescaler(SHRTIMER0, SHRTIMER_EXEV_PRESCALER_DIV1);
```

## shrtimer\_synccfg\_struct\_para\_init

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

**Table 3-533. Function shrtimer\_synccfg\_struct\_para\_init**

<b>Function name</b>	shrtimer_synccfg_struct_para_init
<b>Function prototype</b>	void shrtimer_synccfg_struct_para_init(shrtimer_synccfg_parameter_struct* synccfg);
<b>Function descriptions</b>	initialize synchronization configuration struct with a default value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>synccfg</b>	shrtimer_synccfg_parameter_struct, the structure members can refer to <a href="#">Table 3-485. Structure shrtimer_synccfg_parameter_struct</a>

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize synchronization configuration struct with a default value */
```

```
shrtimer_synccfg_parameter_struct synccfg;
```

```
shrtimer_synccfg_struct_para_init (&synccfg);
```

### shrtimer\_synchronization\_config

The description of shrtimer\_synchronization\_config is shown as below:

**Table 3-534. Function shrtimer\_synchronization\_config**

Function name	shrtimer_synchronization_config
Function prototype	void shrtimer_synchronization_config(uint32_t shrtimer_periph, shrtimer_synccfg_parameter_struct* synccfg);
Function descriptions	configure the synchronization input/output of the SHRTIMER
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
synccfg	shrtimer_synccfg_parameter_struct, the structure members can refer to <a href="#">Table 3-485. Structure shrtimer_synccfg_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the synchronization input/output of the SHRTIMER */
```

```
shrtimer_synccfg_parameter_struct synccfg;
```

```
shrtimer_synccfg_struct_para_init (&synccfg);
```

```
synccfg.input_source = SHRTIMER_SYNCINPUTSOURCE_EXTERNAL;
```

```
shrtimer_synchronization_config(SHRTIMER0, &synccfg);
```

### shrtimer\_faultcfg\_struct\_para\_init

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

Table 3-535. Function `shrtimer_faultcfg_struct_para_init`

Function name	<code>shrtimer_faultcfg_struct_para_init</code>
Function prototype	<code>void shrtimer_faultcfg_struct_para_init(shrtimer_faultcfg_parameter_struct * faultcfg);</code>
Function descriptions	initialize fault input configuration struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
<code>faultcfg</code>	<code>shrtimer_faultcfg_parameter_struct</code> , the structure members can refer to <a href="#">Table 3-488. Structure <code>shrtimer_faultcfg_parameter_struct</code></a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize fault input configuration struct with a default value */
```

```
shrtimer_faultcfg_parameter_struct faultcfg;
```

```
shrtimer_synccfg_struct_para_init (&faultcfg);
```

## shrtimer\_fault\_config

The description of `shrtimer_fault_config` is shown as below:

Table 3-536. Function `shrtimer_fault_config`

Function name	<code>shrtimer_fault_config</code>
Function prototype	<code>void shrtimer_fault_config(uint32_t shrtimer_periph, uint32_t fault_id, shrtimer_faultcfg_parameter_struct* faultcfg);</code>
Function descriptions	configure the fault input
Precondition	-
The called functions	-
Input parameter{in}	
<code>shrtimer_periph</code>	SHRTIMER peripheral
<code>SHRTIMERx(x=0)</code>	SHRTIMER selection
Input parameter{in}	
<code>fault_id</code>	SHRTIMER fault index
<code>SHRTIMER_FAULT_Y(y=0..4)</code>	SHRTIMER fault selection
Input parameter{in}	
<code>faultcfg</code>	<code>shrtimer_faultcfg_parameter_struct</code> , the structure members can refer to <a href="#">Table 3-488. Structure <code>shrtimer_faultcfg_parameter_struct</code></a>
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure the fault input */

shrtimer_faultcfg_parameter_struct faultcfg;

shrtimer_synccfg_struct_para_init (&faultcfg);

faultcfg_para.control = SHRTIMER_FAULT_CHANNEL_ENABLE;

faultcfg_para.filter = 0x0;

faultcfg_para.polarity = SHRTIMER_FAULT_POLARITY_HIGH;

faultcfg_para.protect = SHRTIMER_FAULT_PROTECT_ENABLE;

faultcfg_para.source = SHRTIMER_FAULT_SOURCE_PIN;

shrtimer_fault_config(SHRTIMER0, SHRTIMER_FAULT_0, &faultcfg);
```

## shrtimer\_fault\_prescaler\_config

The description of shrtimer\_fault\_prescaler\_config is shown as below:

**Table 3-537. Function shrtimer\_fault\_prescaler\_config**

<b>Function name</b>	shrtimer_fault_prescaler_config
<b>Function prototype</b>	void shrtimer_fault_prescaler_config(uint32_t shrtimer_periph, uint32_t prescaler);
<b>Function descriptions</b>	configure the fault input digital filter clock division
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>prescaler</b>	clock division value
<i>SHRTIMER_FAULT_PRESALER_DIVy(y=1, 2, 4, 8)</i>	$f_{SHRTIMER\_FLTCK} = f_{SHRTIMER\_CK}/y$ (y=1,2,4,8)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the fault input digital filter clock division */
```

```
void shrtimer_fault_prescaler_config(SHRTIMER0,
SHRTIMER_FAULT_PRESCALER_DIV1);
```

## shrtimer\_fault\_input\_enable

The description of shrtimer\_fault\_input\_enable is shown as below:

**Table 3-538. Function shrtimer\_fault\_input\_enable**

<b>Function name</b>	shrtimer_fault_input_enable
<b>Function prototype</b>	void shrtimer_fault_input_enable(uint32_t shrtimer_periph, uint32_t fault_id);
<b>Function descriptions</b>	fault input enable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>fault_id</b>	SHRTIMER fault index
<i>SHRTIMER_FAULT_y(y=0..4)</i>	SHRTIMER fault selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* fault input enable */
shrtimer_fault_input_enable(SHRTIMER0, SHRTIMER_FAULT_0);
```

## shrtimer\_fault\_input\_disable

The description of shrtimer\_fault\_input\_disable is shown as below:

**Table 3-539. Function shrtimer\_fault\_input\_disable**

<b>Function name</b>	shrtimer_fault_input_disable
<b>Function prototype</b>	void shrtimer_fault_input_disable(uint32_t shrtimer_periph, uint32_t fault_id);
<b>Function descriptions</b>	fault input disable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection



Input parameter{in}	
<b>fault_id</b>	SHRTIMER fault index
<i>SHRTIMER_FAULT_y</i> ( <i>y=0..4</i> )	SHRTIMER fault selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* fault input disable */
```

```
shrtimer_fault_input_disable(SHRTIMER0, SHRTIMER_FAULT_0);
```

## shrtimer\_timers\_dma\_enable

The description of shrtimer\_timers\_dma\_enable is shown as below:

**Table 3-540. Function shrtimer\_timers\_dma\_enable**

<b>Function name</b>	shrtimer_timers_dma_enable
<b>Function prototype</b>	void shrtimer_timers_dma_enable(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t dmareq);
<b>Function descriptions</b>	enable the Master_TIMER and Slave_TIMER DMA request
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx</i> ( <i>x=0</i> )	SHRTIMER selection
Input parameter{in}	
<b>timer_id</b>	master timer and slave timer index
<i>SHRTIMER_MASTER_TIMER</i>	the master timer
<i>SHRTIMER_SLAVE_TIMERx</i>	the slave timer selection( <i>x=0..4</i> )
Input parameter{in}	
<b>dmareq</b>	DMA request source
<i>SHRTIMER_MT_ST_DMA_CMPy</i> ( <i>y=0..3</i> )	compare y DMA request, for Master_TIMER and Slave_TIMER ( <i>y=0..3</i> )
<i>SHRTIMER_MT_ST_DMA_REP</i>	repetition DMA request, for Master_TIMER and Slave_TIMER
<i>SHRTIMER_MT_DMA_SYNID</i>	synchronization input DMA request, for Master_TIMER
<i>SHRTIMER_MT_ST_DMA_UPD</i>	update DMA request, for Master_TIMER and Slave_TIMER

<i>SHRTIMER_ST_DMA_CAPy(y=0,1)</i>	capture y DMA request, for Slave_TIMER(y=0,1)
<i>SHRTIMER_ST_DMA_CHyOA(y=0,1)</i>	channel y output active DMA request, for Slave_TIMER(y=0,1)
<i>SHRTIMER_ST_DMA_CHyONA(y=0,1)</i>	channel y output inactive DMA request, for Slave_TIMER(y=0,1)
<i>SHRTIMER_ST_DMA_CNTRST</i>	counter reset DMA request, for Slave_TIMER
<i>SHRTIMER_ST_DMA_DLYIDLE</i>	delayed IDLE mode entry DMA request, for Slave_TIMER
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the Master_TIMER and Slave_TIMER DMA request */
```

```
shrtimer_timers_dma_enable(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_DMA_CMP0);
```

### shrtimer\_timers\_dma\_disable

The description of shrtimer\_timers\_dma\_disable is shown as below:

**Table 3-541. Function shrtimer\_timers\_dma\_enable**

<b>Function name</b>	shrtimer_timers_dma_disable
<b>Function prototype</b>	void shrtimer_timers_dma_disable(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t dmareq);
<b>Function descriptions</b>	disable the Master_TIMER and Slave_TIMER DMA request
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	master timer and slave timer index
<i>SHRTIMER_MASTER_TIMER</i>	the master timer
<i>SHRTIMER_SLAVE_TIMERx</i>	the slave timer selection(x=0..4)
<b>Input parameter{in}</b>	
<b>dmareq</b>	DMA request source
<i>SHRTIMER_MT_ST_D</i>	compare y DMA request, for Master_TIMER and Slave_TIMER (y=0..3)

<i>MA_CMPy(y=0..3)</i>	
<i>SHRTIMER_MT_ST_DMA_REP</i>	repetition DMA request, for Master_TIMER and Slave_TIMER
<i>SHRTIMER_MT_DMA_SYNID</i>	synchronization input DMA request, for Master_TIMER
<i>SHRTIMER_MT_ST_DMA_UPD</i>	update DMA request, for Master_TIMER and Slave_TIMER
<i>SHRTIMER_ST_DMA_CAPy(y=0,1)</i>	capture y DMA request, for Slave_TIMER(y=0,1)
<i>SHRTIMER_ST_DMA_CHyOA(y=0,1)</i>	channel y output active DMA request, for Slave_TIMER(y=0,1)
<i>SHRTIMER_ST_DMA_CHyONA(y=0,1)</i>	channel y output inactive DMA request, for Slave_TIMER(y=0,1)
<i>SHRTIMER_ST_DMA_CNTRST</i>	counter reset DMA request, for Slave_TIMER
<i>SHRTIMER_ST_DMA_DLYIDLE</i>	delayed IDLE mode entry DMA request, for Slave_TIMER
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the Master_TIMER and Slave_TIMER DMA request */
```

```
shrtimer_timers_dma_disable(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_DMA_CMP0);
```

### shrtimer\_dmamode\_config

The description of shrtimer\_dmamode\_config is shown as below:

**Table 3-542. Function shrtimer\_dmamode\_config**

<b>Function name</b>	shrtimer_dmamode_config
<b>Function prototype</b>	void shrtimer_dmamode_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t regupdate);
<b>Function descriptions</b>	configure the DMA mode for Master_TIMER or Slave_TIMER
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	master timer and slave timer index

SHRTIMER_MASTER_TIMER	the master timer
SHRTIMER_SLAVE_TIMERx	the slave timer selection(x=0..4)
<b>Input parameter{in}</b>	
regupdate	registers to be updated
SHRTIMER_DMAMOD_E_NONE	No register is updated by DMA mode, for Master_TIMER and Slave_TIMER
SHRTIMER_DMAMOD_E_CTL0	MTCTL0 or STxCTL0 register is updated by DMA mode, for Master_TIMER and Slave_TIMER
SHRTIMER_DMAMOD_E_INTC	MT or STx register is updated by DMA mode, for Master_TIMER and Slave_TIMER
SHRTIMER_DMAMOD_E_DMAINTEN	MTINTC or STxINTC register is updated by DMA mode, for Master_TIMER and Slave_TIMER
SHRTIMER_DMAMOD_E_CNT	MTCNT or STxCNT register is updated by DMA mode, for Master_TIMER and Slave_TIMER
SHRTIMER_DMAMOD_E_CAR	MTCAR or STxCAR register is updated by DMA mode, for Master_TIMER and Slave_TIMER
SHRTIMER_DMAMOD_E_CREP	MTCREP or STxCREP register is updated by DMA mode, for Master_TIMER and Slave_TIMER
SHRTIMER_DMAMOD_E_CMPyV(y=0..3)	MTCMPyV or STxCMPyV register is updated by DMA mode, for Master_TIMER and Slave_TIMER(y=0..3)
SHRTIMER_DMAMOD_E_DTCTL	STxDTCTL register is updated by DMA mode, only for Slave_TIMER
SHRTIMER_DMAMOD_E_CHySET(y=0,1)	STxCHySET register is updated by DMA mode, only for Slave_TIMER(y=0,1)
SHRTIMER_DMAMOD_E_CHyRST(y=0,1)	STxCHyRST register is updated by DMA mode, only for Slave_TIMER(y=0,1)
SHRTIMER_DMAMOD_E_EXEVFCFGy(y=0,1)	STxEXEVFCFGy register is updated by DMA mode, only for Slave_TIMER(y=0,1)
SHRTIMER_DMAMOD_E_CNTRST	STxCNTRST register is updated by DMA mode, only for Slave_TIMER
SHRTIMER_DMAMOD_E_CSCTL	STxCSCCTL register is updated by DMA mode, only for Slave_TIMER
SHRTIMER_DMAMOD_E_CHOCTL	STxCHOCTL register is updated by DMA mode, only for Slave_TIMER
SHRTIMER_DMAMOD_E_FLTCTL	STxFLTCTL register is updated by DMA mode, only for Slave_TIMER
SHRTIMER_DMAMOD_E_ACTL	STxACTL register is updated by DMA mode, only for Slave_TIMER
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* enable the Master_TIMER and Slave_TIMER DMA request */
```

```
shrtimer_dmamode_config(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_DMAMODE_NONE);
```

### shrtimer\_bunchmode\_struct\_para\_init

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

**Table 3-543. Function shrtimer\_bunchmode\_struct\_para\_init**

Function name	shrtimer_bunchmode_struct_para_init
Function prototype	void shrtimer_bunchmode_struct_para_init(shrtimer_bunchmode_parameter_struct* bmcfg);
Function descriptions	initialize bunch mode configuration struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
bmcfg	shrtimer_bunchmode_parameter_struct, the structure members can refer to <a href="#">Table 3-486. Structure shrtimer_bunchmode_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize bunch mode configuration struct with a default value */
```

```
shrtimer_bunchmode_parameter_struct bmcfg;
```

```
shrtimer_bunchmode_struct_para_init(&bmcfg);
```

### shrtimer\_bunchmode\_config

The description of shrtimer\_bunchmode\_config is shown as below:

**Table 3-544. Function shrtimer\_bunchmode\_config**

Function name	shrtimer_bunchmode_config
Function prototype	void shrtimer_bunchmode_config(uint32_t shrtimer_periph, shrtimer_bunchmode_parameter_struct* bmcfg);
Function descriptions	configure bunch mode for the SHRTIMER
Precondition	-
The called functions	-

Input parameter{in}	
<b>bmcfg</b>	shrtimer_bunchmode_parameter_struct, the structure members can refer to <a href="#">Table 3-486. Structure shrtimer_bunchmode_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure bunch mode for the SHRTIMER */

shrtimer_bunchmode_parameter_struct bmcfg;

shrtimer_bunchmode_struct_para_init(&bmcfg);

bmcfg.clock_source = SHRTIMER_BUNCHMODE_CLOCKSOURCE_ST0;

bmcfg.idle_duration = 3;

bmcfg.mode = SHRTIMER_BUNCHMODE_CONTINUOUS;

bmcfg.period = 6;

bmcfg.prescaler = SHRTIMER_BUNCHMODE_PRESCALER_DIV1;

bmcfg.shadow = SHRTIMER_BUNCHMODEPRELOAD_DISABLED;

bmcfg.trigger = SHRTIMER_BUNCHMODE_TRIGGER_SOFTWARE;

shrtimer_bunchmode_config(SHRTIMER0, &bmcfg);

```

## shrtimer\_bunchmode\_enable

The description of shrtimer\_bunchmode\_enable is shown as below:

**Table 3-545. Function shrtimer\_bunchmode\_enable**

Function name	shrtimer_bunchmode_enable
Function prototype	void shrtimer_bunchmode_enable(uint32_t shrtimer_periph);
Function descriptions	enable bunch mode for the SHRTIMER
Precondition	-
The called functions	-
Input parameter{in}	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable bunch mode for the SHRTIMER */
```

```
shrtimer_bunchmode_enable(SHRTIMER0);
```

## shrtimer\_bunchmode\_disable

The description of shrtimer\_bunchmode\_disable is shown as below:

**Table 3-546. Function shrtimer\_bunchmode\_disable**

<b>Function name</b>	shrtimer_bunchmode_disable
<b>Function prototype</b>	void shrtimer_bunchmode_disable(uint32_t shrtimer_periph);
<b>Function descriptions</b>	disable bunch mode for the SHRTIMER
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable bunch mode for the SHRTIMER */
```

```
shrtimer_bunchmode_disable(SHRTIMER0);
```

## shrtimer\_bunchmode\_flag\_get

The description of shrtimer\_bunchmode\_flag\_get is shown as below:

**Table 3-547. Function shrtimer\_bunchmode\_flag\_get**

<b>Function name</b>	shrtimer_bunchmode_flag_get
<b>Function prototype</b>	uint32_t shrtimer_bunchmode_flag_get(uint32_t shrtimer_periph);
<b>Function descriptions</b>	get bunch mode operating flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	SHRTIMER_BUNCHMODE_OPERATION_OFF or SHRTIMER_BUNCHMODE_OPERATION_ON

Example:

```
/* get bunch mode operating flag */
```

```
uint32_t flag;
```

```
flag = shrTIMER_bunchmode_flag_get(SHRTIMER0);
```

### shrTIMER\_bunchmode\_software\_start

The description of shrTIMER\_bunchmode\_software\_start is shown as below:

**Table 3-548. Function shrTIMER\_bunchmode\_software\_start**

<b>Function name</b>	shrTIMER_bunchmode_software_start
<b>Function prototype</b>	void shrTIMER_bunchmode_software_start(uint32_t shrTIMER_periph);
<b>Function descriptions</b>	bunch mode started by software
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrTIMER_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* bunch mode started by software */
```

```
shrTIMER_bunchmode_software_start(SHRTIMER0);
```

### shrTIMER\_slavetimer\_capture\_config

The description of shrTIMER\_slavetimer\_capture\_config is shown as below:

**Table 3-549. Function shrTIMER\_slavetimer\_capture\_config**

<b>Function name</b>	shrTIMER_slavetimer_capture_config
<b>Function prototype</b>	void shrTIMER_slavetimer_capture_config(uint32_t shrTIMER_periph, uint32_t timer_id, uint32_t capturex, uint32_t trgsource);
<b>Function descriptions</b>	configure the capture source in Slave_TIMER
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrTIMER_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	master timer and slave timer index



SHRTIMER_SLAVE_TIMERx	the slave timer selection(x=0..4)
<b>Input parameter{in}</b>	
capturex	capture unit index
SHRTIMER_CAPTURE_y(y=0,1)	capture unit selection(y=0,1)
<b>Input parameter{in}</b>	
trgsource	capture trigger source
SHRTIMER_CAPTURE_TRIGGER_NONE	Capture trigger is disabled
SHRTIMER_CAPTURE_TRIGGER_UPDATE	capture triggered by update event
SHRTIMER_CAPTURE_TRIGGER_EXEV_y(y=0..9)	capture triggered by external event 0(y=0..9)
SHRTIMER_CAPTURE_TRIGGER_STy_ACTIVE(y=0..4)	capture triggered by STyCH0_O output inactive to active transition(y=0..4)
SHRTIMER_CAPTURE_TRIGGER_STy_INACTIVE(y=0..4)	capture triggered by STyCH0_O output active to inactive transition(y=0..4)
SHRTIMER_CAPTURE_TRIGGER_STy_CMP0(y=0..4)	capture triggered by compare 0 event of Slave_TIMERy(y=0..4)
SHRTIMER_CAPTURE_TRIGGER_STy_CMP1(y=0..4)	capture triggered by compare 1 event of Slave_TIMERy(y=0..4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the capture source in Slave_TIMER */
```

```
shrtimer_slavetimer_capture_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,
SHRTIMER_CAPTURE_0, SHRTIMER_CAPTURE_TRIGGER_NONE);
```

### shrtimer\_slavetimer\_capture\_software

The description of shrtimer\_slavetimer\_capture\_software is shown as below:

**Table 3-550. Function shrtimer\_slavetimer\_capture\_software**

<b>Function name</b>	shrtimer_slavetimer_capture_software
----------------------	--------------------------------------

<b>Function prototype</b>	void shrtimer_slavetimer_capture_software(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t capturex);
<b>Function descriptions</b>	configure the capture source in Slave_TIMER
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	master timer and slave timer index
<i>SHRTIMER_SLAVE_TIMERx</i>	the slave timer selection(x=0..4)
<b>Input parameter{in}</b>	
<b>capturex</b>	capture unit index
<i>SHRTIMER_CAPTURE_y(y=0,1)</i>	capture unit selection(y=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the capture source in Slave_TIMER */
```

```
shrtimer_slavetimer_capture_software(SHRTIMER0, SHRTIMER_SLAVE  
SHRTIMER_CAPTURE_0);
```

## shrtimer\_slavetimer\_capture\_value\_read

The description of shrtimer\_slavetimer\_capture\_value\_read is shown as below:

**Table 3-551. Function shrtimer\_slavetimer\_capture\_value\_read**

<b>Function name</b>	shrtimer_slavetimer_capture_value_read
<b>Function prototype</b>	uint32_t shrtimer_slavetimer_capture_value_read(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t capturex);
<b>Function descriptions</b>	read the capture value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	master timer and slave timer index
<i>SHRTIMER_SLAVE_TIMERx</i>	the slave timer selection(x=0..4)

<i>MERx</i>	
<b>Input parameter{in}</b>	
<b>capturex</b>	capture unit index
<i>SHRTIMER_CAPTURE</i> <i>_y(y=0,1)</i>	capture unit selection(y=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	capture value

Example:

```
/* read the capture value */
```

```
uint32_t capture_value;
```

```
capture_value = shrtimer_slavetimer_capture_value_read (SHRTIMER0,
SHRTIMER_SLAVE SHRTIMER_CAPTURE_0);
```

### shrtimer\_adctrigcfg\_struct\_para\_init

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

**Table 3-552. Function shrtimer\_adctrigcfg\_struct\_para\_init**

<b>Function name</b>	shrtimer_adctrigcfg_struct_para_init
<b>Function prototype</b>	void shrtimer_adctrigcfg_struct_para_init(shrtimer_adctrigcfg_parameter_struct* triggercfg);
<b>Function descriptions</b>	initialize ADC trigger configuration struct with a default value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>triggercfg</b>	shrtimer_adctrigcfg_parameter_struct, the structure members can refer to <a href="#">Table 3-489. Structure shrtimer_adctrigcfg_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize ADC trigger configuration struct with a default value */
```

```
shrtimer_adctrigcfg_parameter_struct triggercfg;
```

```
shrtimer_adctrigcfg_struct_para_init(&triggercfg);
```

## shrtimer\_adc\_trigger\_config

The description of shrtimer\_adc\_trigger\_config is shown as below:

**Table 3-553. Function shrtimer\_adc\_trigger\_config**

<b>Function name</b>	shrtimer_adc_trigger_config
<b>Function prototype</b>	void shrtimer_adc_trigger_config(uint32_t shrtimer_periph, uint32_t trigger_id, shrtimer_adctrigcfg_parameter_struct* triggercfg);
<b>Function descriptions</b>	configure the trigger source to ADC and the update source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>trigger_id</b>	ADC trigger event
SHRTIMER_ADCTRIG_y(y=0..3)	the slave timer selection(x=0..4)
<b>Input parameter{in}</b>	
<b>triggercfg</b>	shrtimer_adctrigcfg_parameter_struct, the structure members can refer to <a href="#">Table 3-489. Structure shrtimer_adctrigcfg_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure the trigger source to ADC and the update source */

shrtimer_adctrigcfg_parameter_struct triggercfg;

shrtimer_adctrigcfg_struct_para_init(&triggercfg);

triggercfg.update_source = SHRTIMER_ADCTRG_I_UPDATE_MT;

triggercfg.trigger = SHRTIMER_ADCTRG_I02_EVENT_NONE;

shrtimer_adc_trigger_config(SHRTIMER0, SHRTIMER_ADCTRIG_0, &triggercfg);

```

## shrtimer\_timers\_flag\_get

The description of shrtimer\_timers\_flag\_get is shown as below:

**Table 3-554. Function shrtimer\_timers\_flag\_get**

<b>Function name</b>	shrtimer_timers_flag_get
<b>Function prototype</b>	FlagStatus shrtimer_timers_flag_get(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t flag);

<b>Function descriptions</b>	get the Master_TIMER and Slave_TIMER flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<b>SHRTIMERx(x=0)</b>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	master timer and slave timer index
<b>SHRTIMER_MASTER_TIMER</b>	the master timer
<b>SHRTIMER_SLAVE_TIMERx</b>	the slave timer selection(x=0..4)
<b>Input parameter{in}</b>	
<b>flag</b>	the flag source
<b>SHRTIMER_MT_ST_FLAG_CMPy(y=0..3)</b>	compare y flag, for Master_TIMER and Slave_TIMER (y=0..3)
<b>SHRTIMER_MT_ST_FLAG_LAG_REP</b>	repetition flag, for Master_TIMER and Slave_TIMER
<b>SHRTIMER_MT_ST_FLAG_LAG_SYN1</b>	synchronization input flag, for Master_TIMER
<b>SHRTIMER_MT_ST_FLAG_LAG_UPD</b>	update flag, for Master_TIMER and Slave_TIMER
<b>SHRTIMER_ST_FLAG_CAPy(y=0,1)</b>	capture y flag, for Slave_TIMER(y=0,1)
<b>SHRTIMER_ST_FLAG_CHyOA(y=0,1)</b>	channel y output active flag, for Slave_TIMER(y=0,1)
<b>SHRTIMER_ST_FLAG_CHyONA(y=0,1)</b>	channel y output inactive flag, for Slave_TIMER(y=0,1)
<b>SHRTIMER_ST_FLAG_CNTRST</b>	counter reset flag, for Slave_TIMER
<b>SHRTIMER_ST_FLAG_DLYIDLE</b>	delayed IDLE mode entry flag, for Slave_TIMER
<b>SHRTIMER_ST_FLAG_CBLN</b>	current balanced flag, for Slave_TIMER
<b>SHRTIMER_ST_FLAG_BLNIDLE</b>	balanced IDLE flag, for Slave_TIMER
<b>SHRTIMER_ST_FLAG_CHyOUT(y=0,1)</b>	channel y output flag, for Slave_TIMER(y=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the Master_TIMER and Slave_TIMER flag */
```

```
FlagStatus flag = RESET;
```

```
flag = shrtimer_timers_flag_get(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_FLAG_CMP0);
```

### shrtimer\_timers\_flag\_clear

The description of shrtimer\_timers\_flag\_clear is shown as below:

**Table 3-555. Function shrtimer\_timers\_flag\_clear**

Function name	shrtimer_timers_flag_clear
Function prototype	void shrtimer_timers_flag_clear(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t flag);
Function descriptions	clear the Master_TIMER and Slave_TIMER flag
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	master timer and slave timer index
SHRTIMER_MASTER_TIMER	the master timer
SHRTIMER_SLAVE_TIMERx	the slave timer selection(x=0..4)
Input parameter{in}	
flag	the flag source
SHRTIMER_MT_ST_FLAG_CMPy(y=0..3)	compare y flag, for Master_TIMER and Slave_TIMER (y=0..3)
SHRTIMER_MT_ST_FLAG_REP	repetition flag, for Master_TIMER and Slave_TIMER
SHRTIMER_MT_INT_FLAG_SYNCI	synchronization input flag, for Master_TIMER
SHRTIMER_MT_ST_FLAG_UPD	update flag, for Master_TIMER and Slave_TIMER
SHRTIMER_ST_FLAG_CAPy(y=0,1)	capture y flag, for Slave_TIMER(y=0,1)
SHRTIMER_ST_FLAG_CHyOA(y=0,1)	channel y output active flag, for Slave_TIMER(y=0,1)
SHRTIMER_ST_FLAG_CHyONA(y=0,1)	channel y output inactive flag, for Slave_TIMER(y=0,1)

<i>SHRTIMER_ST_FLAG_CNTRST</i>	counter reset flag, for Slave_TIMER
<i>SHRTIMER_ST_FLAG_DLYIDLE</i>	delayed IDLE mode entry flag, for Slave_TIMER
<i>SHRTIMER_ST_FLAG_CBLN</i>	current balanced flag, for Slave_TIMER
<i>SHRTIMER_ST_FLAG_BLNIDLE</i>	balanced IDLE flag, for Slave_TIMER
<i>SHRTIMER_ST_FLAG_CHyOUT(y=0,1)</i>	channel y output flag, for Slave_TIMER(y=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the Master_TIMER and Slave_TIMER flag */
shrtimer_timers_flag_clear(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_FLAG_CMP0);
```

### shrtimer\_common\_flag\_get

The description of shrtimer\_common\_flag\_get is shown as below:

**Table 3-556. Function shrtimer\_common\_flag\_get**

<b>Function name</b>	shrtimer_common_flag_get
<b>Function prototype</b>	FlagStatus shrtimer_common_flag_get(uint32_t shrtimer_periph, uint32_t flag);
<b>Function descriptions</b>	get the common flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>flag</b>	the flag source
<i>SHRTIMER_FLAG_FLTy(y=0..3)</i>	fault y interrupt flag (y=0..3)
<i>SHRTIMER_FLAG_SY SFLT</i>	system fault interrupt flag
<i>SHRTIMER_FLAG_DL LCAL</i>	DLL calibration completed interrupt flag
<i>SHRTIMER_FLAG_BM</i>	bunch mode period interrupt flag

<i>PER</i>	
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the common flag */
```

```
FlagStatus flag = RESET;
```

```
flag = shrtimer_common_flag_get(SHRTIMER0, SHRTIMER_FLAG_FLT0);
```

### shrtimer\_common\_flag\_clear

The description of shrtimer\_common\_flag\_clear is shown as below:

**Table 3-557. Function shrtimer\_common\_flag\_clear**

<b>Function name</b>	shrtimer_common_flag_clear
<b>Function prototype</b>	void shrtimer_common_flag_clear(uint32_t shrtimer_periph, uint32_t flag);
<b>Function descriptions</b>	clear the common flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>flag</b>	the flag source
<i>SHRTIMER_FLAG_FLT</i> <i>y(y=0..3)</i>	fault y interrupt flag (y=0..3)
<i>SHRTIMER_FLAG_SY</i> <i>SFLT</i>	system fault interrupt flag
<i>SHRTIMER_FLAG_DL</i> <i>LCAL</i>	DLL calibration completed interrupt flag
<i>SHRTIMER_FLAG_BM</i> <i>PER</i>	bunch mode period interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the common flag */
```

```
shrtimer_common_flag_clear(SHRTIMER0, SHRTIMER_FLAG_FLT0);
```



## shrtimer\_timers\_interrupt\_enable

The description of shrtimer\_timers\_interrupt\_enable is shown as below:

**Table 3-558. Function shrtimer\_timers\_interrupt\_enable**

<b>Function name</b>	shrtimer_timers_interrupt_enable
<b>Function prototype</b>	void shrtimer_timers_interrupt_enable(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t interrupt);
<b>Function descriptions</b>	enable the Master_TIMER and Slave_TIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	master timer and slave timer index
SHRTIMER_MASTER_TIMER	the master timer
SHRTIMER_SLAVE_TIMERx	the slave timer selection(x=0..4)
<b>Input parameter{in}</b>	
<b>interrupt</b>	the interrupt source
SHRTIMER_MT_ST_INT_CMPy(y=0..3)	compare y interrupt, for Master_TIMER and Slave_TIMER (y=0..3)
SHRTIMER_MT_ST_INT_T_REP	repetition interrupt, for Master_TIMER and Slave_TIMER
SHRTIMER_MT_INT_SYNCI	synchronization input interrupt, for Master_TIMER
SHRTIMER_MT_ST_INT_T_UPD	update interrupt, for Master_TIMER and Slave_TIMER
SHRTIMER_ST_INT_CAPy(y=0,1)	capture y interrupt, for Slave_TIMER(y=0,1)
SHRTIMER_ST_INT_CHyOA(y=0,1)	channel y output active interrupt, for Slave_TIMER(y=0,1)
SHRTIMER_ST_INT_CHyONA(y=0,1)	channel y output inactive interrupt, for Slave_TIMER(y=0,1)
SHRTIMER_ST_INT_CNTRST	counter reset interrupt, for Slave_TIMER
SHRTIMER_ST_INT_DLYIDLE	delayed IDLE mode entry interrupt, for Slave_TIMER
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* enable the Master_TIMER and Slave_TIMER interrupt */
```

```
shrtimer_timers_interrupt_enable(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_INT_CMP0);
```

## shrtimer\_timers\_interrupt\_disable

The description of shrtimer\_timers\_interrupt\_disable is shown as below:

**Table 3-559. Function shrtimer\_timers\_interrupt\_disable**

<b>Function name</b>	shrtimer_timers_interrupt_disable
<b>Function prototype</b>	void shrtimer_timers_interrupt_disable(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t interrupt);
<b>Function descriptions</b>	disable the Master_TIMER and Slave_TIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	master timer and slave timer index
SHRTIMER_MASTER_TIMER	the master timer
SHRTIMER_SLAVE_TIMERx	the slave timer selection(x=0..4)
<b>Input parameter{in}</b>	
<b>interrupt</b>	the interrupt source
SHRTIMER_MT_ST_INT_CMPy(y=0..3)	compare y interrupt, for Master_TIMER and Slave_TIMER (y=0..3)
SHRTIMER_MT_ST_INT_T_REP	repetition interrupt, for Master_TIMER and Slave_TIMER
SHRTIMER_MT_INT_SYNCI	synchronization input interrupt, for Master_TIMER
SHRTIMER_MT_ST_INT_T_UPD	update interrupt, for Master_TIMER and Slave_TIMER
SHRTIMER_ST_INT_CAPTUREy(y=0,1)	capture y interrupt, for Slave_TIMER(y=0,1)
SHRTIMER_ST_INT_CHANNELyOA(y=0,1)	channel y output active interrupt, for Slave_TIMER(y=0,1)
SHRTIMER_ST_INT_CHANNELyONA(y=0,1)	channel y output inactive interrupt, for Slave_TIMER(y=0,1)

<i>SHRTIMER_ST_INT_CNTRST</i>	counter reset interrupt, for Slave_TIMER
<i>SHRTIMER_ST_INT_DLYIDLE</i>	delayed IDLE mode entry interrupt, for Slave_TIMER
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the Master_TIMER and Slave_TIMER interrupt */
```

```
shrtimer_timers_interrupt_disable(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_INT_CMP0);
```

### shrtimer\_timers\_interrupt\_flag\_get

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

**Table 3-560. Function shrtimer\_timers\_interrupt\_flag\_get**

<b>Function name</b>	shrtimer_timers_interrupt_flag_get
<b>Function prototype</b>	FlagStatus shrtimer_timers_interrupt_flag_get(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t interrupt);
<b>Function descriptions</b>	get the Master_TIMER and Slave_TIMER interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	master timer and slave timer index
<i>SHRTIMER_MASTER_TIMER</i>	the master timer
<i>SHRTIMER_SLAVE_TIMERx</i>	the slave timer selection(x=0..4)
<b>Input parameter{in}</b>	
<b>interrupt</b>	the interrupt source
<i>SHRTIMER_MT_ST_INT_FLAG_CMPy(y=0..3)</i>	compare y interrupt flag, for Master_TIMER and Slave_TIMER (y=0..3)
<i>SHRTIMER_MT_ST_INT_FLAG_REP</i>	repetition interrupt flag, for Master_TIMER and Slave_TIMER
<i>SHRTIMER_MT_INT_FLAG_SYNC</i>	synchronization input interrupt flag, for Master_TIMER

<i>SHRTIMER_MT_ST_INT_FLAG_UPD</i>	update interrupt flag, for Master_TIMER and Slave_TIMER
<i>SHRTIMER_ST_INT_FLAG_CAPy(y=0,1)</i>	capture y interrupt flag, for Slave_TIMER(y=0,1)
<i>SHRTIMER_ST_INT_FLAG_CHyOA(y=0,1)</i>	channel y output active interrupt flag, for Slave_TIMER(y=0,1)
<i>SHRTIMER_ST_INT_FLAG_CHyONA(y=0,1)</i>	channel y output inactive interrupt flag, for Slave_TIMER(y=0,1)
<i>SHRTIMER_ST_INT_FLAG_CNTRST</i>	counter reset interrupt flag, for Slave_TIMER
<i>SHRTIMER_ST_INT_FLAG_DLYIDLE</i>	delayed IDLE mode entry interrupt flag, for Slave_TIMER
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the Master_TIMER and Slave_TIMER interrupt flag */
```

```
FlagStatus flag = RESET;
```

```
flag = shrtimer_timers_interrupt_flag_get(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_INT_FLAG_CMP0);
```

### shrtimer\_timers\_interrupt\_flag\_clear

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

**Table 3-561. Function shrtimer\_timers\_interrupt\_flag\_clear**

<b>Function name</b>	shrtimer_timers_interrupt_flag_clear
<b>Function prototype</b>	void shrtimer_timers_interrupt_flag_clear(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t interrupt);
<b>Function descriptions</b>	clear the Master_TIMER and Slave_TIMER interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>timer_id</b>	master timer and slave timer index
<i>SHRTIMER_MASTER_TIMER</i>	the master timer
<i>SHRTIMER_SLAVE_TIMERx</i>	the slave timer selection(x=0..4)

Input parameter{in}	
<b>interrupt</b>	the interrupt source
<code>SHRTIMER_MT_ST_IN T _FLAG_CMPy(y=0..3)</code>	compare y interrupt flag, for Master_TIMER and Slave_TIMER (y=0..3)
<code>SHRTIMER_MT_ST_IN T_FLAG_REP</code>	repetition interrupt flag, for Master_TIMER and Slave_TIMER
<code>SHRTIMER_MT_INT_F LAG_SYNI</code>	synchronization input interrupt flag, for Master_TIMER
<code>SHRTIMER_MT_ST_IN T_FLAG_UPD</code>	update interrupt flag, for Master_TIMER and Slave_TIMER
<code>SHRTIMER_ST_INT_F LAG_CAPy(y=0,1)</code>	capture y interrupt flag, for Slave_TIMER(y=0,1)
<code>SHRTIMER_ST_INT_F LAG_CHyOA(y=0,1)</code>	channel y output active interrupt flag, for Slave_TIMER(y=0,1)
<code>SHRTIMER_ST_INT_F LAG_CHyONA(y=0,1)</code>	channel y output inactive interrupt flag, for Slave_TIMER(y=0,1)
<code>SHRTIMER_ST_INT_F LAG_CNTRST</code>	counter reset interrupt flag, for Slave_TIMER
<code>SHRTIMER_ST_INT_F LAG_DLYIDLE</code>	delayed IDLE mode entry interrupt flag, for Slave_TIMER
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the Master_TIMER and Slave_TIMER interrupt flag */
```

```
shrtimer_timers_interrupt_flag_clear(SHRTIMER0, SHRTIMER_MASTER_TIMER,  
SHRTIMER_MT_ST_INT_FLAG_CMP0);
```

## shrtimer\_common\_interrupt\_enable

The description of shrtimer\_common\_interrupt\_enable is shown as below:

**Table 3-562. Function shrtimer\_common\_interrupt\_enable**

<b>Function name</b>	shrtimer_common_interrupt_enable
<b>Function prototype</b>	void shrtimer_common_interrupt_enable(uint32_t shrtimer_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable SHRTIMER common interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	

<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>interrupt</b>	the interrupt source
<i>SHRTIMER_INT_FLTy(y=0..3)</i>	fault y interrupt (y=0..3)
<i>SHRTIMER_INT_SYSLT</i>	system fault interrupt
<i>SHRTIMER_INT_DLLCAL</i>	DLL calibration completed interrupt
<i>SHRTIMER_INT_BMPER</i>	bunch mode period interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SHRTIMER common interrupt */
```

```
shrtimer_common_interrupt_enable(SHRTIMER0, SHRTIMER_INT_FLT0);
```

## shrtimer\_common\_interrupt\_disable

The description of shrtimer\_common\_interrupt\_disable is shown as below:

**Table 3-563. Function shrtimer\_common\_interrupt\_disable**

<b>Function name</b>	shrtimer_common_interrupt_disable
<b>Function prototype</b>	void shrtimer_common_interrupt_disable(uint32_t shrtimer_periph, uint32_t interrupt);
<b>Function descriptions</b>	disable SHRTIMER common interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>interrupt</b>	the interrupt source
<i>SHRTIMER_INT_FLTy(y=0..3)</i>	fault y interrupt (y=0..3)
<i>SHRTIMER_INT_SYSLT</i>	system fault interrupt
<i>SHRTIMER_INT_DLLCAL</i>	DLL calibration completed interrupt

<i>SHRTIMER_INT_BMP</i> <i>ER</i>	bunch mode period interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SHRTIMER common interrupt */
shrtimer_common_interrupt_disable(SHRTIMER0, SHRTIMER_INT_FLT0);
```

### shrtimer\_common\_interrupt\_flag\_get

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

**Table 3-564. Function shrtimer\_common\_interrupt\_flag\_get**

<b>Function name</b>	shrtimer_common_interrupt_flag_get
<b>Function prototype</b>	FlagStatus shrtimer_common_interrupt_flag_get(uint32_t shrtimer_periph, uint32_t interrupt);
<b>Function descriptions</b>	get the common interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx</i> (x=0)	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>interrupt</b>	the interrupt source
<i>SHRTIMER_INT_FLAG_FLTy</i> (y=0..3)	fault y interrupt flag (y=0..3)
<i>SHRTIMER_INT_FLAG_SYSFLT</i>	system fault interrupt flag
<i>SHRTIMER_INT_FLAG_DLLCAL</i>	DLL calibration completed interrupt flag
<i>SHRTIMER_INT_FLAG_BMPER</i>	bunch mode period interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the common interrupt flag */
FlagStatus flag = RESET;
```

```
flag = shrtimer_common_interrupt_flag_get(SHRTIMER0, SHRTIMER_INT_FLAG_FLT0);
```

## shrtimer\_common\_interrupt\_flag\_clear

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

**Table 3-565. Function shrtimer\_common\_interrupt\_flag\_clear**

<b>Function name</b>	shrtimer_common_interrupt_flag_clear
<b>Function prototype</b>	void shrtimer_common_interrupt_flag_clear(uint32_t shrtimer_periph, uint32_t interrupt);
<b>Function descriptions</b>	clear the common interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_periph</b>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
<b>Input parameter{in}</b>	
<b>interrupt</b>	the interrupt source
<i>SHRTIMER_INT _FLAG_FLTy(y=0..3)</i>	fault y interrupt flag (y=0..3)
<i>SHRTIMER_INT _FLAG_SYSFLT</i>	system fault interrupt flag
<i>SHRTIMER_INT _FLAG_DLLCAL</i>	DLL calibration completed interrupt flag
<i>SHRTIMER_INT _FLAG_BMPER</i>	bunch mode period interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the common interrupt flag */
```

```
shrtimer_common_interrupt_flag_clear(SHRTIMER0, SHRTIMER_INT_FLAG_FLT0);
```

## 3.18. I2C

The I2C (inter-integrated circuit) module provides an I2C interface which is an industry standard two-line serial interface for MCU to communicate with external I2C interface. The I2C registers are listed in chapter [3.18.1](#), the I2C firmware functions are introduced in chapter [3.18.2](#).



### 3.18.1. Descriptions of Peripheral registers

I2C registers are listed in the table shown as below:

**Table 3-566. I2C Registers**

Registers	Descriptions
I2C_CTL0	Control register 0
I2C_CTL1	Control register 1
I2C_SADDR0	Slave address register 0
I2C_SADDR1	Slave address register 1
I2C_DATA	Transfer buffer register
I2C_STAT0	Transfer status register 0
I2C_STAT1	Transfer status register 1
I2C_CKCFG	Clock configure register
I2C_RT	Rise time register
I2C_SAMCS	SAM control and status register
I2C_CTL2	Control register 2
I2C_CS	Control and status register
I2C_STATC	Status clear register
I2C2_CTL0	Control register 0
I2C2_CTL1	Control register 1
I2C2_SADDR0	Slave address register 0
I2C2_SADDR1	Slave address register 1
I2C2_TIMING	Timing register
I2C2_TIMEOUT	Timeout register
I2C2_STAT	Status register
I2C2_STATC	Status clear register
I2C2_PEC	PEC register
I2C2_RDATA	Receive data register
I2C2_TDATA	Transmit data register

### 3.18.2. Descriptions of Peripheral functions

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

**Table 3-567. I2C firmware function**

Function name	Function description
i2c_deinit	reset I2C
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_slave_response_to_gcall_enable	enable the response to a general call

Function name	Function description
i2c_slave_response_to_gcall_disable	disable the response to a general call
i2c_stretch_scl_low_enable	enable to stretch SCL low when data is not ready in slave mode
i2c_stretch_scl_low_disable	disable to stretch SCL low when data is not ready in slave mode
i2c_data_transmit	I2C transmit data function
i2c_data_receive	I2C receive data function
i2c_pec_transfer	I2C transfers PEC value
i2c_pec_enable	enable I2C PEC calculation
i2c_pec_disable	disable I2C PEC calculation
i2c_pec_value_get	get packet error checking value
i2c_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	dual-address mode switch
i2c_dualaddr_disable	disable dual-address mode
i2c_dma_config	configure I2C DMA mode
i2c_dma_last_transfer_config	configure whether next DMA EOT is DMA last transfer or not
i2c_software_reset_config	configure software reset I2C
i2c_smbus_alert_config	configure I2C alert through SMBA pin
i2c_smbus_arp_config	configure I2C ARP protocol in SMBus
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_start_early_termination_mode_config	configure I2C start early termination mode
i2c_timeout_calculation_enable	enable i2c timeout calculation
i2c_timeout_calculation_disable	disable i2c timeout calculation
i2c_record_received_slave_address_enable	enable i2c record the received slave address to the transfer buffer register
i2c_record_received_slave_address_disable	disable i2c record the received slave address to the transfer buffer register
i2c_address_bit_compare_config	define which bits of ADDRESS[7:1] need to compare with the incoming address byte
i2c_status_clear_enable	enable i2c status register clear
i2c_status_clear_disable	disable i2c status register clear
i2c_status_bit_clear	clear i2c status register bit

Function name	Function description
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
i2c_interrupt_flag_clear	clear I2C interrupt flag
i2c_timing_config	configure the timing parameters
i2c_digital_noise_filter_config	configure digital noise filter
i2c_analog_noise_filter_enable	enable analog noise filter
i2c_analog_noise_filter_disable	disable analog noise filter
i2c_wakeup_from_deepsleep_enable	enable wakeup from Deep-sleep mode
i2c_wakeup_from_deepsleep_disable	disable wakeup from Deep-sleep mode
i2c_master_clock_config	configure the SCL high and low period of clock in master mode
i2c2_master_addressing	configure i2c slave addresss and transfer direction in master mode
i2c_address10_header_enable	10-bit address header executes read direction only in master receive mode
i2c_address10_header_disable	10-bit address header executes complete sequence in master receive mode
i2c_address10_enable	enable 10-bit addressing mode in master mode
i2c_address10_disable	disable 10-bit addressing mode in master mode
i2c_automatic_end_enable	enable I2C automatic end mode in master mode
i2c_automatic_end_disable	disable I2C automatic end mode in master mode
i2c_address_config	configure i2c slave address
i2c_address_disable	disable i2c address in slave mode
i2c_second_address_config	configure i2c second slave address
i2c_second_address_disable	disable i2c second address in slave mode
i2c_receivied_address_get	get received match address in slave mode
i2c_slave_byte_control_enable	enable slave byte control
i2c_slave_byte_control_disable	disable slave byte control
i2c_nack_enable	generate a NACK in slave mode
i2c_nack_disable	generate an ACK in slave mode
i2c_reload_enable	enable I2C reload mode
i2c_reload_disable	disable I2C reload mode
i2c_transfer_byte_number_config	configure number of bytes to be transferred
i2c2_dma_enable	enable I2C DMA for transmission or reception
i2c2_dma_disable	disable I2C DMA for transmission or reception
i2c_smbus_alert_enable	enable SMBus Alert
i2c_smbus_alert_disable	disable SMBus Alert
i2c_smbus_default_addr_enable	enable SMBus device default address

Function name	Function description
i2c_smbus_default_addr_disable	disable SMBus device default address
i2c_smbus_host_addr_enable	enable SMBus Host address
i2c_smbus_host_addr_disable	disable SMBus Host address
i2c_extented_clock_timeout_enable	enable extended clock timeout detection
i2c_extented_clock_timeout_disable	disable extended clock timeout detection
i2c_clock_timeout_enable	enable clock timeout detection
i2c_clock_timeout_disable	disable clock timeout detection
i2c_bus_timeout_b_config	configure bus timeout B
i2c_bus_timeout_a_config	configure bus timeout A
i2c_idle_clock_timeout_config	configure idle clock timeout detection
i2c2_flag_get	get I2C flag status
i2c2_flag_clear	clear I2C flag status
i2c2_interrupt_enable	enable I2C interrupt
i2c2_interrupt_disable	disable I2C interrupt
i2c2_interrupt_flag_get	get I2C interrupt flag
i2c2_interrupt_flag_clear	clear I2C interrupt flag

### Enum i2c\_flag\_enum

Table 3-568. i2c\_flag\_enum

Member name	Function description
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
I2C_FLAG_LOSTARB	arbitration lost in master mode
I2C_FLAG_AERR	acknowledge error
I2C_FLAG_OUERR	overflow or underrun situation occurs in slave mode
I2C_FLAG_PECERR	PEC error when receiving data
I2C_FLAG_SMBTO	timeout signal in SMBus mode
I2C_FLAG_SMBALT	SMBus alert status
I2C_FLAG_MASTER	a flag indicating whether I2C block is in master or slave mode
I2C_FLAG_I2CBSY	busy flag
I2C_FLAG_TR	whether the I2C is a transmitter or a receiver
I2C_FLAG_RXGC	general call address (00h) received
I2C_FLAG_DEFSMB	default address of SMBus device

Member name	Function description
I2C_FLAG_HSTSMB	SMBus host header detected in slave mode
I2C_FLAG_DUMOD	dual flag in slave mode indicating which address is matched in dual-address mode
I2C_FLAG_TFF	txframe fall flag
I2C_FLAG_TFR	txframe rise flag
I2C_FLAG_RFF	rxframe fall flag
I2C_FLAG_RFR	rxframe rise flag
I2C_FLAG_STLO	start lost flag
I2C_FLAG_STPSEND	stop condition sent flag

## Enum i2c\_interrupt\_enum

Table 3-569. i2c\_interrupt\_enum

Member name	Function description
I2C_INT_ERR	error interrupt disable
I2C_INT_EV	event interrupt disable
I2C_INT_BUF	buffer interrupt disable
I2C_INT_TFF	txframe fall interrupt enable
I2C_INT_TFR	txframe rise interrupt enable
I2C_INT_RFF	rxframe fall interrupt enable
I2C_INT_RFR	rxframe rise interrupt enable
I2C_INT_STLO	start lost interrupt enable
I2C_INT_STPSEND	stop condition sent interrupt enable

## Enum i2c\_interrupt\_flag\_enum

Table 3-570. i2c\_interrupt\_flag\_enum

Member name	Function description
I2C_INT_FLAG_SBSSEND	start condition sent out in master mode interrupt flag
I2C_INT_FLAG_ADDSEND	address is sent in master mode or received and matches in slave mode interrupt flag
I2C_INT_FLAG_BTC	byte transmission finishes
I2C_INT_FLAG_ADD10SEND	header of 10-bit address is sent in master mode interrupt flag
I2C_INT_FLAG_STPDET	stop condition detected in slave mode interrupt flag
I2C_INT_FLAG_RBNE	I2C_DATA is not Empty during receiving interrupt flag
I2C_INT_FLAG_TBE	I2C_DATA is empty during transmitting interrupt flag
I2C_INT_FLAG_BERR	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag
I2C_INT_FLAG_LOSTARB	arbitration lost in master mode interrupt flag
I2C_INT_FLAG_AERR	acknowledge error interrupt flag
I2C_INT_FLAG_OUERR	over-run or under-run situation occurs in slave mode interrupt flag
I2C_INT_FLAG_PECERR	PEC error when receiving data interrupt flag

Member name	Function description
I2C_INT_FLAG_SMBTO	timeout signal in SMBus mode interrupt flag
I2C_INT_FLAG_SMBALT	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
I2C_INT_FLAG_STLO	start lost interrupt flag
I2C_INT_FLAG_STPSEND	stop condition sent interrupt flag

### Enum i2c2\_interrupt\_flag\_enum

Table 3-571. i2c2\_interrupt\_flag\_enum

Member name	Function description
I2C2_INT_FLAG_TI	transmit interrupt flag
I2C2_INT_FLAG_RBNE	I2C_RDATA is not empty during receiving interrupt flag
I2C2_INT_FLAG_ADDSEND	address received matches in slave mode interrupt flag
I2C2_INT_FLAG_NACK	not acknowledge interrupt flag
I2C2_INT_FLAG_STPDET	stop condition detected in slave mode interrupt flag
I2C2_INT_FLAG_TC	transfer complete in master mode interrupt flag
I2C2_INT_FLAG_TCR	transfer complete reload interrupt flag
I2C2_INT_FLAG_BERR	bus error interrupt flag
I2C2_INT_FLAG_LOSTARB	arbitration lost interrupt flag
I2C2_INT_FLAG_OUERR	overflow/underrun error in slave mode interrupt flag
I2C2_INT_FLAG_PECERR	PEC error interrupt flag
I2C2_INT_FLAG_TIMEOUT	timeout interrupt flag
I2C2_INT_FLAG_SMBALT	SMBus Alert interrupt flag

### i2c\_deinit

The description of i2c\_deinit is shown as below:

Table 3-572. 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}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

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

```
i2c_deinit (I2C0);
```

## i2c\_enable

The description of i2c\_enable is shown as below:

**Table 3-573. Function i2c\_enable**

<b>Function name</b>	i2c_enable
<b>Function prototype</b>	void i2c_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 */
```

```
i2c_enable (I2C0);
```

## i2c\_disable

The description of i2c\_disable is shown as below:

**Table 3-574. Function i2c\_disable**

<b>Function name</b>	i2c_disable
<b>Function prototype</b>	void i2c_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* disable I2C0 */
```

```
i2c_disable (I2C0);
```

### i2c\_start\_on\_bus

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

**Table 3-575. Function i2c\_start\_on\_bus**

<b>Function name</b>	i2c_start_on_bus
<b>Function prototype</b>	void i2c_start_on_bus(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a START condition on I2C bus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
i2c_start_on_bus (I2C0);
```

### i2c\_stop\_on\_bus

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

**Table 3-576. Function i2c\_stop\_on\_bus**

<b>Function name</b>	i2c_stop_on_bus
<b>Function prototype</b>	void i2c_stop_on_bus(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a STOP condition on I2C bus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	



-	-
---	---

Example:

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

```
i2c_stop_on_bus (I2C0);
```

### i2c\_slave\_response\_to\_gcall\_enable

The description of i2c\_slave\_response\_to\_gcall\_enable is shown as below:

**Table 3-577. Function i2c\_slave\_response\_to\_gcall\_enable**

<b>Function name</b>	i2c_slave_response_to_gcall_enable
<b>Function prototype</b>	void i2c_slave_response_to_gcall_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable the response to a general call
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the response to a general call */
```

```
i2c_slave_response_to_gcall_enable (I2C0);
```

### i2c\_slave\_response\_to\_gcall\_disable

The description of i2c\_slave\_response\_to\_gcall\_disable is shown as below:

**Table 3-578. Function i2c\_slave\_response\_to\_gcall\_disable**

<b>Function name</b>	i2c_slave_response_to_gcall_disable
<b>Function prototype</b>	void i2c_slave_response_to_gcall_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable the response to a general call
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* disable the response to a general call */
```

```
i2c_slave_response_to_gcall_disable (I2C0);
```

### i2c\_stretch\_scl\_low\_enable

The description of i2c\_stretch\_scl\_low\_enable is shown as below:

**Table 3-579. Function i2c\_stretch\_scl\_low\_enable**

<b>Function name</b>	i2c_stretch_scl_low_enable
<b>Function prototype</b>	void i2c_stretch_scl_low_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable to stretch SCL low when data is not ready in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable to stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_enable (I2C0);
```

### i2c\_stretch\_scl\_low\_disable

The description of i2c\_stretch\_scl\_low\_disable is shown as below:

**Table 3-580. Function i2c\_stretch\_scl\_low\_disable**

<b>Function name</b>	i2c_stretch_scl_low_disable
<b>Function prototype</b>	void i2c_stretch_scl_low_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable to stretch SCL low when data is not ready in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* disable to stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_disable (I2C0);
```

## i2c\_data\_transmit

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

**Table 3-581. Function i2c\_data\_transmit**

<b>Function name</b>	i2c_data_transmit
<b>Function prototype</b>	void i2c_data_transmit(uint32_t i2c_periph, uint32_t data);
<b>Function descriptions</b>	I2C transmit data 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>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 transmit data */
```

```
i2c_data_transmit (I2C0, 0x55);
```

## i2c\_data\_receive

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

**Table 3-582. Function i2c\_data\_receive**

<b>Function name</b>	i2c_data_receive
<b>Function prototype</b>	uint8_t i2c_data_receive(uint32_t i2c_periph);
<b>Function descriptions</b>	I2C receive data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	

-	-
<b>Return value</b>	
uint32_t	0x00..0xFF

Example:

```
/* I2C0 receive data */
```

```
uint32_t i2c_receiver;
```

```
i2c_receiver = i2c_data_receive(I2C0);
```

### i2c\_pec\_transfer

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

**Table 3-583. Function i2c\_pec\_transfer**

<b>Function name</b>	i2c_pec_transfer
<b>Function prototype</b>	void i2c_pec_transfer(uint32_t i2c_periph);
<b>Function descriptions</b>	I2C transfers PEC value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C transfers PEC value */
```

```
i2c_pec_transfer (I2C0);
```

### i2c\_pec\_enable

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

**Table 3-584. Function i2c\_pec\_enable**

<b>Function name</b>	i2c_pec_enable
<b>Function prototype</b>	void i2c_pec_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable I2C PEC calculation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral

<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
i2c_pec_enable (I2C0);
```

### i2c\_pec\_disable

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

**Table 3-585. Function i2c\_pec\_disable**

<b>Function name</b>	i2c_pec_disable
<b>Function prototype</b>	void i2c_pec_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable I2C PEC calculation
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
i2c_pec_disable (I2C0);
```

### i2c\_pec\_value\_get

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

**Table 3-586. Function i2c\_pec\_value\_get**

<b>Function name</b>	i2c_pec_value_get
<b>Function prototype</b>	uint32_t i2c_pec_value_get(uint32_t i2c_periph);
<b>Function descriptions</b>	get packet error checking value
<b>Precondition</b>	-
<b>The called functions</b>	-
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>	
<b>uint32_t</b>	PEC value

Example:

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

```
uint32_t pec_value;
```

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

### i2c\_clock\_config

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

**Table 3-587. 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)
<b>Input parameter{in}</b>	
<b>clkspeed</b>	i2c clock speed
<b>Input parameter{in}</b>	
<b>dutycyc</b>	duty cycle in fast mode
<i>I2C_DTCY_2</i>	T_low/T_high=2
<i>I2C_DTCY_16_9</i>	T_low/T_high=16/9
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure I2C0 clock speed as 100KHz*/
```

```
i2c_clock_config(I2C0, 100000, I2C_DTCY_2);
```

## i2c\_mode\_addr\_config

The description of i2c\_mode\_addr\_config is shown as below:

**Table 3-588. Function i2c\_mode\_addr\_config**

<b>Function name</b>	i2c_mode_addr_config
<b>Function prototype</b>	void i2c_mode_addr_config(uint32_t i2c_periph, uint32_t mode, uint32_t addformat, uint32_t addr);
<b>Function descriptions</b>	configure I2C address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>mode</b>	I2C mode select
<i>I2C_I2CMODE_ENABLE</i>	I2C mode
<i>I2C_SMBUSMODE_ENABLE</i>	SMBus mode
<b>Input parameter{in}</b>	
<b>addformat</b>	7bits or 10bits
<i>I2C_ADDFORMAT_7BITS</i>	7bits
<i>I2C_ADDFORMAT_10BITS</i>	10bits
<b>Input parameter{in}</b>	
<b>addr</b>	I2C address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure I2C0 address as 0x82, using 7 bits */
```

```
i2c_mode_addr_config(I2C0, I2C_I2CMODE_ENABLE, I2C_ADDFORMAT_7BITS, 0x82);
```

## i2c\_smbus\_type\_config

The description of i2c\_smbus\_type\_config is shown as below:

Table 3-589. Function `i2c_smbus_type_config`

Function name	<code>i2c_smbus_type_config</code>
Function prototype	<code>void i2c_smbus_type_config(uint32_t i2c_periph, uint32_t type);</code>
Function descriptions	SMBus type selection
Precondition	-
The called functions	-
Input parameter{in}	
<code>i2c_periph</code>	I2C peripheral
<code>I2Cx</code>	(x=0,1)
Input parameter{in}	
<code>type</code>	Device or host
<code>I2C_SMBUS_DEVICE</code>	device
<code>I2C_SMBUS_HOST</code>	host
Output parameter{out}	
-	-
Return value	
-	-

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 3-590. Function `i2c_ack_config`

Function name	<code>i2c_ack_config</code>
Function prototype	<code>void i2c_ack_config(uint32_t i2c_periph, uint32_t ack);</code>
Function descriptions	whether or not to send an ACK
Precondition	-
The called functions	-
Input parameter{in}	
<code>i2c_periph</code>	I2C peripheral
<code>I2Cx</code>	(x=0,1)
Input parameter{in}	
<code>ack</code>	I2C peripheral
<code>I2C_ACK_ENABLE</code>	ACK will be sent
<code>I2C_ACK_DISABLE</code>	ACK will not be sent
Output parameter{out}	
-	-
Return value	
-	-



Example:

```
/* I2C0 will send ACK */
```

```
i2c_ack_config (I2C0, I2C_ACK_ENABLE);
```

## i2c\_ackpos\_config

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

**Table 3-591. 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)
<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 3-592. 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>	master sends slave address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>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 3-593. 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)
<b>Input parameter{in}</b>	
<b>addr</b>	the second address in dual-address mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 dual-address */
```

```
i2c_dualaddr_enable (I2C0, 0x82);
```

## i2c\_dualaddr\_disable

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

**Table 3-594. Function i2c\_dualaddr\_disable**

<b>Function name</b>	i2c_dualaddr_disable
<b>Function prototype</b>	void i2c_dualaddr_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable 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)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable dual-address mode */
```

```
i2c_dualaddr_disable (I2C0);
```

## i2c\_dma\_config

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

**Table 3-595. Function i2c\_dma\_config**

<b>Function name</b>	i2c_dma_config
<b>Function prototype</b>	void i2c_dma_config (uint32_t i2c_periph, uint32_t dmastate);
<b>Function descriptions</b>	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)
<b>Input parameter{in}</b>	
<b>dmastate</b>	On or off
<i>I2C_DMA_ON</i>	DMA mode enable
<i>I2C_DMA_OFF</i>	DMA mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 DMA mode enable */
```

```
i2c_dma_config (I2C0, I2C_DMA_ON);
```

### i2c\_dma\_last\_transfer\_config

The description of i2c\_dma\_last\_transfer\_config is shown as below:

**Table 3-596. Function i2c\_dma\_last\_transfer\_config**

<b>Function name</b>	i2c_dma_last_transfer_config
<b>Function prototype</b>	void i2c_dma_last_transfer_config (uint32_t i2c_periph, uint32_t dmalast);
<b>Function descriptions</b>	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)
<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\_software\_reset\_config

The description of i2c\_software\_reset\_config is shown as below:

**Table 3-597. 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 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)

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

Example:

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

```
i2c_software_reset_config (I2C0, I2C_SRESET_SET);
```

### i2c\_smbus\_alert\_config

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

**Table 3-598. 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>	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

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 3-599. Function i2c\_smbus\_arp\_config**

<b>Function name</b>	i2c_smbus_arp_config
<b>Function prototype</b>	void i2c_smbus_arp_config(uint32_t i2c_periph, uint32_t arpstate);
<b>Function descriptions</b>	configure I2C ARP protocol in SMBus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>arpstate</b>	ARP protocol in SMBus switch
<i>I2C_ARP_ENABLE</i>	enable ARP
<i>I2C_ARP_DISABLE</i>	disable ARP
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 ARP protocol in SMBus switch */
```

```
i2c_smbus_arp_config (I2C0, I2C_ARP_ENABLE);
```

## i2c\_sam\_enable

The description of i2c\_sam\_enable is shown as below:

**Table 3-600. 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)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SAM_V interface */

i2c_sam_enable (I2C0);
```

## i2c\_sam\_disable

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

**Table 3-601. 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)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SAM_V interface */

i2c_sam_disable (I2C0);
```

## i2c\_sam\_timeout\_enable

The description of i2c\_sam\_timeout\_enable is shown as below:

**Table 3-602. 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)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable 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 3-603. 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)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SAM_V interface timeout detect */
```

```
i2c_sam_timeout_disable (I2C0);
```

### **i2c\_start\_early\_termination\_mode\_config**

The description of i2c\_start\_early\_termination\_mode\_config is shown as below:

**Table 3-604. Function i2c\_start\_early\_termination\_mode\_config**

<b>Function name</b>	i2c_start_early_termination_mode_config
<b>Function prototype</b>	void i2c_start_early_termination_mode_config(uint32_t i2c_periph, uint32_t mode);
<b>Function descriptions</b>	configure I2C start early termination mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>mode</b>	I2C start early termination mode
<i>STANDARD_I2C_PRO</i>	do as the standard i2c protocol



<i>TOCOL_MODE</i>	
<i>ARBITRATION_LOST_MODE</i>	do the same thing as arbitration lost
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure I2C start early termination mode */
```

```
i2c_start_early_termination_mode_config (I2C0, ARBITRATION_LOST_MODE);
```

## i2c\_timeout\_calculation\_enable

The description of i2c\_timeout\_calculation\_enable is shown as below:

**Table 3-605. Function i2c\_timeout\_calculation\_enable**

<b>Function name</b>	i2c_timeout_calculation_enable
<b>Function prototype</b>	void i2c_timeout_calculation_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable i2c timeout calculation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable i2c timeout calculation */
```

```
i2c_timeout_calculation_enable (I2C0);
```

## i2c\_timeout\_calculation\_disable

The description of i2c\_timeout\_calculation\_disable is shown as below:

**Table 3-606. Function i2c\_timeout\_calculation\_disable**

<b>Function name</b>	i2c_timeout_calculation_disable
<b>Function prototype</b>	void i2c_timeout_calculation_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable i2c timeout calculation
<b>Precondition</b>	-
<b>The called functions</b>	-

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

Example:

```
/* disable i2c timeout calculation */
```

```
i2c_timeout_calculation_disable (I2C0);
```

### i2c\_record\_received\_slave\_address\_enable

The description of i2c\_record\_received\_slave\_address\_enable is shown as below:

**Table 3-607. Function i2c\_record\_received\_slave\_address\_enable**

<b>Function name</b>	i2c_record_received_slave_address_enable
<b>Function prototype</b>	void i2c_record_received_slave_address_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable i2c record the received slave address to the transfer buffer register
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable i2c record the received slave address to the transfer buffer register */
```

```
i2c_record_received_slave_address_enable (I2C0);
```

### i2c\_record\_received\_slave\_address\_disable

The description of i2c\_record\_received\_slave\_address\_disable is shown as below:

**Table 3-608. Function i2c\_record\_received\_slave\_address\_disable**

<b>Function name</b>	i2c_record_received_slave_address_disable
<b>Function prototype</b>	void i2c_record_received_slave_address_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable i2c record the received slave address to the transfer buffer register
<b>Precondition</b>	-
<b>The called functions</b>	-

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

Example:

```
/* disable i2c record the received slave address to the transfer buffer register */
```

```
i2c_record_received_slave_address_disable (I2C0);
```

### i2c\_address\_bit\_compare\_config

The description of i2c\_address\_bit\_compare\_config is shown as below:

**Table 3-609. Function i2c\_address\_bit\_compare\_config**

<b>Function name</b>	i2c_address_bit_compare_config
<b>Function prototype</b>	void i2c_address_bit_compare_config(uint32_t i2c_periph, uint16_t compare_bits);
<b>Function descriptions</b>	define which bits of ADDRESS[7:1] need to compare with the incoming address byte
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
<b>mode</b>	the bits need to compare
ADDRESS_BIT1_COMPARE	address bit1 needs compare
ADDRESS_BIT2_COMPARE	address bit2 needs compare
ADDRESS_BIT3_COMPARE	address bit3 needs compare
ADDRESS_BIT4_COMPARE	address bit4 needs compare
ADDRESS_BIT5_COMPARE	address bit5 needs compare
ADDRESS_BIT6_COMPARE	address bit6 needs compare
ADDRESS_BIT7_COMPARE	address bit7 needs compare

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* define which bits of ADDRESS[7:1] need to compare with the incoming address byte */
i2c_address_bit_compare_config (I2C0, ADDRESS_BIT7_COMPARE);
```

### i2c\_status\_clear\_enable

The description of i2c\_status\_clear\_enable is shown as below:

**Table 3-610. Function i2c\_status\_clear\_enable**

Function name	i2c_status_clear_enable
Function prototype	void i2c_status_clear_enable(uint32_t i2c_periph);
Function descriptions	enable i2c status register clear
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable i2c status register clear */
i2c_status_clear_enable (I2C0);
```

### i2c\_status\_clear\_disable

The description of i2c\_status\_clear\_disable is shown as below:

**Table 3-611. Function i2c\_status\_clear\_disable**

Function name	i2c_status_clear_disable
Function prototype	void i2c_status_clear_disable(uint32_t i2c_periph);
Function descriptions	disable i2c status register clear
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable i2c status register clear */
```

```
i2c_status_clear_disable (I2C0);
```

## i2c\_status\_bit\_clear

The description of i2c\_status\_bit\_clear is shown as below:

**Table 3-612. Function i2c\_start\_early\_termination\_mode\_config**

Function name	i2c_status_bit_clear
Function prototype	void i2c_status_bit_clear(uint32_t i2c_periph, uint32_t clear_bit);
Function descriptions	clear i2c status register bit
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
clear_bit	which bit needs to clear
CLEAR_STPDET	clear STPDET bit in I2C_STAT0
CLEAR_ADD10SEND	clear ADD10SEND bit in I2C_STAT0
CLEAR_BTC	clear BTC bit in I2C_STAT0
CLEAR_ADDSEND	clear ADDSEND bit in I2C_STAT0
CLEAR_SBSEND	clear SBSEND bit in I2C_STAT0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear i2c status register bit */
```

```
i2c_status_bit_clear (I2C0, CLEAR_ADDSEND);
```

## i2c\_flag\_get

The description of i2c\_flag\_get is shown as below:

**Table 3-613. Function i2c\_flag\_get**

Function name	i2c_flag_get
---------------	--------------

<b>Function prototype</b>	FlagStatus i2c_flag_get(uint32_t i2c_periph, i2c_flag_enum flag);
<b>Function descriptions</b>	get I2C flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag, refer to <a href="#">Table 3-568. i2c_flag_enum</a> .
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
I2C_FLAG_LOSTARB	arbitration lost in master mode
I2C_FLAG_AERR	acknowledge error
I2C_FLAG_OUERR	overflow or underrun situation occurs in slave mode
I2C_FLAG_PECERR	PEC error when receiving data
I2C_FLAG_SMBTO	timeout signal in SMBus mode
I2C_FLAG_SMBALT	SMBus alert status
I2C_FLAG_MASTER	a flag indicating whether I2C block is in master or slave mode
I2C_FLAG_I2CBSY	busy flag
I2C_FLAG_TR	whether the I2C is a transmitter or a receiver
I2C_FLAG_RXGC	general call address (00h) received
I2C_FLAG_DEFSMB	default address of SMBus device
I2C_FLAG_HSTSMB	SMBus host header detected in slave mode
I2C_FLAG_DUMOD	dual flag in slave mode indicating which address is matched in dual-address mode
I2C_FLAG_TFF	txframe fall flag
I2C_FLAG_TFR	txframe rise flag
I2C_FLAG_RFF	rxframe fall flag
I2C_FLAG_RFR	rxframe rise flag
I2C_FLAG_STLO	start lost flag
I2C_FLAG_STPSEND	stop condition sent 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 3-614. Function i2c\_flag\_clear**

<b>Function name</b>	i2c_flag_clear
<b>Function prototype</b>	void i2c_flag_clear(uint32_t i2c_periph, i2c_flag_enum flag);
<b>Function descriptions</b>	clear I2C flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>flag</b>	flag type, refer to <a href="#">Table 3-568. i2c_flag_enum</a> .
<i>I2C_FLAG_SMBALT</i>	SMBus Alert status
<i>I2C_FLAG_SMBTO</i>	timeout signal in SMBus mode
<i>I2C_FLAG_PECERR</i>	PEC error when receiving data
<i>I2C_FLAG_OUERR</i>	over-run or under-run situation occurs in slave mode
<i>I2C_FLAG_AERR</i>	acknowledge error
<i>I2C_FLAG_LOSTARB</i>	arbitration lost in master mode
<i>I2C_FLAG_BERR</i>	a bus error
<i>I2C_FLAG_ADDSEND</i>	cleared by reading I2C_STAT0 and reading I2C_STAT1
<i>I2C_FLAG_TFF</i>	txframe fall flag
<i>I2C_FLAG_TFR</i>	txframe rise flag
<i>I2C_FLAG_RFF</i>	rxframe fall flag
<i>I2C_FLAG_RFR</i>	rxframe rise flag
<i>I2C_FLAG_STLO</i>	start lost flag
<i>I2C_FLAG_STPSEND</i>	stop condition sent flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear a bus error flag*/
```

i2c\_flag\_clear (I2C0, I2C\_FLAG\_BERR);

## i2c\_interrupt\_enable

The description of i2c\_interrupt\_enable is shown as below:

**Table 3-615. Function i2c\_interrupt\_enable**

<b>Function name</b>	i2c_interrupt_enable
<b>Function prototype</b>	void i2c_interrupt_enable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);
<b>Function descriptions</b>	enable I2C interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt type, refer to <a href="#">Table 3-569. i2c_interrupt_enum</a> .
<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
<i>I2C_INT_STLO</i>	start lost interrupt enable
<i>I2C_INT_STPSEND</i>	stop condition sent 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 3-616. Function i2c\_interrupt\_disable**

<b>Function name</b>	i2c_interrupt_disable
<b>Function prototype</b>	void i2c_interrupt_disable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);
<b>Function descriptions</b>	disable I2C interrupt



<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt type, refer to <a href="#">Table 3-569. i2c interrupt enum.</a>
<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>	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
<i>I2C_INT_STLO</i>	start lost interrupt enable
<i>I2C_INT_STPSEND</i>	stop condition sent interrupt enable
<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 3-617. Function i2c\_interrupt\_flag\_get**

<b>Function name</b>	i2c_interrupt_flag_get
<b>Function prototype</b>	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	get I2C interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt flag, refer to <a href="#">Table 3-570. i2c interrupt flag enum.</a>
<i>I2C_INT_FLAG_SBSE</i> <i>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 O</i>	timeout signal in SMBus mode interrupt flag
<i>I2C_INT_FLAG_SMBA LT</i>	SMBus Alert status interrupt flag
<i>I2C_INT_FLAG_TFF</i>	txframe fall interrupt flag
<i>I2C_INT_FLAG_TFR</i>	txframe rise interrupt flag
<i>I2C_INT_FLAG_RFF</i>	rxframe fall interrupt flag
<i>I2C_INT_FLAG_RFR</i>	rxframe rise interrupt flag
<i>I2C_INT_FLAG_STLO</i>	start lost interrupt flag
<i>I2C_INT_FLAG_STPSE ND</i>	stop condition sent interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* check the byte transmission finishes interrupt flag is set or not*/
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_interrupt_flag_get (I2C0, I2C_INT_FLAG_BTC);
```

## **i2c\_interrupt\_flag\_clear**

The description of i2c\_interrupt\_flag\_clear is shown as below:

**Table 3-618. Function i2c\_interrupt\_flag\_clear**

<b>Function name</b>	i2c_interrupt_flag_clear
<b>Function prototype</b>	void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	clear I2C interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt flag, refer to <a href="#">Table 3-570. i2c_interrupt_flag_enum</a> .
I2C_INT_FLAG_ADDS END	address is sent in master mode or received and matches in slave mode interrupt flag
I2C_INT_FLAG_BERR	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag
I2C_INT_FLAG_LOSTA RB	arbitration lost in master mode interrupt flag
I2C_INT_FLAG_AERR	acknowledge error interrupt flag
I2C_INT_FLAG_OUER R	over-run or under-run situation occurs in slave mode interrupt flag
I2C_INT_FLAG_PECE RR	PEC error when receiving data interrupt flag
I2C_INT_FLAG_SMBT O	timeout signal in SMBus mode interrupt flag
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
I2C_INT_FLAG_STLO	start lost interrupt flag
I2C_INT_FLAG_STPSE ND	stop condition sent 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);
```

## i2c\_timing\_config

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

**Table 3-619. Function i2c\_timing\_config**

<b>Function name</b>	i2c_timing_config
<b>Function prototype</b>	void i2c_timing_config(uint32_t i2c_periph, uint32_t psc, uint32_t scl_dely, uint32_t sda_dely);
<b>Function descriptions</b>	configure the timing parameters
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Input parameter{in}</b>	
<b>psc</b>	0-0xf, timing prescaler
<b>Input parameter{in}</b>	
<b>scl_dely</b>	0-0xf, data setup time
<b>Input parameter{in}</b>	
<b>sda_dely</b>	0-0xf, data hold time
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the timing parameters */
i2c_timing_config (I2C2, 0x1, 0x2, 0x1);
```

## i2c\_digital\_noise\_filter\_config

The description of i2c\_digital\_noise\_filter\_config is shown as below:

**Table 3-620. Function i2c\_digital\_noise\_filter\_config**

<b>Function name</b>	i2c_digital_noise_filter_config
<b>Function prototype</b>	void i2c_digital_noise_filter_config(uint32_t i2c_periph, uint32_t filter_length);
<b>Function descriptions</b>	configure digital noise filter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Input parameter{in}</b>	

<b>filter_length</b>	<b>filter_length</b>
<i>FILTER_DISABLE</i>	digital filter is disabled
<i>FILTER_LENGTH_1</i>	digital filter is enabled and filter spikes with a length of up to 1 tI2CCLK
<i>FILTER_LENGTH_2</i>	digital filter is enabled and filter spikes with a length of up to 2 tI2CCLK
<i>FILTER_LENGTH_3</i>	digital filter is enabled and filter spikes with a length of up to 3 tI2CCLK
<i>FILTER_LENGTH_4</i>	digital filter is enabled and filter spikes with a length of up to 4 tI2CCLK
<i>FILTER_LENGTH_5</i>	digital filter is enabled and filter spikes with a length of up to 5 tI2CCLK
<i>FILTER_LENGTH_6</i>	digital filter is enabled and filter spikes with a length of up to 6 tI2CCLK
<i>FILTER_LENGTH_7</i>	digital filter is enabled and filter spikes with a length of up to 7 tI2CCLK
<i>FILTER_LENGTH_8</i>	digital filter is enabled and filter spikes with a length of up to 8 tI2CCLK
<i>FILTER_LENGTH_9</i>	digital filter is enabled and filter spikes with a length of up to 9 tI2CCLK
<i>FILTER_LENGTH_10</i>	digital filter is enabled and filter spikes with a length of up to 10 tI2CCLK
<i>FILTER_LENGTH_11</i>	digital filter is enabled and filter spikes with a length of up to 11 tI2CCLK
<i>FILTER_LENGTH_12</i>	digital filter is enabled and filter spikes with a length of up to 12 tI2CCLK
<i>FILTER_LENGTH_13</i>	digital filter is enabled and filter spikes with a length of up to 13 tI2CCLK
<i>FILTER_LENGTH_14</i>	digital filter is enabled and filter spikes with a length of up to 14 tI2CCLK
<i>FILTER_LENGTH_15</i>	digital filter is enabled and filter spikes with a length of up to 15 tI2CCLK
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C2 digital filter filters spikes with a length of up to 1 tI2CCLK */
```

```
i2c_digital_noise_filter_config(I2C2, FILTER_LENGTH_1);
```

## i2c\_analog\_noise\_filter\_enable

The description of i2c\_analog\_noise\_filter\_enable is shown as below:

**Table 3-621. Function i2c\_analog\_noise\_filter\_enable**

<b>Function name</b>	<b>i2c_analog_noise_filter_enable</b>
<b>Function prototype</b>	<b>void i2c_analog_noise_filter_enable(uint32_t i2c_periph);</b>
<b>Function descriptions</b>	<b>enable analog noise filter</b>
<b>Precondition</b>	<b>-</b>
<b>The called functions</b>	<b>-</b>
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable analog noise filter */
```

```
i2c_analog_noise_filter_enable (I2C2);
```

### i2c\_analog\_noise\_filter\_disable

The description of i2c\_analog\_noise\_filter\_disable is shown as below:

**Table 3-622. Function i2c\_analog\_noise\_filter\_disable**

<b>Function name</b>	i2c_analog_noise_filter_disable
<b>Function prototype</b>	void i2c_analog_noise_filter_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable analog noise filter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable analog noise filter */
```

```
i2c_analog_noise_filter_disable (I2C2);
```

### i2c\_wakeup\_from\_deepsleep\_enable

The description of i2c\_wakeup\_from\_deepsleep\_enable is shown as below:

**Table 3-623. Function i2c\_wakeup\_from\_deepsleep\_enable**

<b>Function name</b>	i2c_wakeup_from_deepsleep_enable
<b>Function prototype</b>	void i2c_wakeup_from_deepsleep_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable wakeup from Deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable wakeup from Deep-sleep mode */

i2c_wakeup_from_deepsleep_enable (I2C2);
```

### i2c\_wakeup\_from\_deepsleep\_disable

The description of i2c\_wakeup\_from\_deepsleep\_disable is shown as below:

**Table 3-624. Function i2c\_wakeup\_from\_deepsleep\_disable**

<b>Function name</b>	i2c_wakeup_from_deepsleep_disable
<b>Function prototype</b>	void i2c_wakeup_from_deepsleep_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable wakeup from Deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable wakeup from Deep-sleep mode */

i2c_wakeup_from_deepsleep_disable (I2C2);
```

### i2c\_master\_clock\_config

The description of i2c\_master\_clock\_config is shown as below:

**Table 3-625. Function i2c\_master\_clock\_config**

<b>Function name</b>	i2c_master_clock_config
<b>Function prototype</b>	void i2c_master_clock_config(uint32_t i2c_periph, uint32_t sclh, uint32_t scll);
<b>Function descriptions</b>	configure the SCL high and low period of clock in master mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Input parameter{in}</b>	
<b>sclh</b>	0-0xff, SCL high period
<b>Input parameter{in}</b>	

<b>scll</b>	0-0xff, SCL low period
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the SCL and SDA period of clock in master mode */
```

```
i2c_master_clock_config (I2C2, 0x0f, 0x0f);
```

## i2c2\_master\_addressing

The description of i2c2\_master\_addressing is shown as below:

**Table 3-626. Function i2c2\_master\_transfer\_direction\_config**

<b>Function name</b>	i2c2_master_addressing
<b>Function prototype</b>	void i2c2_master_addressing(uint32_t i2c_periph, uint32_t address, uint32_t trans_direction);
<b>Function descriptions</b>	configure i2c slave addresss and transfer direction in master mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Input parameter{in}</b>	
<b>address</b>	0-0x3FF except reserved address, I2C slave address to be sent
<b>Input parameter{in}</b>	
<b>trans_direction</b>	I2C transfer direction in master mode
<i>I2C2_MASTER_TRANSMIT</i>	master transmit
<i>I2C2_MASTER_RECEIVE</i>	master receive
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* send slave address to I2C bus */
```

```
i2c2_master_addressing (I2C2, 0x82, I2C2_MASTER_TRANSMIT);
```



## i2c\_address10\_header\_enable

The description of i2c\_address10\_header\_enable is shown as below:

**Table 3-627. Function i2c\_address10\_header\_enable**

<b>Function name</b>	i2c_address10_header_enable
<b>Function prototype</b>	void i2c_address10_header_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	10-bit address header executes read direction only in master receive mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* 10-bit address header executes read direction only in master receive mode */
```

```
i2c_address10_header_enable (I2C2);
```

## i2c\_address10\_header\_disable

The description of i2c\_address10\_header\_disable is shown as below:

**Table 3-628. Function i2c\_address10\_header\_disable**

<b>Function name</b>	i2c_address10_header_disable
<b>Function prototype</b>	void i2c_address10_header_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	10-bit address header executes complete sequence in master receive mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* 10-bit address header executes complete sequence in master receive mode */
```

```
i2c_address10_header_disable (I2C2);
```

## i2c\_address10\_enable

The description of i2c\_address10\_enable is shown as below:

**Table 3-629. Function i2c\_address10\_enable**

<b>Function name</b>	i2c_address10_enable
<b>Function prototype</b>	void i2c_address10_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable 10-bit addressing mode in master mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable 10-bit addressing mode in master mode */
```

```
i2c_address10_enable (I2C2);
```

## i2c\_address10\_disable

The description of i2c\_address10\_disable is shown as below:

**Table 3-630. Function i2c\_address10\_disable**

<b>Function name</b>	i2c_address10_disable
<b>Function prototype</b>	void i2c_address10_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable 10-bit addressing mode in master mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable 10-bit addressing mode in master mode */
```

```
i2c_address10_disable (I2C2);
```

## i2c\_automatic\_end\_enable

The description of i2c\_automatic\_end\_enable is shown as below:

**Table 3-631. Function i2c\_automatic\_end\_enable**

<b>Function name</b>	i2c_automatic_end_enable
<b>Function prototype</b>	void i2c_automatic_end_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable I2C automatic end mode in master mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_enable (I2C2);
```

## i2c\_automatic\_end\_disable

The description of i2c\_automatic\_end\_disable is shown as below:

**Table 3-632. Function i2c\_automatic\_end\_disable**

<b>Function name</b>	i2c_automatic_end_disable
<b>Function prototype</b>	void i2c_automatic_end_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable I2C automatic end mode in master mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_disable (I2C2);
```

## i2c\_address\_config

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

**Table 3-633. Function i2c\_address\_config**

<b>Function name</b>	i2c_address_config
<b>Function prototype</b>	void i2c_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_format);
<b>Function descriptions</b>	configure i2c slave address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Input parameter{in}</b>	
<b>address</b>	I2C address
<b>Input parameter{in}</b>	
<b>addr_format</b>	7bits or 10bits
<i>I2C_ADDFORMAT_7BITS</i>	7bits
<i>I2C_ADDFORMAT_10BITS</i>	10bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure i2c slave address */
```

```
i2c_address_config (I2C2, 0x82, I2C_ADDFORMAT_7BITS);
```

## i2c\_address\_disable

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

**Table 3-634. Function i2c\_address\_disable**

<b>Function name</b>	i2c_address_disable
<b>Function prototype</b>	void i2c_address_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable i2c address in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable i2c address in slave mode */
```

```
i2c_address_disable (I2C2);
```

## i2c\_second\_address\_config

The description of i2c\_second\_address\_config is shown as below:

**Table 3-635. Function i2c\_second\_address\_config**

Function name	i2c_second_address_config
Function prototype	void i2c_second_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_mask);
Function descriptions	configure i2c second slave address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Input parameter{in}	
address	I2C address
Input parameter{in}	
addr_mask	the bits not need to compare
ADDRESS2_NO_MASK	no mask, all the bits must be compared
ADDRESS2_MASK_BIT1	ADDRESS2[1] is masked, only ADDRESS2[7:2] are compared
ADDRESS2_MASK_BIT1_2	ADDRESS2[2:1] is masked, only ADDRESS2[7:3] are compared
ADDRESS2_MASK_BIT1_3	ADDRESS2[3:1] is masked, only ADDRESS2[7:4] are compared
ADDRESS2_MASK_BIT1_4	ADDRESS2[4:1] is masked, only ADDRESS2[7:5] are compared
ADDRESS2_MASK_BIT1_5	ADDRESS2[5:1] is masked, only ADDRESS2[7:6] are compared
ADDRESS2_MASK_BIT1_6	ADDRESS2[6:1] is masked, only ADDRESS2[7] are compared
ADDRESS2_MASK_ALL	all the ADDRESS2[7:1] bits are masked

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure i2c second slave address */
```

```
i2c_second_address_config (I2C2, 0x82, ADDRESS2_MASK_BIT1_2);
```

## i2c\_second\_address\_disable

The description of i2c\_second\_address\_disable is shown as below:

**Table 3-636. Function i2c\_second\_address\_disable**

Function name	i2c_second_address_disable
Function prototype	void i2c_second_address_disable(uint32_t i2c_periph);
Function descriptions	disable i2c second address in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable i2c second address in slave mode */
```

```
i2c_second_address_disable (I2C2);
```

## i2c\_receieved\_address\_get

The description of i2c\_receieved\_address\_get is shown as below:

**Table 3-637. Function i2c\_receieved\_address\_get**

Function name	i2c_receieved_address_get
Function prototype	uint32_t i2c_receieved_address_get(uint32_t i2c_periph);
Function descriptions	get received match address in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)

Output parameter{out}	
-	-
Return value	
uint32_t	0x00..0x7F

Example:

```
/* get received match address in slave mode */
```

```
uint32_t address;
```

```
address = i2c_receivied_address_get (I2C2);
```

### i2c\_slave\_byte\_control\_enable

The description of i2c\_slave\_byte\_control\_enable is shown as below:

**Table 3-638. Function i2c\_slave\_byte\_control\_enable**

Function name	i2c_slave_byte_control_enable
Function prototype	void i2c_slave_byte_control_enable(uint32_t i2c_periph);
Function descriptions	enable slave byte control
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable slave byte control */
```

```
i2c_slave_byte_control_enable (I2C2);
```

### i2c\_slave\_byte\_control\_disable

The description of i2c\_slave\_byte\_control\_disable is shown as below:

**Table 3-639. Function i2c\_slave\_byte\_control\_disable**

Function name	i2c_slave_byte_control_disable
Function prototype	void i2c_slave_byte_control_disable(uint32_t i2c_periph);
Function descriptions	disable slave byte control
Precondition	-
The called functions	-
Input parameter{in}	

<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable slave byte control */
```

```
i2c_slave_byte_control_disable (I2C2);
```

### i2c\_nack\_enable

The description of i2c\_nack\_enable is shown as below:

**Table 3-640. Function i2c\_nack\_enable**

<b>Function name</b>	i2c_nack_enable
<b>Function prototype</b>	void i2c_nack_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a NACK in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* generate a NACK in slave mode */
```

```
i2c_nack_enable (I2C2);
```

### i2c\_nack\_disable

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

**Table 3-641. Function i2c\_nack\_disable**

<b>Function name</b>	i2c_nack_disable
<b>Function prototype</b>	void i2c_nack_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a ACK in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	



<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* generate a ACK in slave mode */
```

```
i2c_nack_disable (I2C2);
```

### i2c\_reload\_enable

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

**Table 3-642. Function i2c\_reload\_enable**

<b>Function name</b>	i2c_reload_enable
<b>Function prototype</b>	void i2c_reload_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable I2C reload mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C reload mode */
```

```
i2c_reload_enable (I2C2);
```

### i2c\_reload\_disable

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

**Table 3-643. Function i2c\_reload\_disable**

<b>Function name</b>	i2c_reload_disable
<b>Function prototype</b>	void i2c_reload_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable I2C reload mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C reload mode */
```

```
i2c_reload_disable (I2C2);
```

### i2c\_transfer\_byte\_number\_config

The description of i2c\_transfer\_byte\_number\_config is shown as below:

**Table 3-644. Function i2c\_transfer\_byte\_number\_config**

<b>Function name</b>	i2c_transfer_byte_number_config
<b>Function prototype</b>	void i2c_transfer_byte_number_config(uint32_t i2c_periph, uint32_t byte_number);
<b>Function descriptions</b>	configure number of bytes to be transferred
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Input parameter{in}</b>	
<b>byte_number</b>	0x0-0xFF, number of bytes to be transferred
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure number of bytes to be transferred */
```

```
i2c_transfer_byte_number_config (I2C2, 0xFF);
```

### i2c2\_dma\_enable

The description of i2c2\_dma\_enable is shown as below:

**Table 3-645. Function i2c2\_dma\_enable**

<b>Function name</b>	i2c2_dma_enable
<b>Function prototype</b>	void i2c2_dma_enable(uint32_t i2c_periph, uint8_t dma);
<b>Function descriptions</b>	enable I2C DMA for transmission or reception

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Input parameter{in}</b>	
<b>dma</b>	I2C DMA
<i>I2C2_DMA_TRANSMIT</i>	transmit data using DMA
<i>I2C2_DMA_RECEIVE</i>	receive data using DMA
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C DMA for transmission or reception */
i2c2_dma_enable (I2C2, I2C2_DMA_RECEIVE);
```

## i2c2\_dma\_disable

The description of i2c2\_dma\_disable is shown as below:

**Table 3-646. Function i2c2\_dma\_disable**

<b>Function name</b>	i2c2_dma_disable
<b>Function prototype</b>	void i2c2_dma_disable(uint32_t i2c_periph, uint8_t dma);
<b>Function descriptions</b>	disable I2C DMA for transmission or reception
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Input parameter{in}</b>	
<b>dma</b>	I2C DMA
<i>I2C2_DMA_TRANSMIT</i>	transmit data using DMA
<i>I2C2_DMA_RECEIVE</i>	receive data using DMA
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C DMA for transmission or reception */
i2c2_dma_disable (I2C2, I2C2_DMA_RECEIVE);
```

## i2c\_smbus\_alert\_enable

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

**Table 3-647. Function i2c\_smbus\_alert\_enable**

<b>Function name</b>	i2c_smbus_alert_enable
<b>Function prototype</b>	void i2c_smbus_alert_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable SMBus Alert
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SMBus Alert */
i2c_smbus_alert_enable (I2C2);
```

## i2c\_smbus\_alert\_disable

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

**Table 3-648. Function i2c\_smbus\_alert\_disable**

<b>Function name</b>	i2c_smbus_alert_disable
<b>Function prototype</b>	void i2c_smbus_alert_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable SMBus Alert
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SMBus Alert */
i2c_smbus_alert_disable (I2C2);
```

## i2c\_smbus\_default\_addr\_enable

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

**Table 3-649. Function i2c\_smbus\_default\_addr\_enable**

<b>Function name</b>	i2c_smbus_default_addr_enable
<b>Function prototype</b>	void i2c_smbus_default_addr_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable SMBus device default address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SMBus device default address */
i2c_smbus_default_addr_enable (I2C2);
```

## i2c\_smbus\_default\_addr\_disable

The description of i2c\_smbus\_default\_addr\_disable is shown as below:

**Table 3-650. Function i2c\_smbus\_default\_addr\_disable**

<b>Function name</b>	i2c_smbus_default_addr_disable
<b>Function prototype</b>	void i2c_smbus_default_addr_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable SMBus device default address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SMBus device default address */
i2c_smbus_default_addr_disable (I2C2);
```

## i2c\_smbus\_host\_addr\_enable

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

**Table 3-651. Function i2c\_smbus\_host\_addr\_enable**

<b>Function name</b>	i2c_smbus_host_addr_enable
<b>Function prototype</b>	void i2c_smbus_host_addr_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable SMBus Host address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SMBus Host address */
i2c_smbus_host_addr_enable (I2C2);
```

## i2c\_smbus\_host\_addr\_disable

The description of i2c\_smbus\_host\_addr\_disable is shown as below:

**Table 3-652. Function i2c\_smbus\_host\_addr\_disable**

<b>Function name</b>	i2c_smbus_host_addr_disable
<b>Function prototype</b>	void i2c_smbus_host_addr_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable SMBus Host address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SMBus Host address */
i2c_smbus_host_addr_disable (I2C2);
```

## i2c\_extented\_clock\_timeout\_enable

The description of i2c\_extented\_clock\_timeout\_enable is shown as below:

**Table 3-653. Function i2c\_extented\_clock\_timeout\_enable**

<b>Function name</b>	i2c_extented_clock_timeout_enable
<b>Function prototype</b>	void i2c_extented_clock_timeout_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable extended clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable extended clock timeout detection */
```

```
i2c_extented_clock_timeout_enable (I2C2);
```

## i2c\_extented\_clock\_timeout\_disable

The description of i2c\_extented\_clock\_timeout\_disable is shown as below:

**Table 3-654. Function i2c\_extented\_clock\_timeout\_disable**

<b>Function name</b>	i2c_extented_clock_timeout_disable
<b>Function prototype</b>	void i2c_extented_clock_timeout_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable extended clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable extended clock timeout detection */
```

```
i2c_extented_clock_timeout_disable (I2C2);
```

## i2c\_clock\_timeout\_enable

The description of i2c\_clock\_timeout\_enable is shown as below:

**Table 3-655. Function i2c\_clock\_timeout\_enable**

<b>Function name</b>	i2c_clock_timeout_enable
<b>Function prototype</b>	void i2c_clock_timeout_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable clock timeout detection */
i2c_clock_timeout_enable (I2C2);
```

## i2c\_clock\_timeout\_disable

The description of i2c\_clock\_timeout\_disable is shown as below:

**Table 3-656. Function i2c\_clock\_timeout\_disable**

<b>Function name</b>	i2c_clock_timeout_disable
<b>Function prototype</b>	void i2c_clock_timeout_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable clock timeout detection */
i2c_clock_timeout_disable (I2C2);
```



## i2c\_bus\_timeout\_b\_config

The description of i2c\_bus\_timeout\_b\_config is shown as below:

**Table 3-657. Function i2c\_bus\_timeout\_b\_config**

<b>Function name</b>	i2c_bus_timeout_b_config
<b>Function prototype</b>	void i2c_bus_timeout_b_config(uint32_t i2c_periph, uint32_t timeout);
<b>Function descriptions</b>	configure bus timeout B
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Input parameter{in}</b>	
<b>timeout</b>	0-0xfff, bus timeout B
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure bus timeout B */
```

```
i2c_bus_timeout_b_config (I2C2, 0xff);
```

## i2c\_bus\_timeout\_a\_config

The description of i2c\_bus\_timeout\_a\_config is shown as below:

**Table 3-658. Function i2c\_bus\_timeout\_a\_config**

<b>Function name</b>	i2c_bus_timeout_a_config
<b>Function prototype</b>	void i2c_bus_timeout_a_config(uint32_t i2c_periph, uint32_t timeout);
<b>Function descriptions</b>	configure bus timeout A
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Input parameter{in}</b>	
<b>timeout</b>	0-0xfff, bus timeout A
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure bus timeout A */
```

```
i2c_bus_timeout_a_config (I2C2, 0xff);
```

### i2c\_idle\_clock\_timeout\_config

The description of i2c\_idle\_clock\_timeout\_config is shown as below:

**Table 3-659. Function i2c\_idle\_clock\_timeout\_config**

<b>Function name</b>	i2c_idle_clock_timeout_config
<b>Function prototype</b>	void i2c_idle_clock_timeout_config(uint32_t i2c_periph, uint32_t timeout);
<b>Function descriptions</b>	configure idle clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Input parameter{in}</b>	
<b>timeout</b>	bus timeout A
<i>BUSTOA_DETECT_SCL_LOW</i>	BUSTOA is used to detect SCL low timeout
<i>BUSTOA_DETECT_IDLE</i>	BUSTOA is used to detect both SCL and SDA high timeout when the bus is idle
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure idle clock timeout detection */
```

```
i2c_idle_clock_timeout_config (I2C2, BUSTOA_DETECT_SCL_LOW);
```

### i2c2\_flag\_get

The description of i2c2\_flag\_get is shown as below:

**Table 3-660. Function i2c2\_flag\_get**

<b>Function name</b>	i2c2_flag_get
<b>Function prototype</b>	FlagStatus i2c2_flag_get(uint32_t i2c_periph, uint32_t flag);
<b>Function descriptions</b>	get I2C flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Input parameter{in}</b>	
<b>flag</b>	I2C flags
<i>I2C2_FLAG_TBE</i>	I2C2_TDATA is empty during transmitting
<i>I2C2_FLAG_TI</i>	transmit interrupt
<i>I2C2_FLAG_RBNE</i>	I2C2_RDATA is not empty during receiving
<i>I2C2_FLAG_ADDSEND</i>	address received matches in slave mode
<i>I2C2_FLAG_NACK</i>	not acknowledge flag
<i>I2C2_FLAG_STPDET</i>	STOP condition detected in slave mode
<i>I2C2_FLAG_TC</i>	transfer complete in master mode
<i>I2C2_FLAG_TCR</i>	transfer complete reload
<i>I2C2_FLAG_BERR</i>	bus error
<i>I2C2_FLAG_LOSTARB</i>	arbitration Lost
<i>I2C2_FLAG_OUERR</i>	overflow/underrun error in slave mode
<i>I2C2_FLAG_PECERR</i>	PEC error
<i>I2C2_FLAG_TIMEOUT</i>	timeout flag
<i>I2C2_FLAG_SMBALT</i>	SMBus Alert
<i>I2C2_FLAG_I2CBSY</i>	busy flag
<i>I2C2_FLAG_TR</i>	whether the I2C is a transmitter or a receiver in slave mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get I2C flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c2_flag_get (I2C2, I2C2_FLAG_TBE);
```

### i2c2\_flag\_clear

The description of i2c2\_flag\_clear is shown as below:

**Table 3-661. Function i2c2\_flag\_clear**

<b>Function name</b>	i2c2_flag_clear
<b>Function prototype</b>	void i2c2_flag_clear(uint32_t i2c_periph, uint32_t flag);
<b>Function descriptions</b>	clear I2C flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)

Input parameter{in}	
flag	I2C flags
<i>I2C2_FLAG_ADDSEND</i>	address received matches in slave mode
<i>I2C2_FLAG_NACK</i>	not acknowledge flag
<i>I2C2_FLAG_STPDET</i>	STOP condition detected in slave mode
<i>I2C2_FLAG_BERR</i>	bus error
<i>I2C2_FLAG_LOSTARB</i>	arbitration Lost
<i>I2C2_FLAG_OUERR</i>	overflow/underrun error in slave mode
<i>I2C2_FLAG_PECERR</i>	PEC error
<i>I2C2_FLAG_TIMEOUT</i>	timeout flag
<i>I2C2_FLAG_SMBALT</i>	SMBus Alert
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear a bus error flag*/
```

```
i2c2_flag_clear (I2C2, I2C2_FLAG_BERR);
```

## i2c2\_interrupt\_enable

The description of i2c2\_interrupt\_enable is shown as below:

**Table 3-662. Function i2c2\_interrupt\_enable**

Function name	i2c2_interrupt_enable
Function prototype	void i2c2_interrupt_enable(uint32_t i2c_periph, uint32_t interrupt);
Function descriptions	enable I2C interrupt
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=2)
Input parameter{in}	
interrupt	I2C interrupts
<i>I2C2_INT_ERR</i>	error interrupt
<i>I2C2_INT_TC</i>	transfer complete interrupt
<i>I2C2_INT_STPDET</i>	stop detection interrupt
<i>I2C2_INT_NACK</i>	not acknowledge received interrupt
<i>I2C2_INT_ADDM</i>	address match interrupt
<i>I2C2_INT_RBNE</i>	receive interrupt
<i>I2C2_INT_TI</i>	transmit interrupt
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable I2C2 transmit interrupt */
```

```
i2c2_interrupt_enable (I2C2, I2C2_INT_TI);
```

### i2c2\_interrupt\_disable

The description of i2c2\_interrupt\_disable is shown as below:

**Table 3-663. Function i2c2\_interrupt\_disable**

<b>Function name</b>	i2c2_interrupt_disable
<b>Function prototype</b>	void i2c2_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=2)
<b>Input parameter{in}</b>	
<b>interrupt</b>	I2C interrupts
<i>I2C2_INT_ERR</i>	error interrupt
<i>I2C2_INT_TC</i>	transfer complete interrupt
<i>I2C2_INT_STPDET</i>	stop detection interrupt
<i>I2C2_INT_NACK</i>	not acknowledge received interrupt
<i>I2C2_INT_ADDM</i>	address match interrupt
<i>I2C2_INT_RBNE</i>	receive interrupt
<i>I2C2_INT_TI</i>	transmit interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C2 transmit interrupt */
```

```
i2c2_interrupt_disable (I2C2, I2C2_INT_TI);
```

### i2c2\_interrupt\_flag\_get

The description of i2c2\_interrupt\_flag\_get is shown as below:

**Table 3-664. Function i2c2\_interrupt\_flag\_get**

<b>Function name</b>	i2c2_interrupt_flag_get
<b>Function prototype</b>	FlagStatus i2c2_interrupt_flag_get(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	get I2C interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
I2Cx	(x=2)
<b>Input parameter{in}</b>	
<b>int_flag</b>	I2C interrupt flags, refer to <a href="#">Table 3-571. i2c2_interrupt_flag_enum</a> .
I2C2_INT_FLAG_TI	transmit interrupt flag
I2C2_INT_FLAG_RBNE	I2C2_RDATA is not empty during receiving interrupt flag
I2C2_INT_FLAG_ADDSEND	address received matches in slave mode interrupt flag
I2C2_INT_FLAG_NACK	not acknowledge interrupt flag
I2C2_INT_FLAG_STOPDET	stop condition detected in slave mode interrupt flag
I2C2_INT_FLAG_TC	transfer complete in master mode interrupt flag
I2C2_INT_FLAG_TCR	transfer complete reload interrupt flag
I2C2_INT_FLAG_BERR	bus error interrupt flag
I2C2_INT_FLAG_LOSTARB	arbitration lost interrupt flag
I2C2_INT_FLAG_OVERR	overflow/underrun error in slave mode interrupt flag
I2C2_INT_FLAG_PECERR	PEC error interrupt flag
I2C2_INT_FLAG_TIMEOUT	timeout interrupt flag
I2C2_INT_FLAG_SMBALERT	SMBus Alert interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get I2C interrupt flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c2_interrupt_flag_get (I2C2, I2C2_INT_FLAG_TI);
```

## i2c2\_interrupt\_flag\_clear

The description of i2c2\_interrupt\_flag\_clear is shown as below:

**Table 3-665. Function i2c2\_interrupt\_flag\_clear**

<b>Function name</b>	i2c2_interrupt_flag_clear
<b>Function prototype</b>	void i2c2_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	clear I2C interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=2)
<b>Input parameter{in}</b>	
<b>int_flag</b>	I2C interrupt flags, refer to <a href="#">Table 3-571. i2c2_interrupt_flag_enum</a> .
<i>I2C2_INT_FLAG_ADD SEND</i>	address received matches in slave mode interrupt flag
<i>I2C2_INT_FLAG_NACK</i>	not acknowledge interrupt flag
<i>I2C2_INT_FLAG_STOP DET</i>	stop condition detected in slave mode interrupt flag
<i>I2C2_INT_FLAG_BERR</i>	bus error interrupt flag
<i>I2C2_INT_FLAG_LOST ARB</i>	arbitration lost interrupt flag
<i>I2C2_INT_FLAG_OVERR RR</i>	overrun/underrun error in slave mode interrupt flag
<i>I2C2_INT_FLAG_PECERR</i>	PEC error interrupt flag
<i>I2C2_INT_FLAG_TIMEOUT</i>	timeout interrupt flag
<i>I2C2_INT_FLAG_SMBALERT</i>	SMBus Alert interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear a bus error flag */
```

```
i2c2_interrupt_flag_clear (I2C2, I2C2_INT_FLAG_BERR);
```

## 3.19. MISC

MISC is a software package that provide the interfaces for NVIC and SysTick. The NVIC and SysTick registers are listed in chapter [3.19.1](#), the MISC firmware functions are introduced in chapter [3.19.2](#).

### 3.19.1. Descriptions of Peripheral registers

**Table 3-666. NVIC Registers**

Registers	Descriptions
ISER <sup>(1)</sup>	Interrupt Set Enable Register
ICER <sup>(1)</sup>	Interrupt Clear Enable Register
ISPR <sup>(1)</sup>	Interrupt Set Pending Register
ICPR <sup>(1)</sup>	Interrupt Clear Pending Register
IABR <sup>(1)</sup>	Interrupt Active bit Register
ITNS <sup>(1)</sup>	Interrupt Non-Secure State Register
IPR <sup>(1)</sup>	Interrupt Priority Register
STIR <sup>(1)</sup>	Software Trigger Interrupt Register
CPUID <sup>(2)</sup>	CPUID Base Register
ICSR <sup>(2)</sup>	Interrupt Control and State Register
VTOR <sup>(2)</sup>	Vector Table Offset Register
AIRCR <sup>(2)</sup>	Application Interrupt and Reset Control Register
SCR <sup>(2)</sup>	System Control Register
CCR <sup>(2)</sup>	Configuration Control Register
SHPR <sup>(2)</sup>	System Handlers Priority Registers
SHCSR <sup>(2)</sup>	System Handler Control and State Register

1. refer to the structure NVIC\_Type, is defined in the core\_cm33.h file

2. refer to the structure SCB\_Type, is defined in the core\_cm33.h file

**Table 3-667. SysTick Registers**

Registers	Descriptions
CTRL <sup>(1)</sup>	SysTick Control and Status Register
LOAD <sup>(1)</sup>	SysTick Reload Value Register
VAL <sup>(1)</sup>	SysTick Current Value Register
CALIB <sup>(1)</sup>	SysTick Calibration Register

1. refer to the structure SysTick\_Type, is defined in the core\_cm33.h file

### 3.19.2. Descriptions of Peripheral functions

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



Table 3-668. MISC firmware function

Function name	Function description
<code>nvic_priority_group_set</code>	set the priority group
<code>nvic_irq_enable</code>	enable NVIC interrupt request
<code>nvic_irq_disable</code>	disable NVIC interrupt request
<code>nvic_system_reset</code>	initiates a system reset request to reset the MCU
<code>nvic_vector_table_set</code>	set the NVIC vector table base address
<code>system_lowpower_set</code>	set the state of the low power mode
<code>system_lowpower_reset</code>	reset the state of the low power mode
<code>systick_clksource_set</code>	set the systick clock source

## Enum IRQn\_Type

Table 3-669. IRQn\_Type

Member name	Function description
<code>WWDGT_IRQn</code>	window watchDog timer interrupt
<code>LVD_IRQn</code>	LVD through EXTI line detect interrupt
<code>TAMPER_IRQn</code>	tamper through EXTI line detect
<code>RTC_IRQn</code>	RTC through EXTI line interrupt
<code>FMC_IRQn</code>	FMC interrupt
<code>RCU_CTC_IRQn</code>	RCU and CTC interrupt
<code>EXTI0_IRQn</code>	EXTI line 0 interrupts
<code>EXTI1_IRQn</code>	EXTI line 1 interrupts
<code>EXTI2_IRQn</code>	EXTI line 2 interrupts
<code>EXTI3_IRQn</code>	EXTI line 3 interrupts
<code>EXTI4_IRQn</code>	EXTI line 4 interrupts
<code>DMA0_Channel0_IRQn</code>	DMA0 channel0 interrupt
<code>DMA0_Channel1_IRQn</code>	DMA0 channel1 interrupt
<code>DMA0_Channel2_IRQn</code>	DMA0 channel2 interrupt
<code>DMA0_Channel3_IRQn</code>	DMA0 channel3 interrupt
<code>DMA0_Channel4_IRQn</code>	DMA0 channel4 interrupt
<code>DMA0_Channel5_IRQn</code>	DMA0 channel5 interrupt
<code>DMA0_Channel6_IRQn</code>	DMA0 channel6 interrupt
<code>ADC0_1_IRQn</code>	ADC0 and ADC1 interrupt
<code>USBD_HP_CAN0_TX_IRQn / CAN0_TX_IRQn</code>	USBD High Priority or CAN0 TX interrupts / CAN0 transmit interrupt
<code>USBD_LP_CAN0_RX0_IRQn / CAN0_RX0_IRQn</code>	USBD Low Priority or CAN0 RX0 interrupts / CAN0 receive0 interrupts
<code>CAN0_RX1_IRQn</code>	CAN0 receive1 interrupts
<code>CAN0_EWMC_IRQn</code>	CAN0 EWMC interrupts
<code>EXTI5_9_IRQn</code>	EXTI[9:5] interrupts
<code>TIMER0_BRK_TIMER8_IRQn</code>	TIMER0 break and TIMER8 interrupts
<code>TIMER0_UP_TIMER9_IRQn</code>	TIMER0 update and TIMER9 interrupts

Member name	Function description
TIMER0_TRG_CMT_TIMER10_IRQn	TIMER0 trigger and commutation and TIMER10 interrupts
TIMER0_Channel_IRQn	TIMER0 channel capture compare interrupt
TIMER1_IRQn	TIMER1 interrupt
TIMER2_IRQn	TIMER2 interrupt
TIMER3_IRQn	TIMER3 interrupt
I2C0_EV_IRQn	I2C0 event interrupt
I2C0_ER_IRQn	I2C0 error interrupt
I2C1_EV_IRQn	I2C1 event interrupt
I2C1_ER_IRQn	I2C1 error interrupt
SPI0_IRQn	SPI0 interrupt
SPI1_I2S1ADD_IRQn	SPI1 or I2S1ADD 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
USBD_WKUP_IRQn / USBHS_WKUP_IRQn	USBD / USBHS 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
EXMC_IRQn	EXMC global interrupt
SDIO_IRQn	SDIO global interrupt
TIMER4_IRQn	TIMER4 global interrupt
SPI2_I2S2ADD_IRQn	SPI2 or I2S2ADD global interrupt
UART3_IRQn	UART3 global interrupt
UART4_IRQn	UART4 global interrupt
TIMER5_DAC_IRQn	TIMER5 or 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_Channel4_IRQn / DMA1_Channel3_IRQn	DMA1 channel3 and channel4 global interrupt / DMA1 channel3 global interrupt
DMA1_Channel4_IRQn	DMA1 channel4 global interrupt
ENET_IRQn	ENET global 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

Member name	Function description
CAN1_EWMC_IRQn	CAN1 EWMC interrupt
USBHS_IRQn	USBHS global interrupt
SHRTIMER_IRQ2_IRQn	SHRTIMER_IRQ2 interrupt
SHRTIMER_IRQ3_IRQn	SHRTIMER_IRQ3 interrupt
SHRTIMER_IRQ4_IRQn	SHRTIMER_IRQ4 interrupt
SHRTIMER_IRQ5_IRQn	SHRTIMER_IRQ5 interrupt
SHRTIMER_IRQ6_IRQn	SHRTIMER_IRQ6 interrupt
USBHS_EP1_OUT_IRQn	USBHS end point 1 out interrupt
USBHS_EP1_IN_IRQn	USBHS end point 1 in interrupt
SHRTIMER_IRQ0_IRQn	SHRTIMER_IRQ0 interrupt
SHRTIMER_IRQ1_IRQn	SHRTIMER_IRQ1 interrupt
CAN2_TX_IRQn	CAN2 TX interrupt
CAN2_RX0_IRQn	CAN2 RX0 interrupt
CAN2_RX1_IRQn	CAN2 RX1 interrupt
CAN2_EWMC_IRQn	CAN2 EWMC interrupt
I2C2_EV_IRQn	I2C2 EV interrupt
I2C2_ER_IRQn	I2C2 ER interrupt
USART5_IRQn	USART5 global interrupt
I2C2_WKUP_IRQn	I2C2 Wakeup interrupt
USART5_WKUP_IRQn	USART5 Wakeup interrupt
TMU_IRQn	TMU interrupt

## nvic\_priority\_group\_set

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

**Table 3-670. Function nvic\_priority\_group\_set**

Function name	nvic_priority_group_set
Function prototype	void nvic_priority_group_set(uint32_t nvic_prigroup);
Function descriptions	set 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_PRE4_SUB0</i>	4 bits for pre-emption priority 0 bits for subpriority
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* priority group configuration, 0 bits for pre-emption priority 4 bits for subpriority */
nvic_priority_group_set(NVIC_PRIGROUP_PRE0_SUB4);
```

### **nvic\_irq\_enable**

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

**Table 3-671. Function nvic\_irq\_enable**

<b>Function name</b>	nvic_irq_enable
<b>Function prototype</b>	void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority);
<b>Function descriptions</b>	enable NVIC interrupt request
<b>Precondition</b>	-
<b>The called functions</b>	nvic_priority_group_set
<b>Input parameter{in}</b>	
<b>nvic_irq</b>	NVIC interrupt, refer to <a href="#">Table 3-669. IRQn Type</a>
<b>Input parameter{in}</b>	
<b>nvic_irq_pre_priority</b>	the pre-emption priority needed to set
<b>Input parameter{in}</b>	
<b>nvic_irq_sub_priority</b>	the subpriority needed to set
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable window watchDog timer interrupt , pre-emption priority is 1, subpriority is 1 */
nvic_irq_enable(WWDGT_IRQn, 1, 1);
```

### **nvic\_irq\_disable**

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

**Table 3-672. 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 interrupt request
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>nvic_irq</b>	NVIC interrupt, refer to <a href="#">Table 3-669. IRQn_Type</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable window watchDog timer interrupt */
nvic_irq_disable(WWDGT_IRQn);
```

## **nvic\_system\_reset**

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

**Table 3-673. Function nvic\_system\_reset**

<b>Function name</b>	nvic_system_reset
<b>Function prototype</b>	void nvic_system_reset(void);
<b>Function descriptions</b>	initiates a system reset request to reset the MCU
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initiates a system reset request to reset the MCU */
nvic_system_reset();
```

## **nvic\_vector\_table\_set**

The description of nvic\_vector\_table\_set is shown as below:

**Table 3-674. Function nvic\_vector\_table\_set**

<b>Function name</b>	nvic_vector_table_set
<b>Function prototype</b>	void nvic_vector_table_set(uint32_t nvic_vect_tab, uint32_t offset);
<b>Function descriptions</b>	set the NVIC vector table base address
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>nvic_vect_tab</b>	the RAM or FLASH base address
<i>NVIC_VECTTAB_RAM</i>	RAM base address
<i>NVIC_VECTTAB_FLASH</i>	Flash base address
Input parameter{in}	
<b>offset</b>	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 3-675. Function system\_lowpower\_set**

<b>Function name</b>	system_lowpower_set
<b>Function prototype</b>	void system_lowpower_set(uint8_t lowpower_mode);
<b>Function descriptions</b>	set the state of the low power mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>lowpower_mode</b>	the low power mode state
<i>SCB_LPM_SLEEP_EXIT_ISR</i>	if chose this para, the system always enter low power mode by exiting from ISR
<i>SCB_LPM_DEEPSLEEP</i>	if chose this para, the system will enter the DEEPSLEEP mode
<i>SCB_LPM_WAKE_BY_ALL_INT</i>	if chose this para, the lowpower mode can be woke up by all the enable and disable interrupts
Output parameter{out}	
-	-
Return value	
-	-

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 3-676. Function system\_lowpower\_reset**

<b>Function name</b>	system_lowpower_reset
<b>Function prototype</b>	void system_lowpower_reset(uint8_t lowpower_mode);
<b>Function descriptions</b>	reset the state of the low power mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lowpower_mode</b>	the low power mode state
SCB_LPM_SLEEP_EXIT_ISR	if chose this para, the system will exit low power mode by exiting from ISR
SCB_LPM_DEEPSLEEP	if chose this para, the system will enter the SLEEP mode
SCB_LPM_WAKE_BY_ALL_INT	if chose this para, the lowpower mode only can be woke up by the enable 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 3-677. 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>	

-	-
Return value	
-	-

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 five types of power saving modes, including Sleep, Deep-sleep, Deep-sleep 1, Deep-sleep 2 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 3-678. PMU Registers**

Registers	Descriptions
PMU_CTL0	Control register 0
PMU_CS0	Control and status register 0
PMU_CTL1	Control register 1
PMU_CS1	Control and status register 1

### 3.20.2. Descriptions of Peripheral functions

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

**Table 3-679. PMU firmware function**

Function name	Function description
pmu_deinit	deinitialize the PMU
pmu_lvd_select	select low voltage detector threshold
pmu_lvd_disable	disable PMU lvd
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 mode
pmu_lowdriver_mode_disable	disable low-driver mode in deep-sleep mode
pmu_lowpower_driver_config	in deep-sleep mode, driver mode when use low power LDO
pmu_normalpower_driver_config	in deep-sleep mode, driver mode when use normal power LDO



Function name	Function description
pmu_to_sleepmode	PMU work in sleep mode
pmu_to_deepsleepmode	PMU work in deepsleep mode
pmu_to_deepsleepmode_1	PMU work in deepsleep mode 1
pmu_to_deepsleepmode_2	PMU work in deepsleep mode 2
pmu_to_standbymode	pmu work in standby mode
pmu_wakeup_pin_enable	enable wakeup pin
pmu_wakeup_pin_disable	disable wakeup pin
pmu_backup_write_enable	enable backup domain write
pmu_backup_write_disable	disable backup domain write
pmu_flag_get	get flag state
pmu_flag_clear	clear flag bit

## pmu\_deinit

The description of pmu\_deinit is shown as below:

**Table 3-680. Function pmu\_deinit**

Function name	pmu_deinit
Function prototype	void pmu_deinit(void);
Function descriptions	deinitialize the PMU
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset PMU */
pmu_deinit ();
```

## pmu\_lvd\_select

The description of pmu\_lvd\_select is shown as below:

**Table 3-681. 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}	
<b>lvdt_n</b>	voltage threshold value
<i>PMU_LVDT_0</i>	voltage threshold is 2.1V
<i>PMU_LVDT_1</i>	voltage threshold is 2.3V
<i>PMU_LVDT_2</i>	voltage threshold is 2.4V
<i>PMU_LVDT_3</i>	voltage threshold is 2.6V
<i>PMU_LVDT_4</i>	voltage threshold is 2.7V
<i>PMU_LVDT_5</i>	voltage threshold is 2.9V
<i>PMU_LVDT_6</i>	voltage threshold is 3.0V
<i>PMU_LVDT_7</i>	voltage threshold is 3.1V
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 3-682. Function pmu\_lvd\_disable**

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

Example:

```
/* disable PMU lvd */
```

```
pmu_lvd_disable ();
```

### pmu\_highdriver\_mode\_enable

The description of pmu\_highdriver\_mode\_enable is shown as below:

**Table 3-683. Function pmu\_highdriver\_mode\_enable**

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

Example:

```
/* enable high-driver mode */
pmu_highdriver_mode_enable ();
```

## pmu\_highdriver\_mode\_disable

The description of pmu\_highdriver\_mode\_disable is shown as below:

**Table 3-684. Function pmu\_highdriver\_mode\_disable**

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

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 3-685. Function pmu\_highdriver\_switch\_select**

<b>Function name</b>	pmu_highdriver_switch_select
----------------------	------------------------------

<b>Function prototype</b>	void pmu_highdriver_switch_select(uint32_t highdr_switch);
<b>Function descriptions</b>	switch high-driver mode
<b>Precondition</b>	-
<b>The called functions</b>	pmu_flag_get()
<b>Input parameter{in}</b>	
<b>highdr_switch</b>	enable or disable high-driver mode switch
<i>PMU_HIGHDR_SWITC H_NONE</i>	disable high-driver mode switch
<i>PMU_HIGHDR_SWITC H_EN</i>	enable high-driver mode switch
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable high-driver mode switch */
pmu_highdriver_switch_select (PMU_HIGHDR_SWITCH_EN);
```

## pmu\_lowdriver\_mode\_enable

The description of pmu\_lowdriver\_mode\_enable is shown as below:

**Table 3-686. Function pmu\_lowdriver\_mode\_enable**

<b>Function name</b>	pmu_lowdriver_mode_enable
<b>Function prototype</b>	void pmu_lowdriver_mode_enable(void);
<b>Function descriptions</b>	enable low-driver mode in deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable low-driver mode in deep-sleep mode */
pmu_lowdriver_mode_enable ();
```

## pmu\_lowdriver\_mode\_disable

The description of pmu\_lowdriver\_mode\_disable is shown as below:

**Table 3-687. 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>	e disable low-driver mode in deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable low-driver mode in deep-sleep mode */
```

```
pmu_lowdriver_mode_disable ();
```

## pmu\_lowpower\_driver\_config

The description of pmu\_lowpower\_driver\_config is shown as below:

**Table 3-688. Function pmu\_lowpower\_driver\_config**

<b>Function name</b>	pmu_lowpower_driver_config
<b>Function prototype</b>	void pmu_lowpower_driver_config(uint32_t mode);
<b>Function descriptions</b>	driver mode when use low power LDO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mode</b>	driver mode
<i>PMU_NORMALDR_LO WPWR</i>	normal driver when use low power LDO
<i>PMU_LOWDR_LOWP WR</i>	low-driver mode enabled when LDEN is 11 and use low power LDO
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* normal driver when use low power LDO */
```

```
pmu_lowpower_driver_config (PMU_NORMALDR_LOWPWR);
```

## pmu\_normalpower\_driver\_config

The description of pmu\_normalpower\_driver\_config is shown as below:

**Table 3-689. Function pmu\_normalpower\_driver\_config**

<b>Function name</b>	pmu_normalpower_driver_config
<b>Function prototype</b>	void pmu_normalpower_driver_config (uint32_t mode);
<b>Function descriptions</b>	driver mode when use normal power LDO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mode</b>	driver mode
<i>PMU_NORMALDR_LOWPWR</i>	normal driver when use normal power LDO
<i>PMU_LOWDR_LOWPWR</i>	low-driver mode enabled when LDEN is 11 and use normal power LDO
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* normal driver when use normal power LDO */
```

```
pmu_normalpower_driver_config (PMU_NORMALDR_LOWPWR);
```

## pmu\_to\_sleepmode

The description of pmu\_to\_sleepmode is shown as below:

**Table 3-690. Function pmu\_to\_sleepmode**

<b>Function name</b>	pmu_to_sleepmode
<b>Function prototype</b>	void pmu_to_sleepmode(uint8_t sleepmodecmd);
<b>Function descriptions</b>	PMU work in sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sleepmodecmd</b>	command to enter sleep mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work in sleep mode */
```

```
pmu_to_sleepmode (WFI_CMD);
```

## pmu\_to\_deepsleepmode

The description of pmu\_to\_deepsleepmode is shown as below:

**Table 3-691. Function pmu\_to\_deepsleepmode**

<b>Function name</b>	pmu_to_deepsleepmode
<b>Function prototype</b>	void pmu_to_deepsleepmode(uint32_t ldo, uint32_t lowdrive, uint8_t deepsleepmodecmd);
<b>Function descriptions</b>	PMU work in deepsleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ldo</b>	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
<b>Input parameter{in}</b>	
<b>lowdrive</b>	low-driver mode
PMU_LOWDRIVER_ENABLE	low-driver mode enable in deep-sleep mode
PMU_LOWDRIVER_DISABLE	low-driver mode disable in deep-sleep mode
<b>Input parameter{in}</b>	
<b>deepsleepmodecmd</b>	command to enter deepsleep mode
WFI_CMD	use WFI command
WFE_CMD	use WFE command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work in deepsleep mode */
```

```
pmu_to_deepsleepmode (PMU_LDO_NORMAL, PMU_LOWDRIVER_ENABLE, WFI_CMD);
```

## pmu\_to\_deepsleepmode\_1

The description of pmu\_to\_deepsleepmode\_1 is shown as below:

**Table 3-692. Function pmu\_to\_deepsleepmode\_1**

<b>Function name</b>	pmu_to_deepsleepmode_1
----------------------	------------------------

<b>Function prototype</b>	void pmu_to_deepsleepmode_1(uint32_t ldo, uint32_t lowdrive, uint8_t deepsleepmode1cmd);
<b>Function descriptions</b>	PMU work in deepsleep mode 1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ldo</b>	ldo work mode
<i>PMU_LDO_NORMAL</i>	LDO normal work when pmu enter deepsleep mode 1
<i>PMU_LDO_LOWPOWER</i>	LDO work at low power mode when pmu enter deepsleep mode 1
<b>Input parameter{in}</b>	
<b>lowdrive</b>	low-driver mode
<i>PMU_LOWDRIVER_ENABLE</i>	low-driver mode enable in deep-sleep 1 mode
<i>PMU_LOWDRIVER_DISABLE</i>	low-driver mode disable in deep-sleep 1 mode
<b>Input parameter{in}</b>	
<b>deepsleepmode1cmd</b>	command to enter deepsleep mode 1
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work in deepsleep mode 1 */
```

```
pmu_to_deepsleepmode_1(PMU_LDO_NORMAL, PMU_LOWDRIVER_ENABLE, WFI_CMD);
```

## pmu\_to\_deepsleepmode\_2

The description of pmu\_to\_deepsleepmode\_2 is shown as below:

**Table 3-693. Function pmu\_to\_deepsleepmode\_2**

<b>Function name</b>	pmu_to_deepsleepmode_2
<b>Function prototype</b>	void pmu_to_deepsleepmode_2(uint32_t ldo, uint32_t lowdrive, uint8_t deepsleepmode2cmd);
<b>Function descriptions</b>	PMU work in deepsleep mode 2
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ldo</b>	ldo work mode



<i>PMU_LDO_NORMAL</i>	LDO normal work when pmu enter deepsleep mode 2
<i>PMU_LDO_LOWPOWER</i>	LDO work at low power mode when pmu enter deepsleep mode 2
<b>Input parameter{in}</b>	
<b>lowdrive</b>	low-driver mode
<i>PMU_LOWDRIVER_ENABLE</i>	low-driver mode enable in deep-sleep 2 mode
<i>PMU_LOWDRIVER_DISABLE</i>	low-driver mode disable in deep-sleep 2 mode
<b>Input parameter{in}</b>	
<b>deepsleepmode2cmd</b>	command to enter deepsleep mode 2
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work in deepsleep mode 2 */
```

```
pmu_to_deepsleepmode_2(PMU_LDO_NORMAL, PMU_LOWDRIVER_ENABLE, WFI_CMD);
```

## pmu\_to\_standbymode

The description of pmu\_to\_standbymode is shown as below:

**Table 3-694. Function pmu\_to\_standbymode**

<b>Function name</b>	pmu_to_standbymode
<b>Function prototype</b>	void pmu_to_standbymode(void);
<b>Function descriptions</b>	pmu work in standby mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work in standby mode */
```

```
pmu_to_standby ();
```

## pmu\_backup\_write\_enable

The description of pmu\_backup\_write\_enable is shown as below:

**Table 3-695. 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 3-696. Function pmu\_backup\_write\_disable**

<b>Function name</b>	pmu_backup_write_disable
<b>Function prototype</b>	void pmu_backup_write_disable (void);
<b>Function descriptions</b>	disable backup domain write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable backup domain write */
pmu_backup_write_disable ();
```

## pmu\_wakeup\_pin\_enable

The description of pmu\_wakeup\_pin\_enable is shown as below:

**Table 3-697. Function pmu\_wakeup\_pin\_enable**

<b>Function name</b>	pmu_wakeup_pin_enable
<b>Function prototype</b>	void pmu_wakeup_pin_enable(uint32_t wakeup_pin);
<b>Function descriptions</b>	enable wakeup pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_pin</b>	the pin to wakeup system
PMU_WAKEUP_PIN0	WKUP Pin 0 (PA0)
PMU_WAKEUP_PIN1	WKUP Pin 1 (PC13)
PMU_WAKEUP_PIN2	WKUP Pin 2 (PE6)
PMU_WAKEUP_PIN3	WKUP Pin 3 (PA2)
PMU_WAKEUP_PIN4	WKUP Pin 4 (PC5)
PMU_WAKEUP_PIN5	WKUP Pin 5 (PB5)
PMU_WAKEUP_PIN6	WKUP Pin 6 (PB15)
PMU_WAKEUP_PIN7	WKUP Pin 7 (PF8)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable wakeup pin 0 */
```

```
pmu_wakeup_pin_enable (PMU_WAKEUP_PIN0);
```

## pmu\_wakeup\_pin\_disable

The description of pmu\_wakeup\_pin\_disable is shown as below:

**Table 3-698. Function pmu\_wakeup\_pin\_disable**

<b>Function name</b>	pmu_wakeup_pin_disable
<b>Function prototype</b>	void pmu_wakeup_pin_disable (uint32_t wakeup_pin);
<b>Function descriptions</b>	disable wakeup pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_pin</b>	the pin to wakeup system
PMU_WAKEUP_PIN0	WKUP Pin 0 (PA0)
PMU_WAKEUP_PIN1	WKUP Pin 1 (PC13)
PMU_WAKEUP_PIN2	WKUP Pin 2 (PE6)

<i>PMU_WAKEUP_PIN3</i>	WKUP Pin 3 (PA2)
<i>PMU_WAKEUP_PIN4</i>	WKUP Pin 4 (PC5)
<i>PMU_WAKEUP_PIN5</i>	WKUP Pin 5 (PB5)
<i>PMU_WAKEUP_PIN6</i>	WKUP Pin 6 (PB15)
<i>PMU_WAKEUP_PIN7</i>	WKUP Pin 7 (PF8)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable wakeup pin 0 */
```

```
pmu_wakeup_pin_disable (PMU_WAKEUP_PIN0);
```

## pmu\_flag\_clear

The description of `pmu_flag_clear` is shown as below:

**Table 3-699. Function `pmu_flag_clear`**

<b>Function name</b>	<code>pmu_flag_clear</code>
<b>Function prototype</b>	<code>void pmu_flag_clear(uint32_t flag);</code>
<b>Function descriptions</b>	clear flag bit
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flag
<i>PMU_FLAG_RESET_WAKEUP</i>	reset wakeup flag
<i>PMU_FLAG_RESET_STANDBY</i>	reset standby flag
<i>PMU_FLAG_RESET_DEEPSLEEP_1</i>	reset deep-sleep 1 mode status flag
<i>PMU_FLAG_RESET_DEEPSLEEP_2</i>	reset deep-sleep 2 mode status flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear flag bit */
```

```
pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

## pmu\_flag\_get

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

**Table 3-700. 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
PMU_FLAG_WAKEUP	wakeup flag
PMU_FLAG_STANDBY	standby flag
PMU_FLAG_LVD	low voltage detector status flag
PMU_FLAG_HDRF	high-driver ready flag
PMU_FLAG_HDSRF	high-driver switch ready flag
PMU_FLAG_LDRF	low-driver mode ready flag
PMU_FLAG_DEEPSLEEP_1	deep-sleep 1 mode status flag
PMU_FLAG_DEEPSLEEP_2	deep-sleep 2 mode status flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

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

```
FlagStatus status;
```

```
status = pmu_flag_get (PMU_FLAG_WAKEUP);
```

## 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 3-701. RCU Registers**

Registers	Descriptions
RCU_CTL	control register
RCU_CFG0	clock configuration register 0
RCU_INT	clock interrupt register
RCU_APB2RST	APB2 reset register
RCU_APB1RST	APB1 reset register
RCU_AHBEN	AHB enable register
RCU_APB2EN	APB2 enable register
RCU_APB1EN	APB1 enable register
RCU_BDCTL	backup domain control register
RCU_RSTSCK	reset source/clock register
RCU_AHBRST	AHB reset register(only for CL、EPRT、E508 series)
RCU_CFG1	clock configuration register 1
RCU_DSV	deep-sleep mode voltage register
RCU_ADDCTL	additional clock control register
RCU_ADDCFG	additional clock configuration register(only for CL、E508 series)
RCU_ADDINT	additional clock interrupt register
RCU_PLLSSCTL	PLL clock spread spectrum control register(only for CL、EPRT、E508 series)
RCU_CFG2	clock configuration register 2
RCU_ADDAPB1RST	APB1 additional reset register
RCU_ADDAPB1EN	APB1 additional enable register

### 3.21.2. Descriptions of Peripheral functions

RCU firmware function are listed in the table shown as below:

**Table 3-702. RCU firmware function**

Function name	Function description
rcu_deinit	deinitialize the RCU
rcu_periph_clock_enable	enable the peripherals clock
rcu_periph_clock_disable	disable the peripherals clock
rcu_periph_clock_sleep_enable	enable the peripherals clock when sleep mode
rcu_periph_clock_sleep_disable	disable the peripherals clock when sleep mode
rcu_periph_reset_enable	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

Function name	Function description
rcu_apb2_clock_config	configure the APB2 clock prescaler selection
rcu_ckout0_config	configure the CK_OUT0 clock source
rcu_pll_config	configure the main PLL clock
rcu_pllpresel_config	configure the PLL clock source preselection
rcu_predv0_config	configure the PREDV0 division factor
rcu_predv1_config	configure the PREDV1 division factor
rcu_pll1_config	configure the PLL1 clock
rcu_pll2_config	configure the PLL2 clock
rcu_pllusbpresel_config	configure the PLLUSB clock source preselection(only for CL、E508 series)
rcu_pllusbpredv_config	configure the PLLUSBPREDV division factor and clock source(only for CL、E508 series)
rcu_pllusb_config	configure the PLLUSB clock(only for CL、E508 series)
rcu_adc_clock_config	configure the ADC prescaler factor
rcu_usb_clock_config	configure the USB prescaler factor
rcu_rtc_clock_config	configure the RTC clock source selection
rcu_shrtimer_clock_config	configure the SHRTIMER clock source selection(except for the EPRT series)
rcu_usart5_clock_config	configure the usart5 clock source selection
rcu_i2c2_clock_config	configure the I2C2 clock source selection
rcu_i2s1_clock_config	configure the I2S1 clock source selection(only for CL、EPRT、E508 series)
rcu_i2s2_clock_config	configure the I2S2 clock source selection(only for CL、EPRT、E508 series)
rcu_ck48m_clock_config	configure the CK48M clock selection
rcu_usbhsel_config	configure the USBHSEL source clock selection(only for CL、E508 series)
rcu_usbdv_config	configure the USBHSDV division factor(only for CL、E508 series)
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_irc8m_adjust_value_set	set the IRC8M adjust value
rcu_hxtal_clock_monitor_enable	enable the HXTAL clock monitor
rcu_hxtal_clock_monitor_disable	disable the HXTAL clock monitor
rcu_deepsleep_voltage_set	set the deep-sleep mode voltage value
rcu_clock_freq_get	get the system clock, bus clock frequency
rcu_flag_get	get the clock stabilization and peripheral reset flags

Function name	Function description
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

## Enum rcu\_periph\_enum

**Table 3-703. Enum rcu\_periph\_enum**

enum name	Function description
RCU_DMA0	DMA0 clock
RCU_DMA1	DMA1 clock
RCU_CRC	CRC clock
RCU_EXMC	EXMC clock
RCU_SDIO	SDIO clock(only for HD series)
RCU_USBHS	USBHS clock(only for CL、E508 series)
RCU_ULPI	ULPI clock(only for CL、E508 series)
RCU_ENET	ENET clock(only for CL、EPRT、E508 series)
RCU_ENETTX	ENETTX clock(only for CL、EPRT、E508 series)
RCU_ENETRX	ENETRX clock(only for CL、EPRT、E508 series)
RCU_TMU	TMU clock
RCU_SQPI	SQPI clock
RCU_TIMER1	TIMER1 clock
RCU_TIMER2	TIMER2 clock
RCU_TIMER3	TIMER3 clock
RCU_TIMER4	TIMER4 clock
RCU_TIMER5	TIMER5 clock
RCU_TIMER6	TIMER6 clock
RCU_TIMER11	TIMER11 clock(except for EPRT series)
RCU_TIMER12	TIMER12 clock(except for EPRT series)
RCU_TIMER13	TIMER13 clock(except for EPRT series)
RCU_WWDGT	WWDGT clock
RCU_SPI1	SPI1 clock
RCU_SPI2	SPI2 clock
RCU_USART1	USART1 clock
RCU_USART2	USART2 clock
RCU_UART3	UART3 clock
RCU_UART4	UART4 clock
RCU_I2C0	I2C0 clock
RCU_I2C1	I2C1 clock
RCU_USBD	USBD clock(only for HD、EPRT series)
RCU_I2C2	I2C2 clock



enum name	Function description
RCU_CAN0	CAN0 clock(except for EPRT series)
RCU_CAN1	CAN1 clock(except for EPRT series)
RCU_BKPI	BKPI clock
RCU_PMU	PMU clock
RCU_DAC	DAC clock
RCU_RTC	RTC clock
RCU_CTC	CTC clock
RCU_CAN2	CAN2 clock(only for CL、E508 series)
RCU_AF	alternate function clock
RCU_GPIOA	GPIOA clock
RCU_GPIOB	GPIOB clock
RCU_GPIOC	GPIOC clock
RCU_GPIOD	GPIOD clock
RCU_GPIOE	GPIOE clock
RCU_GPIOF	GPIOF clock
RCU_GPIOG	GPIOG clock
RCU_ADC0	ADC0 clock
RCU_ADC1	ADC1 clock
RCU_TIMER0	TIMER0 clock
RCU_SPI0	SPI0 clock
RCU_TIMER7	TIMER7 clock
RCU_USART0	USART0 clock
RCU_ADC2	ADC2 clock(except for CL series)
RCU_TIMER8	TIMER8 clock(except for EPRT series)
RCU_TIMER9	TIMER9 clock(except for EPRT series)
RCU_TIMER10	TIMER10 clock(except for EPRT series)
RCU_SHRTIMER	SHRTIMER clock(except for EPRT series)
RCU_USART5	USART5 clock
RCU_CMP	CMP clock(only for CL、E508 series)

### Enum rcu\_periph\_sleep\_enum

Table 3-704. Enum rcu\_periph\_sleep\_enum

enum name	Function description
RCU_SRAM_SLP	SRAM clock when sleep mode
RCU_FMC_SLP	FMC clock when sleep mode

### Enum rcu\_periph\_reset\_enum

Table 3-705. Enum rcu\_periph\_reset\_enum

enum name	Function description
RCU_USBHSRST	USBHS clock reset(only for CL、E508 series)

enum name	Function description
RCU_ENETRST	ENET clock reset(only for CL、EPRT、E508 series)
RCU_TMURST	TMU clock reset
RCU_SQPIRST	SQPI clock reset
RCU_TIMER1RST	TIMER1 clock reset
RCU_TIMER2RST	TIMER2 clock reset
RCU_TIMER3RST	TIMER3 clock reset
RCU_TIMER4RST	TIMER4 clock reset
RCU_TIMER5RST	TIMER5 clock reset
RCU_TIMER6RST	TIMER6 clock reset
RCU_TIMER11RST	TIMER11 clock reset(except for EPRT series)
RCU_TIMER12RST	TIMER12 clock reset(except for EPRT series)
RCU_TIMER13RST	TIMER13 clock reset(except for EPRT series)
RCU_WWDGTRST	WWDGT clock reset
RCU_SPI1RST	SPI1 clock reset
RCU_SPI2RST	SPI2 clock reset
RCU_USART1RST	USART1 clock reset
RCU_USART2RST	USART2 clock reset
RCU_UART3RST	UART3 clock reset
RCU_USART4RST	UART4 clock reset
RCU_I2C0RST	I2C0 clock reset
RCU_I2C1RST	I2C1 clock reset
RCU_USBDRST	USB clock reset(only for HD、EPRT series)
RCU_I2C2RST	I2C2 clock reset
RCU_CAN0RST	CAN0 clock reset(except for EPRT series)
RCU_CAN1RST	CAN1 clock reset(except for EPRT series)
RCU_BKPIRST	BKPI clock reset
RCU_PMURST	PMU clock reset
RCU_DACRST	DAC clock reset
RCU_CTCRST	CTC clock reset
RCU_CAN2RST	CAN2 clock reset(only for CL、E508 series)
RCU_AFRST	alternate function clock reset
RCU_GPIOARST	GPIOA clock reset
RCU_GPIOBRST	GPIOB clock reset
RCU_GPIOCRST	GPIOC clock reset
RCU_GPIODRST	GPIOD clock reset
RCU_GPIOERST	GPIOE clock reset
RCU_GPIOFRST	GPIOF clock reset
RCU_GPIOGRST	GPIOG clock reset
RCU_ADC0RST	ADC0 clock reset
RCU_ADC1RST	ADC1 clock reset
RCU_TIMER0RST	TIMER0 clock reset

enum name	Function description
RCU_SPI0RST	SPI0 clock reset
RCU_TIMER7RST	TIMER7 clock reset
RCU_USART0RST	USART0 clock reset
RCU_ADC2RST	ADC2 clock reset(except for CL series)
RCU_TIMER8RST	TIMER8 clock reset(except for EPRT series)
RCU_TIMER9RST	TIMER9 clock reset(except for EPRT series)
RCU_TIMER10RST	TIMER10 clock reset(except for EPRT series)
RCU_USART5RST	USART5 clock reset
RCU_SHRTIMERRST	HPTIEMR clock reset(except for EPRT series)
RCU_CMPRST	CMP clock reset(only for CL、E508 series)

### Enum rcu\_flag\_enum

**Table 3-706. Enum rcu\_flag\_enum**

enum name	Function description
RCU_FLAG_IRC8MST B	IRC8M stabilization flags
RCU_FLAG_HXTALS TB	HXTAL stabilization flags
RCU_FLAG_PLLSTB	PLL stabilization flags
RCU_FLAG_PLL1STB	PLL1 stabilization flags(only for CL、EPRT、E508 series)
RCU_FLAG_PLL2STB	PLL2 stabilization flags(only for CL、EPRT、E508 series)
RCU_FLAG_PLLUSB STB	PLLUSB stabilization flags(only for CL、E508 series)
RCU_FLAG_LXTALST B	LXTAL stabilization flags
RCU_FLAG_IRC40KS TB	IRC40K stabilization flags
RCU_FLAG_IRC48MS TB	IRC48M stabilization flags
RCU_FLAG_BORRST	BOR reset flag
RCU_FLAG_EPRST	external PIN reset flags
RCU_FLAG_PORRST	power reset flags
RCU_FLAG_SWRST	software reset flags
RCU_FLAG_FWDGTR ST	FWDGT reset flags
RCU_FLAG_WWDGT RST	WWDGT reset flags
RCU_FLAG_LPRST	low-power reset flags

## Enum rcu\_int\_flag\_enum

**Table 3-707. Enum rcu\_int\_flag\_enum**

enum name	Function description
RCU_INT_FLAG_IRC4 OKSTB	IRC40K stabilization interrupt flag
RCU_INT_FLAG_LXT ALSTB	LXTAL stabilization interrupt flag
RCU_INT_FLAG_IRC8 MSTB	IRC8M stabilization interrupt flag
RCU_INT_FLAG_HXT ALSTB	HXTAL stabilization interrupt flag
RCU_INT_FLAG_PLL STB	PLL stabilization interrupt flag
RCU_INT_FLAG_PLL 1STB	PLL1 stabilization interrupt flag(only for CL、EPRT、E508 series)
RCU_INT_FLAG_PLL 2STB	PLL2 stabilization interrupt flag(only for CL、EPRT、E508 series)
RCU_INT_FLAG_PLL USBSTB	PLLUSB stabilization interrupt flag(only for CL、E508 series)
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 3-708. Enum rcu\_int\_flag\_clear\_enum**

enum name	Function description
RCU_INT_FLAG_IRC4 OKSTB_CLR	IRC40K stabilization interrupt flags clear
RCU_INT_FLAG_LXT ALSTB_CLR	LXTAL stabilization interrupt flags clear
RCU_INT_FLAG_IRC8 MSTB_CLR	IRC8M stabilization interrupt flags clear
RCU_INT_FLAG_HXT ALSTB_CLR	HXTAL stabilization interrupt flags clear
RCU_INT_FLAG_PLL STB_CLR	PLL stabilization interrupt flags clear
RCU_INT_FLAG_PLL 1STB_CLR	PLL1 stabilization interrupt flags clear(only for CL、EPRT、E508 series)
RCU_INT_FLAG_PLL 2STB_CLR	PLL2 stabilization interrupt flags clear(only for CL、EPRT、E508 series)
RCU_INT_FLAG_PLL USBSTB_CLR	PLLUSB stabilization interrupt flags clear(only for CL、E508 series)

enum name	Function description
RCU_INT_FLAG_CKM_CLR	CKM interrupt flags clear
RCU_INT_FLAG_IRC48MSTB_CLR	internal 48 MHz RC oscillator stabilization interrupt clear

### Enum rcu\_int\_enum

**Table 3-709. Enum rcu\_int\_enum**

enum name	Function description
RCU_INT_IRC40KSTB	IRC40K stabilization interrupt
RCU_INT_LXTALSTB	LXTAL stabilization interrupt
RCU_INT_IRC8MSTB	IRC8M stabilization interrupt
RCU_INT_HXTALSTB	HXTAL stabilization interrupt
RCU_INT_PLLSTB	PLL stabilization interrupt
RCU_INT_PLL1STB	PLL1 stabilization interrupt(only for CL、EPRT、E508 series)
RCU_INT_PLL2STB	PLL2 stabilization interrupt(only for CL、EPRT、E508 series)
RCU_INT_PLLUSBSTB	PLLUSB stabilization interrupt(only for CL、E508 series)
RCU_INT_IRC48MSTB	internal 48 MHz RC oscillator stabilization interrupt

### Enum rcu\_osci\_type\_enum

**Table 3-710. Enum rcu\_osci\_type\_enum**

enum name	Function description
RCU_HXTAL	HXTAL
RCU_LXTAL	LXTAL
RCU_IRC8M	IRC8M
RCU_IRC48M	IRC48M
RCU_IRC40K	IRC40K
RCU_PLL_CK	PLL
RCU_PLL1_CK	PLL1(only for CL、EPRT、E508 series)
RCU_PLL2_CK	PLL2(only for CL、EPRT、E508 series)
RCU_PLLUSB_CK	PLLUSB(only for CL、E508 series)

### Enum rcu\_clock\_freq\_enum

**Table 3-711. Enum rcu\_clock\_freq\_enum**

enum name	Function description
CK_SYS	system clock
CK_AHB	AHB clock
CK_APB1	APB1 clock
CK_APB2	APB2 clock

enum name	Function description
CK_USART	USART5 clock

## rcu\_deinit

The description of rcu\_deinit is shown as below:

**Table 3-712. Function rcu\_deinit**

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

Example:

```
/* reset RCU */
rcu_deinit();
```

## rcu\_periph\_clock\_enable

The description of rcu\_periph\_clock\_enable is shown as below:

**Table 3-713. Function rcu\_periph\_clock\_enable**

<b>Function name</b>	rcu_periph_clock_enable
<b>Function prototype</b>	void rcu_periph_clock_enable(rcu_periph_enum periph);
<b>Function descriptions</b>	enable the peripherals clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-703. Enum rcu_periph_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the USART0 clock */
rcu_periph_clock_enable(RCU_USART0);
```

## rcu\_periph\_clock\_disable

The description of rcu\_periph\_clock\_disable is shown as below:

**Table 3-714. Function rcu\_periph\_clock\_disable**

<b>Function name</b>	rcu_periph_clock_disable
<b>Function prototype</b>	void rcu_periph_clock_disable(rcu_periph_enum periph);
<b>Function descriptions</b>	disable the peripherals clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-703. Enum rcu_periph_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the USART0 clock */
rcu_periph_clock_disable(RCU_USART0);
```

## rcu\_periph\_clock\_sleep\_enable

The description of rcu\_periph\_clock\_sleep\_enable is shown as below:

**Table 3-715. Function rcu\_periph\_clock\_sleep\_enable**

<b>Function name</b>	rcu_periph_clock_sleep_enable
<b>Function prototype</b>	void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);
<b>Function descriptions</b>	enable the peripherals clock when in sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-704. Enum rcu_periph_sleep_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the FMC clock when in sleep mode */
rcu_periph_clock_sleep_enable(RCU_FMC_SLP);
```

## rcu\_periph\_clock\_sleep\_disable

The description of rcu\_periph\_clock\_sleep\_disable is shown as below:

**Table 3-716. 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>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-704. Enum rcu_periph_sleep_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the FMC clock when in sleep mode */
rcu_periph_clock_sleep_disable(RCU_FMC_SLP);
```

## rcu\_periph\_reset\_enable

The description of rcu\_periph\_reset\_enable is shown as below:

**Table 3-717. Function rcu\_periph\_reset\_enable**

<b>Function name</b>	rcu_periph_reset_enable
<b>Function prototype</b>	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);
<b>Function descriptions</b>	enable the peripherals reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph_reset</b>	RCU peripherals reset, refer to <a href="#">Table 3-705. Enum rcu_periph_reset_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 reset */
rcu_periph_reset_enable(RCU_SPI0RST);
```



## rcu\_periph\_reset\_disable

The description of rcu\_periph\_reset\_disable is shown as below:

**Table 3-718. Function rcu\_periph\_reset\_disable**

<b>Function name</b>	rcu_periph_reset_disable
<b>Function prototype</b>	void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);
<b>Function descriptions</b>	disable the peripheral reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph_reset</b>	RCU peripherals reset, refer to <a href="#">Table 3-705. Enum rcu_periph_reset_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 reset */
rcu_periph_reset_disable(RCU_SPI0RST);
```

## rcu\_bkp\_reset\_enable

The description of rcu\_bkp\_reset\_enable is shown as below:

**Table 3-719. Function rcu\_bkp\_reset\_enable**

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

Example:

```
/* reset the BKP domain */
rcu_bkp_reset_enable();
```

## rcu\_bkp\_reset\_disable

The description of rcu\_bkp\_reset\_disable is shown as below:

**Table 3-720. 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 3-721. Function rcu\_system\_clock\_source\_config**

<b>Function name</b>	rcu_system_clock_source_config
<b>Function prototype</b>	void rcu_system_clock_source_config(uint32_t ck_sys);
<b>Function descriptions</b>	configure the system clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_sys</b>	system clock source select
<i>RCU_CKSYSSRC_IRC8M</i>	select CK_IRC8M as the CK_SYS source
<i>RCU_CKSYSSRC_HXTAL</i>	select CK_HXTAL as the CK_SYS source
<i>RCU_CKSYSSRC_PLL</i>	select CK_PLL as the CK_SYS source
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the CK_HXTAL as the CK_SYS source */
```

```
rcu_system_clock_source_config(RCU_CKSYS_SRC_HXTAL);
```

### rcu\_system\_clock\_source\_get

The description of rcu\_system\_clock\_source\_get is shown as below:

**Table 3-722. Function rcu\_system\_clock\_source\_get**

<b>Function name</b>	rcu_system_clock_source_get
<b>Function prototype</b>	uint32_t rcu_system_clock_source_get(void);
<b>Function descriptions</b>	get the system clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	RCU_SCSS_IRC8M / RCU_SCSS_HXTAL / RCU_SCSS_PLL

Example:

```
uint32_t temp_cksys_status;
```

```
/* get the CK_SYS source */
```

```
temp_cksys_status = rcu_system_clock_source_get();
```

### rcu\_ahb\_clock\_config

The description of rcu\_ahb\_clock\_config is shown as below:

**Table 3-723. Function rcu\_ahb\_clock\_config**

<b>Function name</b>	rcu_ahb_clock_config
<b>Function prototype</b>	void rcu_ahb_clock_config(uint32_t ck_ahb);
<b>Function descriptions</b>	configure the AHB clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
ck_ahb	AHB clock prescaler selection
RCU_AHB_CKSYS_DIVx	select CK_SYS / x as CK_AHB, (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 3-724. Function rcu\_apb1\_clock\_config**

<b>Function name</b>	rcu_apb1_clock_config
<b>Function prototype</b>	void rcu_apb1_clock_config(uint32_t ck_apb1);
<b>Function descriptions</b>	configure the APB1 clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_apb1</b>	APB1 clock prescaler selection
<i>RCU_APB1_CKAHB_D</i> <i>IV1</i>	select CK_AHB as CK_APB1
<i>RCU_APB1_CKAHB_D</i> <i>IV2</i>	select CK_AHB / 2 as CK_APB1
<i>RCU_APB1_CKAHB_D</i> <i>IV4</i>	select CK_AHB / 4 as CK_APB1
<i>RCU_APB1_CKAHB_D</i> <i>IV8</i>	select CK_AHB / 8 as CK_APB1
<i>RCU_APB1_CKAHB_D</i> <i>IV16</i>	select CK_AHB / 16 as CK_APB1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure CK_AHB / 16 as CK_APB1 */
rcu_apb1_clock_config(RCU_APB1_CKAHB_DIV16);
```

### rcu\_apb2\_clock\_config

The description of rcu\_apb2\_clock\_config is shown as below:

**Table 3-725. Function rcu\_apb2\_clock\_config**

<b>Function name</b>	rcu_apb2_clock_config
<b>Function prototype</b>	void rcu_apb2_clock_config(uint32_t ck_apb2);
<b>Function descriptions</b>	configure the APB2 clock prescaler selection

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_apb2</b>	APB2 clock prescaler selection
<i>RCU_APB2_CKAHB_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 3-726. Function rcu\_ckout0\_config**

<b>Function name</b>	rcu_ckout0_config
<b>Function prototype</b>	void rcu_ckout0_config(uint32_t ckout0_src);
<b>Function descriptions</b>	configure the CK_OUT0 clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ckout0_src</b>	CK_OUT0 clock source selection
<i>RCU_CKOUT0SRC_N</i> <i>ONE</i>	no clock selected
<i>RCU_CKOUT0SRC_C</i> <i>KSYS</i>	system clock selected
<i>RCU_CKOUT0SRC_IR</i> <i>C8M</i>	high speed 8M internal oscillator clock selected
<i>RCU_CKOUT0SRC_H</i> <i>XTAL</i>	HXTAL selected

<i>RCU_CKOUT0SRC_C</i> <i>KPLL_DIV2</i>	CK_PLL / 2 selected
<i>RCU_CKOUT0SRC_C</i> <i>KPLL1</i>	CK_PLL1 selected (only available for CL and EPRT series)
<i>RCU_CKOUT0SRC_C</i> <i>KPLL2_DIV2</i>	CK_PLL2 / 2 selected (only available for CL and EPRT series)
<i>RCU_CKOUT0SRC_C</i> <i>KPLL2</i>	EXT1 selected (only available for CL and EPRT series)
<i>RCU_CKOUT0SRC_E</i> <i>XT1</i>	CK_PLL2 clock selected (only available for CL and EPRT series)
<i>RCU_CKOUT0SRC_C</i> <i>KIRC48M</i>	CK_IRC48M clock selected (only available for CL and EPRT series)
<i>RCU_CKOUT0SRC_C</i> <i>KIRC48M_DIV8</i>	CK_IRC48M / 8 clock selected (only available for CL and EPRT series)
<i>RCU_CKOUT0SRC_C</i> <i>KPLLUSB_DIV32</i>	CK_PLLUSB / 32 clock selected (only available for CL series)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the HXTAL as CK_OUT0 clock source */
```

```
rcu_ckout0_config(RCU_CKOUT0SRC_HXTAL);
```

### rcu\_pll\_config

The description of rcu\_pll\_config is shown as below:

**Table 3-727. Function rcu\_pll\_config**

<b>Function name</b>	rcu_pll_config
<b>Function prototype</b>	void rcu_pll_config(uint32_t pll_src, uint32_t pll_mul);
<b>Function descriptions</b>	configure the main PLL clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pll_src</b>	PLL clock source selection
<i>RCU_PLLSRC_IRC8M</i> <i>_DIV2</i>	IRC8M/2 clock is selected as source clock of PLL
<i>RCU_PLLSRC_HXTAL</i> <i>_IRC48M</i>	HXTAL or IRC48M is selected as source clock of PLL
<b>Input parameter{in}</b>	
<b>pll_mul</b>	PLL clock multiplication factor

<i>RCU_PLL_MULx</i>	PLL clock * x (HD series x = 2..63, CL series x = 2..14, 16..64, 6.5)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL */
```

```
rcu_pll_config(RCU_PLLSRC_HXTAL, RCU_PLL_MUL10);
```

### rcu\_pllpresel\_config

The description of rcu\_pllpresel\_config is shown as below:

**Table 3-728. Function rcu\_pllpresel\_config**

<b>Function name</b>	rcu_pllpresel_config
<b>Function prototype</b>	void rcu_pllpresel_config(uint32_t pll_presel);
<b>Function descriptions</b>	configure the PLL clock source preselection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pll_presel</b>	PLL clock source preselection
<i>RCU_PLLPRESRC_HXTAL</i>	HXTAL selected as PREDV0 input source clock
<i>RCU_PLLPRESRC_IRC48M</i>	CK_PLL selected as PREDV0 input source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL clock source preselection */
```

```
rcu_pllpresel_config (RCU_PLLPRESRC_HXTAL);
```

### rcu\_predv0\_config (HD series)

The description of rcu\_predv0\_config is shown as below:

**Table 3-729. Function rcu\_predv0\_config**

<b>Function name</b>	rcu_predv0_config
<b>Function prototype</b>	void rcu_predv0_config(uint32_t predv0_div);
<b>Function descriptions</b>	configure the PREDV0 division factor
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>predv0_div</b>	PREDV0 division factor
<i>RCU_PREDV0_DIVx</i>	PREDV0 input source clock is divided x (x = 1, 2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PREDV0 division factor */
```

```
rcu_predv0_config(RCU_PREDV0_DIV1);
```

## rcu\_predv0\_config (CL、EPRT、E508 series)

The description of rcu\_predv0\_config is shown as below:

**Table 3-730. Function rcu\_predv0\_config**

<b>Function name</b>	rcu_predv0_config
<b>Function prototype</b>	void rcu_predv0_config(uint32_t predv0_source, uint32_t predv0_div);
<b>Function descriptions</b>	configure the PREDV0 division factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>predv0_source</b>	PREDV0 input clock source selection
<i>RCU_PREDV0SRC_HXTAL_IRC48M</i>	HXTAL or IRC48M selected as PREDV0 input source clock
<i>RCU_PREDV0SRC_CKPLL1</i>	CK_PLL1 selected as PREDV0 input source clock
<b>Input parameter{in}</b>	
<b>predv0_div</b>	PREDV0 division factor
<i>RCU_PREDV0_DIVx</i>	PREDV0 input source clock is divided x (x = 1..16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PREDV0 division factor */
```

```
rcu_predv0_config(RCU_PREDV0SRC_HXTAL_IRC48M, RCU_PREDV0_DIV);
```



## rcu\_predv1\_config

The description of rcu\_predv1\_config is shown as below:

**Table 3-731. Function rcu\_predv1\_config**

<b>Function name</b>	rcu_predv1_config
<b>Function prototype</b>	void rcu_predv1_config(uint32_t predv1_div);
<b>Function descriptions</b>	configure the PREDV1 division factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>predv1_div</b>	PREDV1 division factor
<i>RCU_PREDV1_DIVx</i>	PREDV1 input source clock is divided x (x = 1..16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PREDV1 division factor */
rcu_predv1_config(RCU_PREDV1_DIV8);
```

## rcu\_pll1\_config

The description of rcu\_pll1\_config is shown as below:

**Table 3-732. Function rcu\_pll1\_config**

<b>Function name</b>	rcu_pll1_config
<b>Function prototype</b>	void rcu_pll1_config(uint32_t pll_mul);
<b>Function descriptions</b>	configure the PLL1 clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pll_mul</b>	PLL clock multiplication factor
<i>RCU_PLL1_MULx</i>	PLL1 clock * x, (x = 8..14, 16, 20)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL1 clock */
rcu_pll1_config(RCU_PLL1_MUL8);
```

## rcu\_pll2\_config

The description of rcu\_pll2\_config is shown as below:

**Table 3-733. Function rcu\_pll2\_config**

<b>Function name</b>	rcu_pll2_config
<b>Function prototype</b>	void rcu_pll2_config(uint32_t pll_mul);
<b>Function descriptions</b>	configure the PLL2 clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pll_mul</b>	PLL clock multiplication factor
<i>RCU_PLL2_MULx</i>	PLL2 clock * x, (x = 8..14, 16, 20, 18..32, 40, 34..64, 80)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL2 clock */
```

```
rcu_pll2_config(RCU_PLL2_MUL8);
```

## rcu\_pllusbpresel\_config

The description of rcu\_pllusbpresel\_config is shown as below:

**Table 3-734. Function rcu\_pllusbpresel\_config**

<b>Function name</b>	rcu_pllusbpresel_config
<b>Function prototype</b>	void rcu_pllusbpresel_config(uint32_t pllusb_presel);
<b>Function descriptions</b>	configure the PLLUSB clock source preselection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pllusb_presel</b>	PLLUSB clock source preselection
<i>RCU_PLLUSBPRESR</i> <i>C_HXTAL</i>	HXTAL selected as PLLUSB source clock
<i>RCU_PLLUSBPRESR</i> <i>C_IRC48M</i>	IRC48M clock selected as PLLUSB source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLLUSB clock source preselection */
```

```
rcu_pllusbpresel_config(RCU_PLLUSBPRESRC_HXTAL);
```

## rcu\_pllusbpredv\_config

The description of rcu\_pllusbpredv\_config is shown as below:

**Table 3-735. Function rcu\_pllusbpredv\_config**

<b>Function name</b>	rcu_pllusbpredv_config
<b>Function prototype</b>	void rcu_pllusbpredv_config(uint32_t pllusbpredv_source, uint32_t pllusbpredv_div);
<b>Function descriptions</b>	configure the PLLUSBPREDV division factor and clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pllusbpredv_source</b>	PLLUSBPREDV input clock source selection
<i>RCU_PLLUSBPREDVSR</i> <i>RC_HXTAL_IRC48M</i>	HXTAL or IRC48M selected as PLLUSBPREDV input source clock
<i>RCU_PLLUSBPREDVSR</i> <i>RC_CKPLL1</i>	CK_PLL1 selected as PLLUSBPREDV input source clock
<b>Input parameter{in}</b>	
<b>pllusbpredv_div</b>	PLLUSBPREDV division factor
<i>RCU_PLLUSBPREDV_</i> <i>DIVx</i>	PLLUSBPREDV input source clock divided by x,( x = 1..15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLLUSBPREDV division factor and clock source */
```

```
rcu_pllusbpredv_config(RCU_PLLUSBPREDVSR_HXTAL_IRC48M, RCU_PLLUSBPREDV_DIV15);
```

## rcu\_pllusb\_config

The description of rcu\_pllusb\_config is shown as below:

**Table 3-736. Function rcu\_pllusb\_config**

<b>Function name</b>	rcu_pllusb_config
<b>Function prototype</b>	void rcu_pllusb_config(uint32_t pllusb_mul);
<b>Function descriptions</b>	configure the PLLUSB clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>pllusb_mul</b>	PLLUSB clock multiplication factor
<i>RCU_PLLUSB_MULx</i>	PLLUSB source clock multiply by x, (x = 16, 17..127)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLLUSB clock */
```

```
rcu_pllusb_config(RCU_PLLUSB_MUL16);
```

### rcu\_adc\_clock\_config

The description of rcu\_adc\_clock\_config is shown as below:

**Table 3-737. Function rcu\_adc\_clock\_config**

<b>Function name</b>	rcu_adc_clock_config
<b>Function prototype</b>	void rcu_adc_clock_config(uint32_t adc_psc);
<b>Function descriptions</b>	configure the ADC prescaler factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_psc</b>	ADC prescaler factor
<i>RCU_CKADC_CKAPB2_DIV2</i>	ADC prescaler select CK_APB2 / 2
<i>RCU_CKADC_CKAPB2_DIV4</i>	ADC prescaler select CK_APB2 / 4
<i>RCU_CKADC_CKAPB2_DIV6</i>	ADC prescaler select CK_APB2 / 6
<i>RCU_CKADC_CKAPB2_DIV8</i>	ADC prescaler select CK_APB2 / 8
<i>RCU_CKADC_CKAPB2_DIV12</i>	ADC prescaler select CK_APB2 / 12
<i>RCU_CKADC_CKAPB2_DIV16</i>	ADC prescaler select CK_APB2 / 16
<i>RCU_CKADC_CKAHB_DIV5</i>	ADC prescaler select CK_AHB / 5
<i>RCU_CKADC_CKAHB_DIV6</i>	ADC prescaler select CK_AHB / 6
<i>RCU_CKADC_CKAHB_DIV10</i>	ADC prescaler select CK_AHB / 10
<i>RCU_CKADC_CKAHB_DIV20</i>	ADC prescaler select CK_AHB / 20

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the ADC prescaler factor */
rcu_adc_clock_config(RCU_CKADC_CKAPB2_DIV8);
```

### rcu\_usb\_clock\_config

The description of rcu\_usb\_clock\_config is shown as below:

**Table 3-738. Function rcu\_usb\_clock\_config**

Function name	rcu_usb_clock_config
Function prototype	void rcu_usb_clock_config(uint32_t usb_psc);
Function descriptions	configure the USB prescaler factor
Precondition	-
The called functions	-
Input parameter{in}	
usb_psc	USB prescaler factor
<i>RCU_CKUSB_CKPLL_DIV1_5</i>	USBD / USBHS prescaler select CK_PLL / 1.5
<i>RCU_CKUSB_CKPLL_DIV1</i>	USBD / USBHS prescaler select CK_PLL / 1
<i>RCU_CKUSB_CKPLL_DIV2_5</i>	USBD / USBHS prescaler select CK_PLL / 2.5
<i>RCU_CKUSB_CKPLL_DIV2</i>	USBD / USBHS prescaler select CK_PLL / 2
<i>RCU_CKUSB_CKPLL_DIV3</i>	USBD / USBHS prescaler select CK_PLL / 3
<i>RCU_CKUSB_CKPLL_DIV3_5</i>	USBD / USBHS prescaler select CK_PLL / 3.5
<i>RCU_CKUSB_CKPLL_DIV4</i>	USBD / USBHS prescaler select CK_PLL / 4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the USB prescaler factor */
rcu_usb_clock_config(RCU_CKUSB_CKPLL_DIV2_5);
```

## rcu\_rtc\_clock\_config

The description of rcu\_rtc\_clock\_config is shown as below:

**Table 3-739. 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_IRC40K</i>	select CK_IRC40K as RTC source clock
<i>RCU_RTCSRC_HXTAL_DIV_128</i>	select CK_HXTAL / 128 as RTC source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the RTC clock source selection */
rcu_rtc_clock_config(RCU_RTCSRC_IRC40K);
```

## rcu\_shrtimer\_clock\_config

The description of rcu\_shrtimer\_clock\_config is shown as below:

**Table 3-740. Function rcu\_shrtimer\_clock\_config**

<b>Function name</b>	rcu_shrtimer_clock_config
<b>Function prototype</b>	void rcu_shrtimer_clock_config (uint32_t shrtimer_clock_source);
<b>Function descriptions</b>	configure the SHRTIMER clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>shrtimer_clock_source</b>	SHRTIMER clock source selection
<i>RCU_SHRTIMERSRC_CKAPB2</i>	APB2 clock selected as SHRTIMER source clock
<i>RCU_SHRTIMERSRC_CKSYS</i>	system clock selected as SHRTIMER source clock

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the SHRTIMER clock source selection */
```

```
rcu_shrtimer_clock_config(RCU_SHRTIMERSRC_CKAPB2);
```

## rcu\_usart5\_clock\_config

The description of rcu\_usart5\_clock\_config is shown as below:

**Table 3-741. Function rcu\_usart5\_clock\_config**

Function name	rcu_usart5_clock_config
Function prototype	void rcu_usart5_clock_config(uint32_t usart5_clock_source);
Function descriptions	configure the USART5 clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
usart5_clock_source	USART5 clock source selection
RCU_USART5SRC_CKAPB2	APB2 clock selected as USART5 source clock
RCU_USART5SRC_CKSYS	system clock selected as USART5 source clock
RCU_USART5SRC_LXTAL	LXTAL clock selected as USART5 source clock
RCU_USART5SRC_IRC8M	IRC8M clock selected as USART5 source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the USART5 clock source selection */
```

```
rcu_usart5_clock_config(RCU_USART5SRC_CKAPB2);
```

## rcu\_i2c2\_clock\_config

The description of rcu\_i2c2\_clock\_config is shown as below:

**Table 3-742. Function rcu\_i2c2\_clock\_config**

Function name	rcu_i2c2_clock_config
---------------	-----------------------

<b>Function prototype</b>	void rcu_i2c2_clock_config(uint32_t i2c2_clock_source);
<b>Function descriptions</b>	configure the I2C2 clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c2_clock_source</b>	I2C2 clock source selection
<i>RCU_I2C2SRC_CKAPB1</i>	APB1 clock selected as I2C2 source clock
<i>RCU_I2C2SRC_CKSYS</i>	System clock selected as I2C2 source clock
<i>RCU_I2C2SRC_SRC_CKIRC8M</i>	CK_IRC8M clock selected as I2C2 source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the I2C2 clock source selection */
```

```
rcu_i2c2_clock_config(RCU_I2C2SRC_CKAPB1);
```

## rcu\_ck48m\_clock\_config

The description of rcu\_ck48m\_clock\_config is shown as below:

**Table 3-743. Function rcu\_ck48m\_clock\_config**

<b>Function name</b>	rcu_ck48m_clock_config
<b>Function prototype</b>	void rcu_ck48m_clock_config(uint32_t ck48m_clock_source);
<b>Function descriptions</b>	configure the CK48M clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck48m_clock_source</b>	CK48M clock source selection
<i>RCU_CK48MSRC_CKPLL</i>	CK_PLL selected as CK48M source clock
<i>RCU_CK48MSRC_CKIRC48M</i>	CK_IRC48M selected as CK48M source clock
<i>RCU_CK48MSRC_CKPLLUSB</i>	CKPLLUSB selected as CK48M source clock
<i>RCU_CK48MSRC_CKPLL2</i>	CKPLL2 selected as CK48M source clock
<b>Output parameter{out}</b>	
-	-



Return value	
-	-

Example:

```
/* configure the CK48M clock source selection */
```

```
rcu_ck48m_clock_config(RCU_CK48MSRC_IRC48M);
```

## rcu\_i2s1\_clock\_config

The description of rcu\_i2s1\_clock\_config is shown as below:

**Table 3-744. Function rcu\_i2s1\_clock\_config**

Function name	rcu_i2s1_clock_config
Function prototype	void rcu_i2s1_clock_config(uint32_t i2s_clock_source);
Function descriptions	configure the I2S1 clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
i2s_clock_source	I2S clock source selection
RCU_I2S1SRC_CKSYS	system clock selected as I2S1 source clock
RCU_I2S1SRC_CKPLL2_MUL2	CK_PLL2 * 2 selected as I2S1 source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the I2S1 clock source selection */
```

```
rcu_i2s1_clock_config(RCU_I2S1SRC_CKPLL2_MUL2);
```

## rcu\_i2s2\_clock\_config

The description of rcu\_i2s2\_clock\_config is shown as below:

**Table 3-745. Function rcu\_i2s2\_clock\_config**

Function name	rcu_i2s2_clock_config
Function prototype	void rcu_i2s2_clock_config(uint32_t i2s_clock_source);
Function descriptions	configure the I2S2 clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
i2s_clock_source	I2S clock source selection

<i>RCU_I2S2SRC_CKSYS</i>	system clock selected as I2S2 source clock
<i>RCU_I2S2SRC_CKPLL2_MUL2</i>	CK_PLL2 * 2 selected as I2S2 source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the I2S2 clock source selection */
```

```
rcu_i2s2_clock_config(RCU_I2S2SRC_CKPLL2_MUL2);
```

The description of rcu\_usbhsel\_config is shown as below:

**Table 3-746. Function rcu\_usbhsel\_config**

<b>Function name</b>	rcu_usbhsel_config
<b>Function prototype</b>	void rcu_usbhsel_config(uint32_t usbhsel_clock_source);
<b>Function descriptions</b>	configure the USBHSSEL source clock selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usbhsel_clock_source</b>	USBHSSEL clock source selection
<i>RCU_USBHSSRC_48M</i>	48M clock selected as USBHS source clock
<i>RCU_CK48MSRC_60M</i>	60M clock selected as USBHS source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the USBHSSEL source clock selection */
```

```
rcu_usbhsel_config(RCU_USBHSSRC_48M);
```

### rcu\_usbdv\_config

The description of rcu\_usbdv\_config is shown as below:

**Table 3-747. Function rcu\_usbdv\_config**

<b>Function name</b>	rcu_usbdv_config
<b>Function prototype</b>	void rcu_usbdv_config(uint32_t usbhs_dv);
<b>Function descriptions</b>	configure the USBHSDV division factor

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usbhs_dv</b>	USBHSDV division factor
<i>RCU_USBHSDV_DIV2</i>	USBHSDV input source clock divided by 2
<i>RCU_USBHSDV_DIV4</i>	USBHSDV input source clock divided by 4
<i>RCU_USBHSDV_DIV6</i>	USBHSDV input source clock divided by 6
<i>RCU_USBHSDV_DIV8</i>	USBHSDV input source clock divided by 8
<i>RCU_USBHSDV_DIV10</i>	USBHSDV input source clock divided by 10
<i>RCU_USBHSDV_DIV12</i>	USBHSDV input source clock divided by 12
<i>RCU_USBHSDV_DIV14</i>	USBHSDV input source clock divided by 14
<i>RCU_USBHSDV_DIV16</i>	USBHSDV input source clock divided by 16
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the USBHSDV division factor */
rcu_usbdv_config(RCU_USBHSDV_DIV16);
```

## rcu\_lxtal\_drive\_capability\_config

The description of rcu\_lxtal\_drive\_capability\_config is shown as below:

**Table 3-748. Function rcu\_lxtal\_drive\_capability\_config**

<b>Function name</b>	rcu_lxtal_drive_capability_config
<b>Function prototype</b>	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);
<b>Function descriptions</b>	configure the LXTAL drive capability
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lxtal_dricap</b>	drive capability of LXTAL
<i>RCU_LXTAL_LOWDRI</i>	lower driving capability
<i>RCU_LXTAL_MED_LOWDRI</i>	medium low driving capability
<i>RCU_LXTAL_MED_HI_GHDRI</i>	medium high driving capability
<i>RCU_LXTAL_HIGHDRI</i>	higher driving capability

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 3-749. 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	-
Input parameter{in}	
osci	oscillator types, refer to <a href="#">Table 3-710. 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 3-750. 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 3-710. 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 3-751. Function rcu\_osci\_off**

Function name	rcu_osci_off
Function prototype	void rcu_osci_off(rcu_osci_type_enum osci);
Function descriptions	turn off the oscillator
Precondition	-
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to <a href="#">Table 3-710. Enum rcu_osci_type_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn off the high speed crystal oscillator */
```

```
rcu_osci_off(RCU_HXTAL);
```

### rcu\_osci\_bypass\_mode\_enable

The description of rcu\_osci\_bypass\_mode\_enable is shown as below:

**Table 3-752. Function rcu\_osci\_bypass\_mode\_enable**

Function name	rcu_osci_bypass_mode_enable
Function prototype	void rcu_osci_bypass_mode_enable(rcu_osci_type_enum osci);
Function descriptions	enable the oscillator bypass mode
Precondition	HXTALEN or LXTALEN must be reset before it
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to <a href="#">Table 3-710. 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 3-753. 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 3-710. 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\_irc8m\_adjust\_value\_set

The description of rcu\_irc8m\_adjust\_value\_set is shown as below:

**Table 3-754. Function rcu\_irc8m\_adjust\_value\_set**

Function name	rcu_irc8m_adjust_value_set
Function prototype	void rcu_irc8m_adjust_value_set(uint32_t irc8m_adjval);
Function descriptions	set the IRC8M adjust value
Precondition	-
The called functions	-
Input parameter{in}	
irc8m_adjval	IRC8M adjust value, must be between 0 and 0x1F

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the IRC8M adjust value */
rcu_irc8m_adjust_value_set(0x10);
```

## rcu\_hxtal\_clock\_monitor\_enable

The description of rcu\_hxtal\_clock\_monitor\_enable is shown as below:

**Table 3-755. Function rcu\_hxtal\_clock\_monitor\_enable**

Function name	rcu_hxtal_clock_monitor_enable
Function prototype	void rcu_hxtal_clock_monitor_enable(void);
Function descriptions	enable the HXTAL clock monitor
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

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 3-756. Function rcu\_hxtal\_clock\_monitor\_disable**

Function name	rcu_hxtal_clock_monitor_disable
Function prototype	void rcu_hxtal_clock_monitor_disable(void);
Function descriptions	disable the HXTAL clock monitor
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* disable the HXTAL clock monitor */
```

```
rcu_hxtal_clock_monitor_disable();
```

## rcu\_deepsleep\_voltage\_set

The description of rcu\_deepsleep\_voltage\_set is shown as below:

**Table 3-757. 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}	
dsvol	deep sleep mode voltage
RCU_DEEPSLEEP_V_1_0	the core voltage is 1.0V in deep-sleep mode
RCU_DEEPSLEEP_V_0_9	the core voltage is 0.9V in deep-sleep mode
RCU_DEEPSLEEP_V_0_8	the core voltage is 0.8V in deep-sleep mode
RCU_DEEPSLEEP_V_0_7	the core voltage is 0.7V in deep-sleep mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the deep-sleep mode voltage */
```

```
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1_0);
```

## rcu\_clock\_freq\_get

The description of rcu\_clock\_freq\_get is shown as below:

**Table 3-758. Function rcu\_clock\_freq\_get**

Function name	rcu_clock_freq_get
Function prototype	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);
Function descriptions	get the system clock, bus clock frequency



<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>clock</b>	Clock frequency, refers to <a href="#">Table 3-711. Enum rcu_clock_freq_enum</a>
<i>CK_SYS</i>	system clock frequency
<i>CK_AHB</i>	AHB clock frequency
<i>CK_APB1</i>	APB1 clock frequency
<i>CK_APB2</i>	APB2 clock frequency
<i>CK_USART</i>	USART5 clock frequency
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ck_freq</b>	clock frequency of system, AHB, APB1, APB2, USART5

Example:

```
uint32_t temp_freq;

/* get the system clock frequency */

temp_freq = rcu_clock_freq_get(CK_SYS);
```

## rcu\_flag\_get

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

**Table 3-759. 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 3-706. 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 3-760. 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 3-761. Function rcu\_interrupt\_flag\_get**

<b>Function name</b>	rcu_interrupt_flag_get
<b>Function prototype</b>	FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag);
<b>Function descriptions</b>	get the clock stabilization interrupt and ckm flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt and ckm flags, refer to <a href="#">Table 3-707. Enum rcu_int_flag_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the clock stabilization interrupt flag */
if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){
}
```

## rcu\_interrupt\_flag\_clear

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

**Table 3-762. Function rcu\_interrupt\_flag\_clear**

<b>Function name</b>	rcu_interrupt_flag_clear
<b>Function prototype</b>	void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag);
<b>Function descriptions</b>	clear the interrupt flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	clock stabilization and stuck interrupt flags clear, refer to <a href="#">Table 3-708. Enum rcu_int_flag_clear_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the interrupt HXTAL stabilization interrupt flag */
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

## rcu\_interrupt\_enable

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

**Table 3-763. Function rcu\_interrupt\_enable**

<b>Function name</b>	rcu_interrupt_enable
<b>Function prototype</b>	void rcu_interrupt_enable(rcu_int_enum interrupt);
<b>Function descriptions</b>	enable the stabilization interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	clock stabilization interrupt, refer to <a href="#">Table 3-709. 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 3-764. Function rcu\_interrupt\_disable**

<b>Function name</b>	rcu_interrupt_disable
<b>Function prototype</b>	void rcu_interrupt_disable(rcu_int_enum interrupt);
<b>Function descriptions</b>	disable the stabilization interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	clock stabilization interrupt, refer to <a href="#">Table 3-709. Enum rcu_int_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the HXTAL stabilization interrupt */
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

## 3.22. 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 3-765. RTC Registers**

Registers	Descriptions
RTC_INTEN	Interrupt enable register
RTC_CTL	Control register
RTC_PSCH	Prescaler high register
RTC_PSCL	Prescaler low register
RTC_DIVH	Divider high register
RTC_DIVL	Divider low register
RTC_CNTH	counter high register
RTC_CNTL	counter low register
RTC_ALRMH	Alarm high register

Registers	Descriptions
RTC_ALRML	Alarm low register

### 3.22.2. Descriptions of Peripheral functions

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

**Table 3-766. RTC firmware function**

Function name	Function description
rtc_configuration_mode_enter	enter RTC configuration mode
rtc_configuration_mode_exit	exit RTC configuration mode
rtc_lwoff_wait	wait RTC last write operation finished flag set
rtc_register_sync_wait	wait RTC registers synchronized flag set
rtc_counter_get	get RTC counter value
rtc_counter_set	set RTC counter value
rtc_prescaler_set	set RTC prescaler value
rtc_alarm_config	set RTC alarm value
rtc_divider_get	get RTC divider value
rtc_interrupt_enable	enable RTC interrupt
rtc_interrupt_disable	disable RTC interrupt
rtc_flag_get	get RTC flag status
rtc_flag_clear	clear RTC flag status

#### rtc\_configuration\_mode\_enter

The description of rtc\_configuration\_mode\_enter is shown as below:

**Table 3-767. Function rtc\_configuration\_mode\_enter**

Function name	rtc_configuration_mode_enter
Function prototype	void rtc_configuration_mode_enter(void);
Function descriptions	enter RTC configuration mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enter RTC configuration mode */
rtc_configuration_mode_enter( );
```

## rtc\_configuration\_mode\_exit

The description of rtc\_configuration\_mode\_exit is shown as below:

**Table 3-768. Function rtc\_configuration\_mode\_exit**

<b>Function name</b>	rtc_configuration_mode_exit
<b>Function prototype</b>	void rtc_configuration_mode_exit(void);
<b>Function descriptions</b>	exit RTC configuration mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* exit RTC configuration mode */
rtc_configuration_mode_exit ( );
```

## rtc\_lwoff\_wait

The description of rtc\_lwoff\_wait is shown as below:

**Table 3-769. Function rtc\_lwoff\_wait**

<b>Function name</b>	rtc_lwoff_wait
<b>Function prototype</b>	void rtc_lwoff_wait(void);
<b>Function descriptions</b>	wait RTC last write operation finished flag set
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */
rtc_lwoff_wait( );

/* enable the RTC second interrupt */
rtc_interrupt_enable(RTC_INT_SECOND);
```

## rtc\_register\_sync\_wait

The description of rtc\_register\_sync\_wait is shown as below:

**Table 3-770. Function rtc\_register\_sync\_wait**

<b>Function name</b>	rtc_register_sync_wait
<b>Function prototype</b>	void rtc_register_sync_wait(void);
<b>Function descriptions</b>	wait RTC registers synchronized flag set
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* wait for RTC registers synchronization */
rtc_register_sync_wait( );
```

## rtc\_counter\_get

The description of rtc\_counter\_get is shown as below:

**Table 3-771. Function rtc\_counter\_get**

<b>Function name</b>	rtc_counter_get
<b>Function prototype</b>	uint32_t rtc_counter_get(void);
<b>Function descriptions</b>	get RTC counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	the value of RTC counter

Example:

```
/* get the counter value */
uint32_t rtc_counter_value;
rtc_counter_value = rtc_counter_get( );
```

## rtc\_counter\_set

The description of rtc\_counter\_set is shown as below:

**Table 3-772. Function rtc\_counter\_set**

<b>Function name</b>	rtc_counter_set
<b>Function prototype</b>	void rtc_counter_set(uint32_t cnt);
<b>Function descriptions</b>	set RTC counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cnt</b>	RTC counter value (0-0xFFFF FFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */
rtc_lwoff_wait( );

/* set counter value to 0xFFFF */
rtc_counter_set (0xFFFF);
```

## rtc\_prescaler\_set

The description of rtc\_prescaler\_set is shown as below:

**Table 3-773. Function rtc\_prescaler\_set**

<b>Function name</b>	rtc_interrupt_rtc_prescaler_set
<b>Function prototype</b>	void rtc_prescaler_set(uint32_t psc);
<b>Function descriptions</b>	set RTC prescaler value
<b>Precondition</b>	before using this function, you must call rtc_lwoff_wait( ) function (wait until LWOFF flag is set)
<b>The called functions</b>	rtc_configuration_mode_enter / rtc_configuration_mode_exit
<b>Input parameter{in}</b>	
<b>psc</b>	RTC prescaler value (0-0x000F FFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */
```



```
rtc_lwoff_wait( );

/* set RTC prescaler value to 0x7FFFF */

rtc_prescaler_set (0x7FFFF);
```

## rtc\_alarm\_config

The description of rtc\_alarm\_config is shown as below:

**Table 3-774. Function rtc\_alarm\_config**

<b>Function name</b>	rtc_alarm_config
<b>Function prototype</b>	void rtc_alarm_config(uint32_t alarm);
<b>Function descriptions</b>	set RTC alarm value
<b>Precondition</b>	before using this function, you must call rtc_lwoff_wait ( ) function (wait until LWOFF flag is set). -
<b>The called functions</b>	rtc_configuration_mode_enter / rtc_configuration_mode_exit
<b>Input parameter{in}</b>	
<b>alarm</b>	RTC alarm value (0-0xFFFF FFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */

rtc_lwoff_wait( );

/* set alarm value to 0xFFFF */

rtc_alarm_config (0xFFFF);
```

## rtc\_divider\_get

The description of rtc\_divider\_get is shown as below:

**Table 3-775. Function rtc\_divider\_get**

<b>Function name</b>	rtc_divider_get
<b>Function prototype</b>	uint32_t rtc_divider_get(void);
<b>Function descriptions</b>	get RTC divider value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

Return value	
uint32_t	the value of RTC divider

Example:

```
/* get the current RTC divider value */
```

```
uint32_t rtc_divider_value;
```

```
rtc_divider_value = rtc_divider_get ( );
```

## rtc\_interrupt\_enable

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

**Table 3-776. 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 RTC interrupt
<b>Precondition</b>	before using this function, you must call rtc_lwoff_wait ( ) function (wait until LWOFF flag is set).
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which RTC interrupt to enable
RTC_INT_SECOND	second interrupt
RTC_INT_ALARM	alarm interrupt
RTC_INT_OVERFLOW	overflow interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */
```

```
rtc_lwoff_wait( );
```

```
/* enable the RTC second interrupt */
```

```
rtc_interrupt_enable(RTC_INT_SECOND);
```

## rtc\_interrupt\_disable

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

**Table 3-777. Function rtc\_interrupt\_disable**

<b>Function name</b>	rtc_interrupt_disable
<b>Function prototype</b>	void rtc_interrupt_disable(uint32_t interrupt);

<b>Function descriptions</b>	disable RTC interrupt
<b>Precondition</b>	before using this function, you must call rtc_lwoff_wait ( ) function (wait until LWOFF flag is set).
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which RTC interrupt to disable
<i>RTC_INT_SECOND</i>	second interrupt
<i>RTC_INT_ALARM</i>	alarm interrupt
<i>RTC_INT_OVERFLOW</i>	overflow interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */
```

```
rtc_lwoff_wait( );
```

```
/* disable the RTC second interrupt */
```

```
rtc_interrupt_disable(RTC_INT_SECOND);
```

## rtc\_flag\_get

The description of rtc\_flag\_getrtc\_interrupt\_enable is shown as below:

**Table 3-778. Function rtc\_flag\_get**

<b>Function name</b>	rtc_flag_get
<b>Function prototype</b>	FlagStatus rtc_flag_get(uint32_t flag);
<b>Function descriptions</b>	get RTC flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which RTC flag status to get
<i>RTC_FLAG_SECOND</i>	second interrupt flag
<i>RTC_FLAG_ALARM</i>	alarm interrupt flag
<i>RTC_FLAG_OVERFLOW</i>	overflow interrupt flag
<i>RTC_FLAG_RSYN</i>	registers synchronized flag
<i>RTC_FLAG_LWOF</i>	last write operation finished flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the RTC overflow interrupt status */

FlagStatus alarm_status;

alarm_status = rtc_flag_get (RTC_FLAG_ALARM);
```

## rtc\_flag\_clear

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

**Table 3-779. Function rtc\_flag\_clear**

<b>Function name</b>	rtc_flag_clear
<b>Function prototype</b>	void rtc_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear RTC flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which RTC flag status to clear
<i>RTC_FLAG_SECOND</i>	second interrupt flag
<i>RTC_FLAG_ALARM</i>	alarm interrupt flag
<i>RTC_FLAG_OVERFLOW</i> <i>W</i>	overflow interrupt flag
<i>RTC_FLAG_RSYN</i>	registers synchronized flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the RTC alarm flag */

rtc_flag_clear (RTC_FLAG_ALARM);
```

## 3.23. SDIO

The secure digital input/output interface (SDIO) defines the SD/SD I/O /MMC CE-ATA card host interface, which provides command/data transfer between the AHB system bus and SD memory cards, SD I/O cards, Multimedia Card (MMC), and CE-ATA devices. The SDIO registers are listed in chapter [3.23.1](#), 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 3-780. SDIO Registers**

Registers	Descriptions
SDIO_PWRCTL	Power control register
SDIO_CLKCTL	Clock control register
SDIO_CMDAGMT	Command argument register
SDIO_CMDCTL	Command control register
SDIO_RSPCMDIDX	Command index response register
SDIO_RESPx x=0..3	Response register
SDIO_DATATO	Data timeout register
SDIO_DATALEN	Data length register
SDIO_DATACTL	Data control register
SDIO_DATACNT	Data counter register
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 3-781. SDIO firmware function**

Function name	Function description
sdio_deinit	deinitialize the SDIO
sdio_clock_config	configure the SDIO clock
sdio_hardware_clock_enable	enable hardware clock control
sdio_hardware_clock_disable	disable hardware clock control
sdio_bus_mode_set	set different SDIO card bus mode
sdio_power_state_set	set the SDIO power state
sdio_power_state_get	get the SDIO power state
sdio_clock_enable	enable SDIO_CLK clock output
sdio_clock_disable	disable SDIO_CLK clock output
sdio_command_response_config	configure the command and response
sdio_wait_type_set	set the command state machine wait type
sdio_csm_enable	enable the CSM(command state machine)
sdio_csm_disable	disable the CSM(command state machine)
sdio_command_index_get	get the last response command index
sdio_response_get	get the response for the last received command
sdio_data_config	configure the data timeout, data length and data block size
sdio_data_transfer_config	configure the data transfer mode and direction
sdio_dsm_enable	enable the DSM(data state machine) for data transfer

Function name	Function description
sdio_dsm_disable	disable the DSM(data state machine)
sdio_data_write	write data(one word) to the transmit FIFO
sdio_data_read	read data(one word) from the receive FIFO
sdio_data_counter_get	get the number of remaining data bytes to be transferred to card
sdio_fifo_counter_get	get the number of words remaining to be written or read from FIFO
sdio_dma_enable	enable the DMA request for SDIO
sdio_dma_disable	disable the DMA request for SDIO
sdio_flag_get	get the flags state of SDIO
sdio_flag_clear	clear the pending flags of SDIO
sdio_interrupt_enable	enable the SDIO interrupt
sdio_interrupt_disable	disable the SDIO interrupt
sdio_interrupt_flag_get	get the interrupt flags state of SDIO
sdio_interrupt_flag_clear	clear the interrupt pending flags of SDIO
sdio_readwait_enable	enable the read wait mode(SD I/O only)
sdio_readwait_disable	disable the read wait mode(SD I/O only)
sdio_stop_readwait_enable	enable the function that stop the read wait process(SD I/O only)
sdio_stop_readwait_disable	disable the function that stop the read wait process(SD I/O only)
sdio_readwait_type_set	set the read wait type(SD I/O only)
sdio_operation_enable	enable the SD I/O mode specific operation(SD I/O only)
sdio_operation_disable	disable the SD I/O mode specific operation(SD I/O only)
sdio_suspend_enable	enable the SD I/O suspend operation(SD I/O only)
sdio_suspend_disable	disable the SD I/O suspend operation(SD I/O only)
sdio_ceata_command_enable	enable the CE-ATA command(CE-ATA only)
sdio_ceata_command_disable	disable the CE-ATA command(CE-ATA only)
sdio_ceata_interrupt_enable	enable the CE-ATA interrupt(CE-ATA only)
sdio_ceata_interrupt_disable	disable the CE-ATA interrupt(CE-ATA only)
sdio_ceata_command_completion_enable	enable the CE-ATA command completion signal(CE-ATA only)
sdio_ceata_command_completion_disable	disable the CE-ATA command completion signal(CE-ATA only)

## sdio\_deinit

The description of sdio\_deinit is shown as below:

**Table 3-782. Function sdio\_deinit**

Function name	sdio_deinit
Function prototype	void sdio_deinit(void);
Function descriptions	deinitialize the SDIO

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

Example:

```
/* deinitialize the SDIO */
```

```
sdio_deinit();
```

### sdio\_clock\_config

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

**Table 3-783. Function sdio\_clock\_config**

<b>Function name</b>	sdio_clock_config
<b>Function prototype</b>	void sdio_clock_config(uint32_t clock_edge, uint32_t clock_bypass, uint32_t clock_powersave, uint16_t clock_division);
<b>Function descriptions</b>	configure the SDIO clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>clock_edge</b>	SDIO_CLK clock edge
SDIO_SDIOLCKEDGE_RISING	select the rising edge of the SDIOCLK to generate SDIO_CLK
SDIO_SDIOLCKEDGE_FALLING	select the falling edge of the SDIOCLK to generate SDIO_CLK
<b>Input parameter{in}</b>	
<b>clock_bypass</b>	clock bypass
SDIO_CLOCKBYPASS_ENABLE	clock bypass
SDIO_CLOCKBYPASS_DISABLE	no bypass
<b>Input parameter{in}</b>	
<b>clock_powersave</b>	SDIO_CLK clock dynamic switch on/off for power saving
SDIO_CLOCKPW RSA_VE_ENABLE	SDIO_CLK closed when bus is idle
SDIO_CLOCKPW RSA_VE_DISABLE	SDIO_CLK clock is always on
<b>Input parameter{in}</b>	

<b>clock_division</b>	clock division, less than 512
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the SDIO clock */
```

```
sdio_clock_config(SDIO_SDIIOCLKEDGE_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 3-784. 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 3-785. 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 3-786. Function sdio\_bus\_mode\_set**

Function name	sdio_bus_mode_set
Function prototype	void sdio_bus_mode_set(uint32_t bus_mode);
Function descriptions	set different SDIO card bus mode
Precondition	-
The called functions	-
Input parameter{in}	
bus_mode	SDIO card bus mode
SDIO_BUSMODE_1BIT	1-bit SDIO card bus mode
SDIO_BUSMODE_4BIT	4-bit SDIO card bus mode
SDIO_BUSMODE_8BIT	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 3-787. Function sdio\_power\_state\_set**

Function name	sdio_power_state_set
Function prototype	void sdio_power_state_set(uint32_t power_state);
Function descriptions	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
<i>SDIO_POWER_ON</i>	SDIO power on
<i>SDIO_POWER_OFF</i>	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 3-788. Function sdio\_power\_state\_get**

<b>Function name</b>	sdio_power_state_get
<b>Function prototype</b>	uint32_t sdio_power_state_get(void);
<b>Function descriptions</b>	get the SDIO power state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	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 3-789. Function sdio\_clock\_enable**

<b>Function name</b>	sdio_clock_enable
----------------------	-------------------

<b>Function prototype</b>	void sdio_clock_enable(void);
<b>Function descriptions</b>	enable SDIO_CLK clock output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SDIO_CLK clock output */
sdio_clock_enable();
```

### sdio\_clock\_disable

The description of sdio\_clock\_disable is shown as below:

**Table 3-790. 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 3-791. 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 3-792. Function sdio\_wait\_type\_set**

<b>Function name</b>	sdio_wait_type_set
<b>Function prototype</b>	void sdio_wait_type_set(uint32_t wait_type);
<b>Function descriptions</b>	set the command state machine wait type
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wait_type</b>	wait type
<i>SDIO_WAITTYPE_NO</i>	not wait interrupt
<i>SDIO_WAITTYPE_INTERRUPT</i>	wait interrupt
<i>SDIO_WAITTYPE_DATAEND</i>	wait the end of data transfer
<b>Output parameter{out}</b>	

-	-
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 3-793. Function sdio\_csm\_enable**

<b>Function name</b>	sdio_csm_enable
<b>Function prototype</b>	void sdio_csm_enable(void);
<b>Function descriptions</b>	enable the CSM(command state machine)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the CSM(command state machine) */
```

```
sdio_csm_enable();
```

### sdio\_csm\_disable

The description of sdio\_csm\_disable is shown as below:

**Table 3-794. Function sdio\_csm\_disable**

<b>Function name</b>	sdio_csm_disable
<b>Function prototype</b>	void sdio_csm_disable(void);
<b>Function descriptions</b>	disable the CSM(command state machine)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* disable the CSM(command state machine) */
```

```
sdio_csm_disable();
```

### sdio\_command\_index\_get

The description of sdio\_command\_index\_get is shown as below:

**Table 3-795. Function sdio\_command\_index\_get**

<b>Function name</b>	sdio_command_index_get
<b>Function prototype</b>	uint8_t sdio_command_index_get(void);
<b>Function descriptions</b>	get the last response command index
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint8_t	last response command index

Example:

```
/* get SDIO command index */
```

```
uint8_t sdio_commond_value;
```

```
sdio_commond_value = sdio_command_index_get();
```

### sdio\_response\_get

The description of sdio\_response\_get is shown as below:

**Table 3-796. Function sdio\_response\_get**

<b>Function name</b>	sdio_response_get
<b>Function prototype</b>	uint32_t sdio_response_get(uint32_t responsex);
<b>Function descriptions</b>	get the response for the last received command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
responsex	SDIO response
SDIO_RESPONSE0	card response[31:0]/card response[127:96]
SDIO_RESPONSE1	card response[95:64]
SDIO_RESPONSE2	card response[63:32]

<b>SDIO_RESPONSE3</b>	card response[31:1], plus bit 0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	response for the last received command

Example:

```
/* store the CID0 numbers */
```

```
uint32_t sdio_cid[0];
```

```
sdio_cid[0] = sdio_response_get(SDIO_RESPONSE0);
```

### sdio\_data\_config

The description of sdio\_data\_config is shown as below:

**Table 3-797. Function sdio\_data\_config**

<b>Function name</b>	sdio_data_config
<b>Function prototype</b>	void sdio_data_config(uint32_t data_timeout, uint32_t data_length, uint32_t data_blocksize);
<b>Function descriptions</b>	configure the data timeout, data length and data block size
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>data_timeout</b>	data timeout period in card bus clock periods
<b>Input parameter{in}</b>	
<b>data_length</b>	number of data bytes to be transferred
<b>Input parameter{in}</b>	
<b>data_blocksize</b>	size of data block for block transfer
<b>SDIO_DATABLOCKSIZE_1BYTE</b>	block size = 1 byte
<b>SDIO_DATABLOCKSIZE_2BYTES</b>	block size = 2 bytes
<b>SDIO_DATABLOCKSIZE_4BYTES</b>	block size = 4 bytes
<b>SDIO_DATABLOCKSIZE_8BYTES</b>	block size = 8 bytes
<b>SDIO_DATABLOCKSIZE_16BYTES</b>	block size = 16 bytes
<b>SDIO_DATABLOCKSIZE_32BYTES</b>	block size = 32 bytes
<b>SDIO_DATABLOCKSIZE_64BYTES</b>	block size = 64 bytes
<b>SDIO_DATABLOCKSIZE_128BYTES</b>	block size = 128 bytes

<i>E_128BYTES</i>	
<i>SDIO_DATABLOCKSIZE_256BYTES</i>	block size = 256 bytes
<i>SDIO_DATABLOCKSIZE_512BYTES</i>	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 3-798. Function sdio\_data\_transfer\_config**

<b>Function name</b>	sdio_data_transfer_config
<b>Function prototype</b>	void sdio_data_transfer_config(uint32_t transfer_mode, uint32_t transfer_direction);
<b>Function descriptions</b>	configure the data transfer mode and direction
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>transfer_mode</b>	mode of data transfer
<i>SDIO_TRANSMODE_BLOCK</i>	block transfer
<i>SDIO_TRANSMODE_STREAM</i>	stream transfer or SDIO multibyte transfer
<b>Input parameter{in}</b>	
<b>transfer_direction</b>	data transfer direction, read or write



<i>SDIO_TRANSDIRECTI ON_TOCARD</i>	write data to card
<i>SDIO_TRANSDIRECTI ON_TOSDIO</i>	read data from card
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SDIO data transmisson */

sdio_data_transfer_config(SDIO_TRANSDIRECTION_TOSDIO,
SDIO_TRANSMODE_BLOCK);
```

## sdio\_dsm\_enable

The description of sdio\_dsm\_enable is shown as below:

**Table 3-799. 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 3-800. 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>	-

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

Example:

```
/* disable the DSM(data state machine) */
```

```
sdio_dsm_disable();
```

### sdio\_data\_write

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

**Table 3-801. Function sdio\_data\_write**

Function name	sdio_data_write
Function prototype	void sdio_data_write(uint32_t data);
Function descriptions	write data(one word) to the transmit FIFO
Precondition	-
The called functions	-
Input parameter{in}	
data	32-bit data write to card
Output parameter{out}	
-	-
Return value	
-	-

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 3-802. Function sdio\_data\_read**

Function name	sdio_data_read
Function prototype	uint32_t sdio_data_read(void);
Function descriptions	read data(one word) from the receive FIFO
Precondition	-
The called functions	-
Input parameter{in}	

-	-
Output parameter{out}	
-	-
Return value	
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 3-803. Function sdio\_data\_counter\_get**

Function name	sdio_data_counter_get
Function prototype	uint32_t sdio_data_counter_get(void);
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	
uint32_t	number of remaining data bytes to be transferred

Example:

```
/* get the number of remaining data bytes to be transferred to card */
```

```
uint32_t sdio_data_value;
```

```
sdio_data_value = sdio_data_counter_get();
```

### sdio\_fifo\_counter\_get

The description of sdio\_fifo\_counter\_get is shown as below:

**Table 3-804. Function sdio\_data\_counter\_get**

Function name	sdio_fifo_counter_get
Function prototype	uint32_t sdio_fifo_counter_get(void);
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 3-805. Function sdio\_dma\_enable**

Function name	sdio_dma_enable
Function prototype	void sdio_dma_enable(void);
Function descriptions	enable the DMA request for SDIO
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the SDIO DMA */
```

```
sdio_dma_enable();
```

### sdio\_dma\_disable

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

**Table 3-806. Function sdio\_dma\_disable**

Function name	sdio_dma_disable
Function prototype	void sdio_dma_disable(void);
Function descriptions	disable the DMA request for SDIO
Precondition	-
The called functions	-
Input parameter{in}	

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

Example:

```
/* disable the SDIO DMA */
```

```
sdio_dma_disable();
```

## sdio\_flag\_get

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

**Table 3-807. 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
SDIO_FLAG_CCR CER 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_DATA_CNT, is zero) flag
SDIO_FLAG_STBITE	start bit error in the bus flag
SDIO_FLAG_DTBLKE ND	data block sent/received (CRC check passed) flag
SDIO_FLAG_CMDRUN	command transmission in progress flag
SDIO_FLAG_TXRUN	data transmission in progress flag
SDIO_FLAG_RXRUN	data reception in progress flag

<i>SDIO_FLAG_TFH</i>	transmit FIFO is half empty flag: at least 8 words can be written into the FIFO
<i>SDIO_FLAG_RFH</i>	receive FIFO is half full flag: at least 8 words can be read in the FIFO
<i>SDIO_FLAG_TFF</i>	transmit FIFO is full flag
<i>SDIO_FLAG_RFF</i>	receive FIFO is full flag
<i>SDIO_FLAG_TFE</i>	transmit FIFO is empty flag
<i>SDIO_FLAG_RFE</i>	receive FIFO is empty flag
<i>SDIO_FLAG_TXDTVAL</i>	data is valid in transmit FIFO flag
<i>SDIO_FLAG_RXDTVAL</i>	data is valid in receive FIFO flag
<i>SDIO_FLAG_SDIOINT</i>	SD I/O interrupt received flag
<i>SDIO_FLAG_ATAEND</i>	CE-ATA command completion signal received (only for CMD61) flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the flags state of SDIO */
```

```
FlagStatus flag_value;
```

```
flag_value = sdio_flag_get(SDIO_FLAG_RFH);
```

## sdio\_flag\_clear

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

**Table 3-808. Function sdio\_flag\_clear**

<b>Function name</b>	sdio_flag_clear
<b>Function prototype</b>	void sdio_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear the pending flags of SDIO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flags state of SDIO
<i>SDIO_FLAG_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_DTBKE ND</i>	data block sent/received (CRC check passed) flag
<i>SDIO_FLAG_SDIOINT</i>	SD I/O interrupt received flag
<i>SDIO_FLAG_ATAEND</i>	CE-ATA command completion signal received (only for CMD61) flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the pending flags of SDIO */
```

```
sdio_flag_clear(SDIO_FLAG_DTCRCERR);
```

## sdio\_interrupt\_enable

The description of *sdio\_interrupt\_enable* is shown as below:

**Table 3-809. Function *sdio\_interrupt\_enable***

<b>Function name</b>	<i>sdio_interrupt_enable</i>
<b>Function prototype</b>	<code>void sdio_interrupt_enable(uint32_t int_flag);</code>
<b>Function descriptions</b>	enable the SDIO interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt flags state of SDIO
<i>SDIO_INT_CCRRCERR</i>	SDIO CCRRCERR interrupt
<i>SDIO_INT_DTCRCER R</i>	SDIO DTCRCERR interrupt
<i>SDIO_INT_CMDTMOU T</i>	SDIO CMDTMOUT interrupt
<i>SDIO_INT_DTTMOUT</i>	SDIO DTTMOUT interrupt
<i>SDIO_INT_TXURE</i>	SDIO TXURE interrupt
<i>SDIO_INT_RXORE</i>	SDIO_INT_RXORE
<i>SDIO_INT_CMDRECV</i>	SDIO CMDRECV interrupt
<i>SDIO_INT_CMDSEND</i>	SDIO CMDSEND interrupt
<i>SDIO_INT_DTEND</i>	SDIO DTEND interrupt

<i>SDIO_INT_STBITE</i>	SDIO STBITE interrupt
<i>SDIO_INT_DTBLKEND</i>	SDIO DTBLKEND interrupt
<i>SDIO_INT_CMDRUN</i>	SDIO CMDRUN interrupt
<i>SDIO_INT_TXRUN</i>	SDIO TXRUN interrupt
<i>SDIO_INT_RXRUN</i>	SDIO RXRUN interrupt
<i>SDIO_INT_TFH</i>	SDIO TFH interrupt
<i>SDIO_INT_RFH</i>	SDIO RFH interrupt
<i>SDIO_INT_TFF</i>	SDIO TFF interrupt
<i>SDIO_INT_RFF</i>	SDIO RFF interrupt
<i>SDIO_INT_TFE</i>	SDIO TFE interrupt
<i>SDIO_INT_RFE</i>	SDIO RFE interrupt
<i>SDIO_INT_TXDTVAL</i>	SDIO TXDTVAL interrupt
<i>SDIO_INT_RXDTVAL</i>	SDIO RXDTVAL interrupt
<i>SDIO_INT_SDIOINT</i>	SDIO SDIOINT interrupt
<i>SDIO_INT_ATAEND</i>	SDIO ATAEND interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the SDIO corresponding interrupts */
```

```
sdio_interrupt_enable(SDIO_INT_CCRRCERR | SDIO_INT_DTTMOUT | SDIO_INT_RXORE  
| SDIO_INT_DTEND | SDIO_INT_STBITE);
```

## sdio\_interrupt\_disable

The description of `sdio_interrupt_disable` is shown as below:

**Table 3-810. Function `sdio_interrupt_disable`**

<b>Function name</b>	<code>sdio_interrupt_disable</code>
<b>Function prototype</b>	<code>void sdio_interrupt_disable(uint32_t int_flag);</code>
<b>Function descriptions</b>	disable the SDIO interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt flags state of SDIO
<i>SDIO_INT_CCRRCERR</i>	SDIO CCRRCERR interrupt
<i>SDIO_INT_DTCRCER R</i>	SDIO DTCRCERR interrupt
<i>SDIO_INT_CMDTMOU T</i>	SDIO CMDTMOUT interrupt
<i>SDIO_INT_DTTMOUT</i>	SDIO DTTMOUT interrupt



<i>SDIO_INT_TXURE</i>	SDIO TXURE interrupt
<i>SDIO_INT_RXORE</i>	SDIO_INT_RXORE
<i>SDIO_INT_CMDRECV</i>	SDIO CMDRECV interrupt
<i>SDIO_INT_CMDSEND</i>	SDIO CMDSEND interrupt
<i>SDIO_INT_DTEND</i>	SDIO DTEND interrupt
<i>SDIO_INT_STBITE</i>	SDIO STBITE interrupt
<i>SDIO_INT_DTBLKEND</i>	SDIO DTBLKEND interrupt
<i>SDIO_INT_CMDRUN</i>	SDIO CMDRUN interrupt
<i>SDIO_INT_TXRUN</i>	SDIO TXRUN interrupt
<i>SDIO_INT_RXRUN</i>	SDIO RXRUN interrupt
<i>SDIO_INT_TFH</i>	SDIO TFH interrupt
<i>SDIO_INT_RFH</i>	SDIO RFH interrupt
<i>SDIO_INT_TFF</i>	SDIO TFF interrupt
<i>SDIO_INT_RFF</i>	SDIO RFF interrupt
<i>SDIO_INT_TFE</i>	SDIO TFE interrupt
<i>SDIO_INT_RFE</i>	SDIO RFE interrupt
<i>SDIO_INT_TXDTVAL</i>	SDIO TXDTVAL interrupt
<i>SDIO_INT_RXDTVAL</i>	SDIO RXDTVAL interrupt
<i>SDIO_INT_SDIOINT</i>	SDIO SDIOINT interrupt
<i>SDIO_INT_ATAEND</i>	SDIO ATAEND interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* 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 3-811. Function `sdio_interrupt_flag_get`**

<b>Function name</b>	<code>sdio_interrupt_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus sdio_interrupt_flag_get(uint32_t int_flag);</code>
<b>Function descriptions</b>	get the interrupt flags state of SDIO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt flags state of SDIO
<i>SDIO_INT_FLAG_CCR CERR</i>	SDIO CCRCERR interrupt flag

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
SDIO_INT_FLAG_CMD SEND	SDIO CMDSEND interrupt flag
SDIO_INT_FLAG_DTE ND	SDIO DTEND interrupt flag
SDIO_INT_FLAG_STBI TE	SDIO STBITE interrupt flag
SDIO_INT_FLAG_DTB LKEND	SDIO DTBLKEND interrupt flag
SDIO_INT_FLAG_CMD RUN	SDIO CMDRUN interrupt flag
SDIO_INT_FLAG_TXR UN	SDIO TXRUN interrupt flag
SDIO_INT_FLAG_RXR UN	SDIO RXRUN interrupt flag
SDIO_INT_FLAG_TFH	SDIO TFH interrupt flag
SDIO_INT_FLAG_RFH	SDIO RFH interrupt flag
SDIO_INT_FLAG_TFF	SDIO TFF interrupt flag
SDIO_INT_FLAG_RFF	SDIO RFF interrupt flag
SDIO_INT_FLAG_TFE	SDIO TFE interrupt flag
SDIO_INT_FLAG_RFE	SDIO RFE interrupt flag
SDIO_INT_FLAG_TXD TVAL	SDIO TXDTVAL interrupt flag
SDIO_INT_FLAG_RXD TVAL	SDIO RXDTVAL interrupt flag
SDIO_INT_FLAG_SDI OINT	SDIO SDIOINT interrupt flag
SDIO_INT_FLAG_ATA END	SDIO ATAEND interrupt flag
Output parameter{out}	
-	-
Return value	

<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 3-812. Function sdio\_interrupt\_flag\_clear**

<b>Function name</b>	sdio_interrupt_flag_clear
<b>Function prototype</b>	void sdio_interrupt_flag_clear(uint32_t int_flag);
<b>Function descriptions</b>	clear the interrupt pending flags of SDIO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt flags state of SDIO
<i>SDIO_INT_FLAG_CCR CERR</i>	command response received (CRC check failed) flag
<i>SDIO_INT_FLAG_DTC RCERR</i>	data block sent/received (CRC check failed) flag
<i>SDIO_INT_FLAG_CMD TMOUT</i>	command response timeout flag
<i>SDIO_INT_FLAG_DTT MOUT</i>	data timeout flag
<i>SDIO_INT_FLAG_TXU RE</i>	transmit FIFO underrun error occurs flag
<i>SDIO_INT_FLAG_RXO RE</i>	received FIFO overrun error occurs flag
<i>SDIO_INT_FLAG_CMD RECV</i>	command response received (CRC check passed) flag
<i>SDIO_INT_FLAG_CMD SEND</i>	command sent (no response required) flag
<i>SDIO_INT_FLAG_DTE ND</i>	data end (data counter, SDIO_DATACNT, is zero) flag
<i>SDIO_INT_FLAG_STBI TE</i>	start bit error in the bus flag
<i>SDIO_INT_FLAG_DTB LKEND</i>	data block sent/received (CRC check passed) flag
<i>SDIO_INT_FLAG_SDI OINT</i>	SD I/O interrupt received flag

<i>SDIO_INT_FLAG_ATA END</i>	CE-ATA command completion signal received (only for CMD61) flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the interrupt pending flags of SDIO */
sdio_interrupt_flag_clear(SDIO_INT_FLAG_DTEND);
```

### sdio\_readwait\_enable

The description of sdio\_readwait\_enable is shown as below:

**Table 3-813. 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 3-814. 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>	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the read wait mode(SD I/O only) */
```

```
sdio_readwait_disable();
```

### sdio\_stop\_readwait\_enable

The description of sdio\_stop\_readwait\_enable is shown as below:

**Table 3-815. Function sdio\_stop\_readwait\_enable**

Function name	sdio_stop_readwait_enable
Function prototype	void sdio_stop_readwait_enable(void);
Function descriptions	enable 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:

```
/* 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 3-816. 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 3-817. Function sdio\_readwait\_type\_set**

Function name	sdio_readwait_type_set
Function prototype	void sdio_readwait_type_set(uint32_t readwait_type);
Function descriptions	set the read wait type(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
readwait_type	SD I/O read wait type
SDIO_READWAITTYPE_CLK	read wait control by stopping SDIO_CLK
SDIO_READWAITTYPE_DAT2	read wait control using SDIO_DAT[2]
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the read wait type(SD I/O only) */
```

```
sdio_readwait_type_set(uint32_t readwait_type);
```

### sdio\_operation\_enable

The description of sdio\_operation\_enable is shown as below:

**Table 3-818. Function sdio\_operation\_enable**

Function name	sdio_operation_enable
Function prototype	void sdio_operation_enable(void);
Function descriptions	enable the SD I/O mode specific operation(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
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 3-819. 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 3-820. Function sdio\_suspend\_enable**

Function name	sdio_suspend_enable
Function prototype	void sdio_suspend_enable(void);
Function descriptions	enable the SD I/O suspend operation(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* enable the SD I/O suspend operation(SD I/O only) */
```

```
sdio_suspend_enable();
```

### sdio\_suspend\_disable

The description of sdio\_suspend\_disable is shown as below:

**Table 3-821. Function sdio\_suspend\_disable**

Function name	sdio_suspend_disable
Function prototype	void sdio_suspend_disable(void);
Function descriptions	disable the SD I/O suspend operation(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the SD I/O suspend operation(SD I/O only) */
```

```
sdio_suspend_disable();
```

### sdio\_ceata\_command\_enable

The description of sdio\_ceata\_command\_enable is shown as below:

**Table 3-822. Function sdio\_ceata\_command\_enable**

Function name	sdio_ceata_command_enable
Function prototype	void sdio_ceata_command_enable(void);
Function descriptions	enable the CE-ATA command(CE-ATA only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-



Example:

```
/* enable the CE-ATA command(CE-ATA only) */  
  
sdio_ceata_command_enable();
```

## sdio\_ceata\_command\_disable

The description of sdio\_ceata\_command\_disable is shown as below:

**Table 3-823. Function sdio\_ceata\_command\_disable**

<b>Function name</b>	sdio_ceata_command_disable
<b>Function prototype</b>	void sdio_ceata_command_disable(void);
<b>Function descriptions</b>	disable the CE-ATA command(CE-ATA only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the CE-ATA command(CE-ATA only) */  
  
sdio_ceata_command_disable();
```

## sdio\_ceata\_interrupt\_enable

The description of sdio\_ceata\_interrupt\_enable is shown as below:

**Table 3-824. Function sdio\_ceata\_interrupt\_enable**

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

Example:

```
/* enable the CE-ATA interrupt(CE-ATA only) */
```

```
sdio_ceata_interrupt_enable();
```

### sdio\_ceata\_interrupt\_disable

The description of sdio\_ceata\_interrupt\_disable is shown as below:

**Table 3-825. Function sdio\_ceata\_interrupt\_disable**

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

Example:

```
/* disable the CE-ATA interrupt(CE-ATA only) */
```

```
sdio_ceata_interrupt_disable();
```

### sdio\_ceata\_command\_completion\_enable

The description of sdio\_ceata\_command\_completion\_enable is shown as below:

**Table 3-826. 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 3-827. 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 3-828. SPI/I2S Registers**

Registers	Descriptions
SPI_CTL0	SPI control register 0
SPI_CTL1	SPI control register 1
SPI_STAT	SPI status register
SPI_DATA	SPI data register
SPI_CRCPOLY	SPI CRC polynomial register
SPI_RCRC	SPI receive CRC register
SPI_TCRC	SPI transmit CRC register
SPI_I2SCTL	SPI/I2S control register

Registers	Descriptions
SPI_I2SPSC	SPI/I2S clock prescaler register
SPI_QCTL	SPI quad mode control register

## 3.24.2. Descriptions of Peripheral functions

SPI/I2S firmware functions are listed in the table shown as below:

**Table 3-829. 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
spi_i2s_format_error_clear	clear TI Mode Format Error flag status
spi_crc_polynomial_set	set SPI CRC polynomial
spi_crc_polynomial_get	get SPI CRC polynomial
spi_crc_on	turn on SPI CRC function
spi_crc_off	turn off SPI CRC function
spi_crc_next	SPI next data is CRC value
spi_crc_get	get SPI CRC send value or receive value
spi_crc_error_clear	clear SPI CRC error flag status
spi_ti_mode_enable	enable SPI TI mode
spi_ti_mode_disable	disable SPI TI mode
spi_nssp_mode_enable	enable SPI NSS pulse mode
spi_nssp_mode_disable	disable SPI NSS pulse mode
i2s_full_duplex_mode_config	configure i2s full duplex mode
spi_quad_enable	enable SPI quad wire mode

Function name	Function description
spi_quad_disable	disable SPI quad wire mode
spi_quad_write_enable	enable SPI quad wire mode write
spi_quad_read_enable	enable SPI quad wire mode read
spi_quad_io23_output_enable	enable SPI quad wire mode SPI_IO2 and SPI_IO3 pin output
spi_quad_io23_output_disable	disable SPI quad wire mode SPI_IO2 and SPI_IO3 pin output
spi_i2s_interrupt_enable	enable SPI and I2S interrupt
spi_i2s_interrupt_disable	disable SPI and I2S interrupt
spi_i2s_interrupt_flag_get	get SPI and I2S interrupt status
spi_i2s_flag_get	get SPI and I2S flag status

## Structure spi\_parameter\_struct

Table 3-830. 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 3-831. Function spi\_i2s\_deinit

Function name	spi_i2s_deinit
Function prototype	void spi_i2s_deinit(uint32_t spi_periph);
Function descriptions	reset SPI and I2S peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
spi_periph	SPI/I2S peripheral

<i>SPIx</i>	<i>x=0,1,2</i>
<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 3-832. 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 3-833. 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
<i>SPIx</i>	x=0,1,2
<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 3-830. 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_FRAMESIZE_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 3-834. Function spi\_enable**

<b>Function name</b>	spi_enable
<b>Function prototype</b>	void spi_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable SPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 */

spi_enable(SPI0);
```

## spi\_disable

The description of spi\_disable is shown as below:

**Table 3-835. Function spi\_disable**

<b>Function name</b>	spi_disable
<b>Function prototype</b>	void spi_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable SPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 */

spi_disable(SPI0);
```

## i2s\_init

The description of i2s\_init is shown as below:

**Table 3-836. 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>	I2S master transmit mode
<i>I2S_MODE_MASTERRX</i>	I2S master receive mode
<b>Input parameter{in}</b>	
<b>i2s_standard</b>	I2S standard
<i>I2S_STD_PHILLIPS</i>	I2S phillips standard
<i>I2S_STD_MSB</i>	I2S MSB standard
<i>I2S_STD_LSB</i>	I2S LSB standard
<i>I2S_STD_PCMSHORT</i>	I2S PCM short standard
<i>I2S_STD_PCMLONG</i>	I2S PCM long standard
<b>Input parameter{in}</b>	
<b>i2s_ckpl</b>	I2S idle state clock polarity
<i>I2S_CKPL_LOW</i>	I2S clock polarity low level
<i>I2S_CKPL_HIGH</i>	I2S clock polarity high level
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize I2S1 */
```

```
i2s_init(SPI1, I2S_MODE_MASTERTX, I2S_STD_PHILLIPS, I2S_CKPL_LOW);
```

## i2s\_psc\_config

The description of i2s\_psc\_config is shown as below:

**Table 3-837. Function i2s\_psc\_config**

<b>Function name</b>	i2s_psc_config
<b>Function prototype</b>	void i2s_psc_config(uint32_t spi_periph, uint32_t i2s_audiosample, uint32_t i2s_frameformat, uint32_t i2s_mckout);
<b>Function descriptions</b>	configure I2S prescaler
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S peripheral
<i>SP1x</i>	x=1,2
<b>Input parameter{in}</b>	
<b>i2s_audiosample</b>	I2S audio sample rate
<i>I2S_AUDIOSAMPLE_8K</i>	audio sample rate is 8KHz

<i>I2S_AUDIOSAMPLE_1</i> 1K	audio sample rate is 11KHz
<i>I2S_AUDIOSAMPLE_1</i> 6K	audio sample rate is 16KHz
<i>I2S_AUDIOSAMPLE_2</i> 2K	audio sample rate is 22KHz
<i>I2S_AUDIOSAMPLE_3</i> 2K	audio sample rate is 32KHz
<i>I2S_AUDIOSAMPLE_4</i> 4K	audio sample rate is 44KHz
<i>I2S_AUDIOSAMPLE_4</i> 8K	audio sample rate is 48KHz
<i>I2S_AUDIOSAMPLE_9</i> 6K	audio sample rate is 96KHz
<i>I2S_AUDIOSAMPLE_1</i> 92K	audio sample rate is 192KHz
<b>Input parameter{in}</b>	
<b>i2s_frameformat</b>	I2S data length and channel length
<i>I2S_FRAMEFORMAT_DT16B_CH16B</i>	I2S data length is 16 bit and channel length is 16 bit
<i>I2S_FRAMEFORMAT_DT16B_CH32B</i>	I2S data length is 16 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT24B_CH32B</i>	I2S data length is 24 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT32B_CH32B</i>	I2S data length is 32 bit and channel length is 32 bit
<b>Input parameter{in}</b>	
<b>i2s_mckout</b>	I2S master clock output
<i>I2S_MCKOUT_ENABL</i> E	I2S master clock output enable
<i>I2S_MCKOUT_DISABL</i> E	I2S master clock output disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure I2S1 prescaler */
```

```
i2s_psc_config(SPI1, I2S_AUDIOSAMPLE_44K, I2S_FRAMEFORMAT_DT16B_CH16B,  
I2S_MCKOUT_DISABLE);
```

## i2s\_enable

The description of i2s\_enable is shown as below:

**Table 3-838. 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 3-839. 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>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2S1*/
i2s_disable(SPI1);
```

## **spi\_nss\_output\_enable**

The description of spi\_nss\_output\_enable is shown as below:

**Table 3-840. Function spi\_nss\_output\_enable**

<b>Function name</b>	spi_nss_output_enable
<b>Function prototype</b>	void spi_nss_output_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable SPI NSS output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 NSS output */
spi_nss_output_enable(SPI0);
```

## **spi\_nss\_output\_disable**

The description of spi\_nss\_output\_disable is shown as below:

**Table 3-841. Function spi\_nss\_output\_disable**

<b>Function name</b>	spi_nss_output_disable
<b>Function prototype</b>	void spi_nss_output_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable SPI NSS output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 NSS output */
spi_nss_output_disable(SPI0);
```

## spi\_nss\_internal\_high

The description of spi\_nss\_internal\_high is shown as below:

**Table 3-842. Function spi\_nss\_internal\_high**

<b>Function name</b>	spi_nss_internal_high
<b>Function prototype</b>	void spi_nss_internal_high(uint32_t spi_periph);
<b>Function descriptions</b>	SPI NSS pin high level in software mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 NSS pin is pulled high level in software mode */
```

```
spi_nss_internal_high(SPI0);
```

## spi\_nss\_internal\_low

The description of spi\_nss\_internal\_low is shown as below:

**Table 3-843. Function spi\_nss\_internal\_low**

<b>Function name</b>	spi_nss_internal_low
<b>Function prototype</b>	void spi_nss_internal_low(uint32_t spi_periph);
<b>Function descriptions</b>	SPI NSS pin low level in software mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 NSS pin is pulled low level in software mode */
```

```
spi_nss_internal_low(SPI0);
```

## spi\_dma\_enable

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

**Table 3-844. Function spi\_dma\_enable**

<b>Function name</b>	spi_dma_enable
<b>Function prototype</b>	void spi_dma_enable(uint32_t spi_periph, uint8_t dma);
<b>Function descriptions</b>	enable SPI DMA function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>dma</b>	SPI DMA mode
<i>SPI_DMA_TRANSMIT</i>	SPI transmit data use DMA
<i>SPI_DMA_RECEIVE</i>	SPI receive data use DMA
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* 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 3-845. Function spi\_dma\_disable**

<b>Function name</b>	spi_dma_disable
<b>Function prototype</b>	void spi_dma_disable(uint32_t spi_periph, uint8_t dma);
<b>Function descriptions</b>	disable SPI DMA function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>dma</b>	SPI DMA mode
<i>SPI_DMA_TRANSMIT</i>	SPI transmit data use DMA
<i>SPI_DMA_RECEIVE</i>	SPI receive data use DMA
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

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 3-846. Function spi\_i2s\_data\_frame\_format\_config**

<b>Function name</b>	spi_i2s_data_frame_format_config
<b>Function prototype</b>	void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
<b>Function descriptions</b>	configure SPI/I2S data frame format
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>frame_format</b>	SPI frame size
<i>SPI_FRAME_SIZE_16BIT</i>	SPI frame size is 16 bits
<i>SPI_FRAME_SIZE_8BIT</i>	SPI frame size is 8 bits
<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 3-847. 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
<b>Input parameter{in}</b>	
<b>data</b>	16-bit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 transmit data */
uint16_t spi0_send_array[] = {0x5050,0xA0A0};
spi_i2s_data_transmit(SPI0, spi0_send_array[0]);
```

### spi\_i2s\_data\_receive

The description of spi\_i2s\_data\_receive is shown as below:

**Table 3-848. Function spi\_i2s\_data\_receive**

<b>Function name</b>	spi_i2s_data_receive
<b>Function prototype</b>	uint16_t spi_i2s_data_receive(uint32_t spi_periph);
<b>Function descriptions</b>	SPI receive data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	16-bit data

Example:

```
/* SPI0 receive data */
uint16_t spi0_receive_data;
spi0_receive_data = spi_i2s_data_receive(SPI0);
```



## spi\_bidirectional\_transfer\_config

The description of spi\_bidirectional\_transfer\_config is shown as below:

**Table 3-849. Function spi\_bidirectional\_transfer\_config**

<b>Function name</b>	spi_bidirectional_transfer_config
<b>Function prototype</b>	void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);
<b>Function descriptions</b>	configure SPI bidirectional transfer direction
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>transfer_direction</b>	SPI transfer direction
<i>SPI_BIDIRECTIONAL_TRANSMIT</i>	SPI work in transmit-only mode
<i>SPI_BIDIRECTIONAL_RECEIVE</i>	SPI work in receive-only mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

## spi\_i2s\_format\_error\_clear

The description of spi\_i2s\_format\_error\_clear is shown as below:

**Table 3-850. 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
<b>Input parameter{in}</b>	
<b>flag</b>	SPI/I2S format error flag

<i>SPI_FLAG_FERR</i>	Format error only for SPI work in TI mode
<i>I2S_FLAG_FERR</i>	Format error for I2S
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear SPI0 format error flag status */
```

```
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 3-851. Function spi\_crc\_polynomial\_set**

<b>Function name</b>	spi_crc_polynomial_set
<b>Function prototype</b>	void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly);
<b>Function descriptions</b>	set SPI CRC polynomial
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>crc_poly</b>	CRC polynomial value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set SPI0 CRC polynomial */
```

```
spi_crc_polynomial_set(SPI0,CRC_VALUE);
```

## spi\_crc\_polynomial\_get

The description of spi\_crc\_polynomial\_get is shown as below:

**Table 3-852. 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
<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 3-853. 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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn on SPI0 CRC function */

spi_crc_on(SPI0);
```

### spi\_crc\_off

The description of spi\_crc\_off is shown as below:

**Table 3-854. Function spi\_crc\_off**

<b>Function name</b>	spi_crc_off
<b>Function prototype</b>	void spi_crc_off(uint32_t spi_periph);

<b>Function descriptions</b>	turn off SPI CRC function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn off SPI0 CRC function */
spi_crc_off(SPI0);
```

## spi\_crc\_next

The description of spi\_crc\_next is shown as below:

**Table 3-855. Function spi\_crc\_next**

<b>Function name</b>	spi_crc_next
<b>Function prototype</b>	void spi_crc_next(uint32_t spi_periph);
<b>Function descriptions</b>	SPI next data is CRC value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 next data is CRC value */
spi_crc_next(SPI0);
```

## spi\_crc\_get

The description of spi\_crc\_get is shown as below:

**Table 3-856. Function spi\_crc\_get**

<b>Function name</b>	spi_crc_get
<b>Function prototype</b>	uint16_t spi_crc_get(uint32_t spi_periph, uint8_t crc);

<b>Function descriptions</b>	get SPI CRC send value or receive value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>crc</b>	SPI crc value
<i>SPI_CRC_TX</i>	get transmit crc value
<i>SPI_CRC_RX</i>	get receive crc value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	16-bit CRC value (0-0xFFFF)

Example:

```
/* get SPI0 CRC send value */
uint16_t crc_val;
crc_val = spi_crc_get(SPI0, SPI_CRC_TX);
```

### spi\_crc\_error\_clear

The description of spi\_crc\_error\_clear is shown as below:

**Table 3-857. Function spi\_crc\_error\_clear**

<b>Function name</b>	spi_crc_error_clear
<b>Function prototype</b>	void spi_crc_error_clear(uint32_t spi_periph);
<b>Function descriptions</b>	clear SPI CRC error flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear SPI0 CRC error flag status */
spi_crc_error_clear(SPI0);
```

## **spi\_ti\_mode\_enable**

The description of spi\_ti\_mode\_enable is shown as below:

**Table 3-858. Function spi\_ti\_mode\_enable**

<b>Function name</b>	spi_ti_mode_enable
<b>Function prototype</b>	void spi_ti_mode_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable SPI TI mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 TI mode */
spi_ti_mode_enable(SPI0);
```

## **spi\_ti\_mode\_disable**

The description of spi\_ti\_mode\_disable is shown as below:

**Table 3-859. Function spi\_ti\_mode\_disable**

<b>Function name</b>	spi_ti_mode_disable
<b>Function prototype</b>	void spi_ti_mode_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable SPI TI mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 TI mode */
spi_ti_mode_disable(SPI0);
```

## spi\_nssp\_mode\_enable

The description of spi\_nssp\_mode\_enable is shown as below:

**Table 3-860. Function spi\_nssp\_mode\_enable**

<b>Function name</b>	spi_nssp_mode_enable
<b>Function prototype</b>	void spi_nssp_mode_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable SPI NSS pulse mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 NSS pulse mode */
spi_nssp_mode_enable(SPI0);
```

## spi\_nssp\_mode\_disable

The description of spi\_nssp\_mode\_disable is shown as below:

**Table 3-861. Function spi\_nssp\_mode\_disable**

<b>Function name</b>	spi_nssp_mode_disable
<b>Function prototype</b>	void spi_nssp_mode_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable SPI NSS pulse mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 NSS pulse mode */
spi_nssp_mode_disable(SPI0);
```

## i2s\_full\_duplex\_mode\_config

The description of i2s\_full\_duplex\_mode\_config is shown as below:

**Table 3-862. Function i2s\_init**

<b>Function name</b>	i2s_full_duplex_mode_config
<b>Function prototype</b>	void i2s_full_duplex_mode_config(uint32_t i2s_add_periph,uint32_t i2s_mode,uint32_t i2s_standard,uint32_t i2s_ckpl,uint32_t i2s_frameformat);
<b>Function descriptions</b>	configure i2s full duplex mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2s_add_periph</b>	I2S_ADD peripheral
<i>I2Sx_ADD</i>	x=1,2
<b>Input parameter{in}</b>	
<b>i2s_mode</b>	I2S operation mode
<i>I2S_MODE_SLAVETX</i>	I2S slave transmit mode
<i>I2S_MODE_SLAVERX</i>	I2S slave receive mode
<i>I2S_MODE_MASTERTX</i>	I2S master transmit mode
<i>I2S_MODE_MASTERRX</i>	I2S master receive mode
<b>Input parameter{in}</b>	
<b>i2s_standard</b>	I2S standard
<i>I2S_STD_PHILLIPS</i>	I2S phillips standard
<i>I2S_STD_MSB</i>	I2S MSB standard
<i>I2S_STD_LSB</i>	I2S LSB standard
<i>I2S_STD_PCMSHORT</i>	I2S PCM short standard
<i>I2S_STD_PCMLONG</i>	I2S PCM long standard
<b>Input parameter{in}</b>	
<b>i2s_ckpl</b>	I2S idle state clock polarity
<i>I2S_CKPL_LOW</i>	I2S clock polarity low level
<i>I2S_CKPL_HIGH</i>	I2S clock polarity high level
<b>Input parameter{in}</b>	
<b>i2s_frameformat</b>	I2S data length and channel length
<i>I2S_FRAMEFORMAT_DT16B_CH16B</i>	I2S data length is 16 bit and channel length is 16 bit
<i>I2S_FRAMEFORMAT_DT16B_CH32B</i>	I2S data length is 16 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT24B_CH32B</i>	I2S data length is 24 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT32B_CH32B</i>	I2S data length is 32 bit and channel length is 32 bit



<i>DT32B_CH32B</i>	
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure I2S1_ADD */
```

```
i2s_full_duplex_mode_config(I2S1_ADD,I2S_MODE_MASTERTX,I2S_STD_PHILLIPS,
I2S_CKPL_HIGH, I2S_FRAMEFORMAT_DT16B_CH16B);
```

### spi\_quad\_enable

The description of spi\_quad\_enable is shown as below:

**Table 3-863. 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 SPI quad wire 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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 quad wire mode */
```

```
spi_quad_enable (SPI0);
```

### spi\_quad\_disable

The description of spi\_quad\_disable is shown as below:

**Table 3-864. 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 SPI quad wire 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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 quad wire mode */
```

```
spi_quad_disable (SPI0);
```

## spi\_quad\_write\_enable

The description of spi\_quad\_write\_enable is shown as below:

**Table 3-865. 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 SPI quad wire mode write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 quad wire write */
```

```
spi_quad_write_enable (SPI0);
```

## spi\_quad\_read\_enable

The description of spi\_quad\_read\_enable is shown as below:

**Table 3-866. 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 SPI quad wire mode read
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 quad wire read */
```

```
spi_quad_read_enable (SPI0);
```

### spi\_quad\_io23\_output\_enable

The description of spi\_quad\_io23\_output\_enable is shown as below:

**Table 3-867. 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 quad wire mode 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=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 SPI_IO2 and SPI_IO3 pin output */
```

```
spi_quad_io23_output_enable (SPI0);
```

### spi\_quad\_io23\_output\_disable

The description of spi\_quad\_io23\_output\_disable is shown as below:

**Table 3-868. 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 quad wire mode 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=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 SPI_IO2 and SPI_IO3 pin output */
```

```
spi_quad_io23_output_disable (SPI0);
```

### spi\_i2s\_interrupt\_enable

The description of spi\_i2s\_interrupt\_enable is shown as below:

**Table 3-869. Function spi\_i2s\_interrupt\_enable**

<b>Function name</b>	spi_i2s_interrupt_enable
<b>Function prototype</b>	void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	enable SPI and I2S interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>interrupt</b>	SPI/I2S interrupt
<i>SPI_I2S_INT_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_ERR</i>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 transmit buffer empty interrupt */
```

```
spi_i2s_interrupt_enable(SPI0, SPI_I2S_INT_TBE);
```

### spi\_i2s\_interrupt\_disable

The description of spi\_i2s\_interrupt\_disable is shown as below:

Table 3-870. Function spi\_i2s\_interrupt\_disable

<b>Function name</b>	spi_i2s_interrupt_disable
<b>Function prototype</b>	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	disable SPI and I2S interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>interrupt</b>	SPI/I2S interrupt
<i>SPI_I2S_INT_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_ERR</i>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 transmit buffer empty interrupt */
```

```
spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_TBE);
```

### spi\_i2s\_interrupt\_flag\_get

The description of spi\_i2s\_interrupt\_flag\_get is shown as below:

Table 3-871. Function spi\_i2s\_interrupt\_flag\_get

<b>Function name</b>	spi_i2s_interrupt_flag_get
<b>Function prototype</b>	FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	get SPI and I2S interrupt status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>interrupt</b>	SPI/I2S interrupt flag status
<i>SPI_I2S_INT_FLAG_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_FLAG_RBNE</i>	receive buffer not empty interrupt

<i>SPI_I2S_INT_FLAG_RXORERR</i>	overrun interrupt
<i>SPI_INT_FLAG_CONFIGERR</i>	config error interrupt
<i>SPI_INT_FLAG_CRCERR</i>	CRC error interrupt
<i>I2S_INT_FLAG_TXURERR</i>	underrun error interrupt
<i>SPI_I2S_INT_FLAG_FORMATERR</i>	format error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```

/* get SPI0 transmit buffer empty interrupt status */
if(RESET != spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE)){
    while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
    spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
}

```

## spi\_i2s\_flag\_get

The description of spi\_i2s\_flag\_get is shown as below:

**Table 3-872. Function spi\_i2s\_flag\_get**

<b>Function name</b>	spi_i2s_flag_get
<b>Function prototype</b>	FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag);
<b>Function descriptions</b>	get SPI and I2S flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>flag</b>	SPI/I2S flag status
<i>SPI_FLAG_TBE</i>	transmit buffer empty flag
<i>SPI_FLAG_RBNE</i>	receive buffer not empty flag
<i>SPI_FLAG_TRANS</i>	transmit on-going flag
<i>SPI_I2S_INT_FLAG_RXORERR</i>	receive overrun error flag

<i>SPI_FLAG_CONFERR</i>	mode config error flag
<i>SPI_FLAG_CRCERR</i>	CRC error flag
<i>SPI_FLAG_FREE</i>	format error flag
<i>I2S_FLAG_TBE</i>	transmit buffer empty flag
<i>I2S_FLAG_RBNE</i>	receive buffer not empty flag
<i>I2S_FLAG_TRANS</i>	transmit on-going flag
<i>I2S_FLAG_RXORERR</i>	overrun error flag
<i>I2S_FLAG_TXURERR</i>	underrun error flag
<i>I2S_FLAG_CH</i>	channel side flag
<i>I2S_FLAG_FERR</i>	format error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get SPI0 transmit buffer empty flag status */
while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
```

## 3.25. SQPI

Serial/Quad Parallel Interface (SQPI) is a controller for external serial/dual/quad parallel interface memory peripheral. The SQPI registers are listed in chapter [3.25.1](#), the SQPI firmware functions are introduced in chapter [3.25.2](#).

### 3.25.1. Descriptions of Peripheral registers

SQPI registers are listed in the table shown as below:

**Table 3-873. SQPI Registers**

Registers	Descriptions
SQPI_INIT	SQPI initial register
SQPI_RCMD	SQPI read command register
SQPI_WCMD	SQPI write command register
SQPI_IDL	SQPI ID low register
SQPI_IDH	SQPI ID high register

### 3.25.2. Descriptions of Peripheral functions

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

**Table 3-874. SQPI firmware function**

Function name	Function description
sqpi_deinit	reset SQPI peripheral
sqpi_struct_para_init	initialize the parameters of SQPI struct with the default values
sqpi_init	initialize SQPI peripheral parameter
sqpi_read_id_command	send SQPI read ID command
sqpi_special_command	send SQPI special command
sqpi_read_command_config	configure SQPI read command
sqpi_write_command_config	configure SQPI write command
sqpi_low_id_receive	SQPI receive low ID
sqpi_high_id_receive	SQPI receive high ID

## Structure sqpi\_parameter\_struct

**Table 3-875. sqpi\_parameter\_struct**

Member name	Function description
polarity	SQPI sample polarity (SQPI_SAMPLE_POLARITY_RISING, SQPI_SAMPLE_POLARITY_FALLING)
id_length	external memory ID length (QSPI_ID_LENGTH_n_BITS (n=8,16,32,64))
addr_bit	bit number of SPI PSRAM address phase (0x00 - 0x1F)
clk_div	clock divider for SQPI output clock (0x01 - 0x3F)
cmd_bit	bit number of SQPI controller command phase (QSPI_CMDBIT_n_BITS (n=4,8,16))

## sqpi\_deinit

The description of sqpi\_deinit is shown as below:

**Table 3-876. Function sqpi\_deinit**

Function name	sqpi_deinit
Function prototype	void sqpi_deinit(void);
Function descriptions	reset SQPI peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:



```
/* reset SQPI */
```

```
sqpi_deinit();
```

## sqpi\_struct\_para\_init

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

**Table 3-877. Function sqpi\_struct\_para\_init**

<b>Function name</b>	sqpi_struct_para_init
<b>Function prototype</b>	void sqpi_struct_para_init(sqpi_parameter_struct* sqpi_struct);
<b>Function descriptions</b>	initialize the parameters of SQPI struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>*sqpi_struct</b>	a sqpi_parameter_struct address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the parameters of SPI */
```

```
sqpi_parameter_struct sqpi_init_struct;
```

```
sqpi_struct_para_init(&sqpi_init_struct);
```

## sqpi\_init

The description of sqpi\_init is shown as below:

**Table 3-878. Function sqpi\_init**

<b>Function name</b>	sqpi_init
<b>Function prototype</b>	void sqpi_init(sqpi_parameter_struct* sqpi_struct);
<b>Function descriptions</b>	initialize SQPI peripheral parameter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sqpi_struct</b>	SPI parameter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-875. sqpi_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize SQPI */
```

```
sqpi_parameter_struct sqpi_init_struct;

sqpi_struct->polarity = SQPI_SAMPLE_POLARITY_RISING;

sqpi_struct->id_length = QSPI_ID_LENGTH_32_BITS;

sqpi_struct->addr_bit = 24U;

sqpi_struct->clk_div = 2U;

sqpi_struct->cmd_bit = QSPI_CMDBIT_8_BITS;

sqpi_init(&sqpi_init_struct);
```

### sqpi\_read\_id\_command

The description of sqpi\_read\_id\_command is shown as below:

**Table 3-879. Function sqpi\_read\_id\_command**

<b>Function name</b>	sqpi_read_id_command
<b>Function prototype</b>	void sqpi_read_id_command (void);
<b>Function descriptions</b>	send SQPI read ID command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* send SQPI read ID command */

sqpi_read_id_command ();
```

### sqpi\_special\_command

The description of sqpi\_special\_command is shown as below:

**Table 3-880. Function sqpi\_special\_command**

<b>Function name</b>	sqpi_special_command
<b>Function prototype</b>	void sqpi_special_command(void);
<b>Function descriptions</b>	send SQPI special command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

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

Example:

```
/* send SQPI special command */
```

```
sqpi_special_command ();
```

### sqpi\_read\_command\_config

The description of sqpi\_read\_command\_config is shown as below:

**Table 3-881. Function sqpi\_read\_command\_config**

<b>Function name</b>	sqpi_read_command_config
<b>Function prototype</b>	void sqpi_read_command_config(uint32_t rmode, uint32_t rwaitcycle, uint32_t rcmd);
<b>Function descriptions</b>	configure SQPI read command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rmode</b>	SQPI read command mode
QSPI_MODE_SSQ	SSQ mode
QSPI_MODE_SSS	SSS mode
QSPI_MODE_SQQ	SQQ mode
QSPI_MODE_QQQ	QQQ mode
QSPI_MODE_SSD	SSD mode
QSPI_MODE_SDD	SDD mode
<b>Input parameter{in}</b>	
<b>rwaitcycle</b>	SQPI read wait cycle (0x00 – 0x1F)
<b>Input parameter{in}</b>	
<b>rcmd</b>	SQPI read command(0x00 – 0xFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SQPI read command */
```

```
sqpi_read_command_config(QSPI_MODE_SSS,0x00,0x9f);
```

## sqpi\_write\_command\_config

The description of sqpi\_write\_command\_config is shown as below:

**Table 3-882. Function sqpi\_write\_command\_config**

<b>Function name</b>	sqpi_write_command_config
<b>Function prototype</b>	void sqpi_write_command_config(uint32_t wmode, uint32_t wwaitcycle, uint32_t wcmd);
<b>Function descriptions</b>	configure SQPI write command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wmode</b>	SQPI write command mode
QSPI_MODE_SSQ	SSQ mode
QSPI_MODE_SSS	SSS mode
QSPI_MODE_SQQ	SQQ mode
QSPI_MODE_QQQ	QQQ mode
QSPI_MODE_SSD	SSD mode
QSPI_MODE_SDD	SDD mode
<b>Input parameter{in}</b>	
<b>wwaitcycle</b>	SQPI write wait cycle (0x00 – 0x1F)
<b>Input parameter{in}</b>	
<b>wcmd</b>	SQPI write command(0x00 – 0xFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SQPI write command */
```

```
sqpi_write_command_config(QSPI_MODE_SSS,0x00,0x9f);
```

## sqpi\_low\_id\_receive

The description of sqpi\_low\_id\_receive is shown as below:

**Table 3-883. Function sqpi\_low\_id\_receive**

<b>Function name</b>	sqpi_low_id_receive
<b>Function prototype</b>	uint32_t sqpi_low_id_receive(void);
<b>Function descriptions</b>	get low ID value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-

Output parameter{out}	
-	-
Return value	
uint32_t	32-bit IDvalue (0-0xFFFF)

Example:

```
/* get SQPI low ID value */
```

```
uint32_t val;
```

```
val = sqpi_low_id_receive ();
```

### sqpi\_high\_id\_receive

The description of sqpi\_high\_id\_receive is shown as below:

**Table 3-884. Function sqpi\_low\_id\_receive**

Function name	sqpi_high_id_receive
Function prototype	uint32_t sqpi_high_id_receive(void);
Function descriptions	get high ID value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	32-bit ID value (0-0xFFFF)

Example:

```
/* get SQPI high ID value */
```

```
uint32_t val;
```

```
val = sqpi_high_id_receive ();
```

## 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 3-885. TIMERx Registers**

Registers	Descriptions
TIMER_CTL0	Control register 0
TIMER_CTL1	Control register 1
TIMER_SMCFG	Slave mode configuration register
TIMER_DMAINTEN	DMA and interrupt enable register
TIMER_INTF	Interrupt flag register
TIMER_SWEVG	Software event generation register
TIMER_CHCTL0	Channel control register 0
TIMER_CHCTL1	Channel control register 1
TIMER_CHCTL2	Channel control register 2
TIMER_CNT	Counter register
TIMER_PSC	Prescaler register
TIMER_CAR	Counter auto reload register
TIMER_CREP	Counter repetition register
TIMER_CH0CV	Channel 0 capture/compare value register
TIMER_CH1CV	Channel 1 capture/compare value register
TIMER_CH2CV	Channel 2 capture/compare value register
TIMER_CH3CV	Channel 3 capture/compare value register
TIMER_CCHP	Channel complementary protection register
TIMER_DMACFG	DMA configuration register
TIMER_DMATB	DMA transfer buffer register
TIMER_CFG	Configuration register

## 3.26.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

**Table 3-886. 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

Function name	Function description
timer_update_event_disable	disable the update event
timer_counter_alignment	set TIMER counter alignment mode
timer_counter_up_direction	set TIMER counter up direction
timer_counter_down_direction	set TIMER counter down direction
timer_prescaler_config	configure TIMER prescaler
timer_repetition_value_config	configure TIMER repetition register value
timer_autoreload_value_config	configure TIMER autoreload register value
timer_counter_value_config	configure TIMER counter register value
timer_counter_read	read TIMER counter value
timer_prescaler_read	read TIMER prescaler value
timer_single_pulse_mode_config	configure TIMER single pulse mode
timer_update_source_config	configure TIMER update source
timer_dma_enable	enable the TIMER DMA
timer_dma_disable	disable the TIMER DMA
timer_channel_dma_request_source_select	channel DMA request source selection
timer_dma_transfer_config	configure the TIMER DMA transfer
timer_event_software_generate	software generate events
timer_break_struct_para_init	initialize the parameters of TIMER break parameter struct with the default values
timer_break_config	configure TIMER break function
timer_break_enable	enable TIMER break function
timer_break_disable	disable TIMER break function
timer_automatic_output_enable	enable TIMER output automatic function
timer_automatic_output_disable	disable TIMER output automatic function
timer_primary_output_config	configure TIMER primary output function
timer_channel_control_shadow_config	channel capture/compare control shadow register enable
timer_channel_control_shadow_update_config	configure TIMER channel control shadow register update control
timer_channel_output_struct_para_init	initialize the parameters of TIMER channel output parameter struct with the default values
timer_channel_output_config	configure TIMER channel output function
timer_channel_output_mode_config	configure TIMER channel output compare mode
timer_channel_output_pulse_value_config	configure TIMER channel output pulse value
timer_channel_output_shadow_config	configure TIMER channel output shadow function
timer_channel_output_fast_config	configure TIMER channel output fast function
timer_channel_output_clear_config	configure TIMER channel output clear function
timer_channel_output_polarity_config	configure TIMER channel output polarity

Function name	Function description
timer_channel_complementary_output_polarity_config	configure TIMER channel complementary output polarity
timer_channel_output_state_config	configure TIMER channel enable state
timer_channel_complementary_output_state_config	configure TIMER channel complementary output enable state
timer_channel_input_struct_para_init	initialize the parameters of TIMER channel input parameter struct with the default values
timer_input_capture_config	configure TIMER input capture parameter
timer_channel_input_capture_prescaler_config	configure TIMER channel input capture prescaler value
timer_channel_capture_value_register_read	read TIMER channel capture compare register value
timer_input_pwm_capture_config	configure TIMER input pwm capture function
timer_hall_mode_config	configure TIMER hall sensor mode
timer_input_trigger_source_select	select TIMER input trigger source
timer_master_output_trigger_source_select	select TIMER master mode output trigger source
timer_slave_mode_select	select TIMER slave mode
timer_master_slave_mode_config	configure TIMER master slave mode
timer_external_trigger_config	configure TIMER external trigger input
timer_quadrature_decoder_mode_config	configure TIMER quadrature decoder mode
timer_internal_clock_config	configure TIMER internal clock mode
timer_internal_trigger_as_external_clock_config	configure TIMER the internal trigger as external clock input
timer_external_trigger_as_external_clock_config	configure TIMER the external trigger as external clock input
timer_external_clock_mode0_config	configure TIMER the external clock mode 0
timer_external_clock_mode1_config	configure TIMER the external clock mode 1
timer_external_clock_mode1_disable	disable TIMER the external clock mode 1
timer_write_chxval_register_config	configure TIMER write CHxVAL register selection
timer_output_value_selection_config	configure TIMER output value selection
timer_flag_get	get TIMER flags
timer_flag_clear	clear TIMER flags
timer_interrupt_enable	enable the TIMER interrupt
timer_interrupt_disable	disable the TIMER interrupt
timer_interrupt_flag_get	get timer interrupt flag
timer_interrupt_flag_clear	clear TIMER interrupt flag



## Structure timer\_parameter\_struct

**Table 3-887. Structure timer\_parameter\_struct**

Member name	Function description
prescaler	prescaler value (0~65535)
alignedmode	aligned mode (TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH)
counterdirection	counter direction (TIMER_COUNTER_UP, TIMER_COUNTER_DOWN)
period	period value
clockdivision	clock division value (TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4)
repetitioncounter	the counter repetition value (0~255)

## Structure timer\_break\_parameter\_struct

**Table 3-888. 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 3-889. Structure timer\_oc\_parameter\_struct**

Member name	Function description
outputstate	channel output state (TIMER_CCX_ENABLE, TIMER_CCX_DISABLE)
outputnstate	channel complementary output state (TIMER_CCXN_ENABLE, TIMER_CCXN_DISABLE)
ocpolarity	channel output polarity (TIMER_OC_POLARITY_HIGH, TIMER_OC_POLARITY_LOW)
ocnpolarity	channel complementary output polarity (TIMER_OCN_POLARITY_HIGH, TIMER_OCN_POLARITY_LOW)
ocidlestate	idle state of channel output (TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH)

Member name	Function description
ocnidlestate	idle state of channel complementary output (TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH)

## Structure timer\_ic\_parameter\_struct

**Table 3-890. Structure timer\_ic\_parameter\_struct**

Member name	Function description
icpolarity	channel input polarity (TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_FALLING, TIMER_IC_POLARITY_BOTH_EDGE)
icselection	channel input mode selection (TIMER_IC_SELECTION_DIRECTTI, TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS)
icprescaler	channel input capture prescaler (TIMER_IC_PSC_DIV1, TIMER_IC_PSC_DIV2, TIMER_IC_PSC_DIV4, TIMER_IC_PSC_DIV8)
icfilter	channel input capture filter control (0~15)

## timer\_deinit

The description of timer\_deinit is shown as below:

**Table 3-891. 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 3-892. Function timer\_struct\_para\_init**

<b>Function name</b>	timer_struct_para_init
<b>Function prototype</b>	void timer_struct_para_init(timer_parameter_struct* initpara);

<b>Function descriptions</b>	initialize the parameters of TIMER init parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>initpara</b>	TIMER init parameter struct, the structure members can refer to <a href="#">Structure timer parameter struct.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize TIMER init parameter struct with a default value */
```

```
timer_parameter_struct timer_initpara;
```

```
timer_struct_para_init(&timer_initpara);
```

## timer\_init

The description of timer\_init is shown as below:

**Table 3-893. 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="#">Structure timer parameter struct.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize TIMER0 */
```

```
timer_parameter_struct timer_initpara;
```

```
timer_initpara.prescaler = 107;
```

```

timer_initpara.alignedmode      = TIMER_COUNTER_EDGE;

timer_initpara.counterdirection = TIMER_COUNTER_UP;

timer_initpara.period          = 999;

timer_initpara.clockdivision    = TIMER_CKDIV_DIV1;

timer_initpara.repetitioncounter = 1;

timer_init(TIMERO,&timer_initpara);

```

## timer\_enable

The description of timer\_enable is shown as below:

**Table 3-894. Function timer\_enable**

<b>Function name</b>	timer_enable
<b>Function prototype</b>	void timer_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable a timer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMEx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* enable TIMERO */

timer_enable(TIMERO);

```

## timer\_disable

The description of timer\_disable is shown as below:

**Table 3-895. 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>TIMEx(x=0..13)</i>	TIMER peripheral selection

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 */
```

```
timer_disable(TIMER0);
```

## timer\_auto\_reload\_shadow\_enable

The description of timer\_auto\_reload\_shadow\_enable is shown as below:

**Table 3-896. Function timer\_auto\_reload\_shadow\_enable**

Function name	timer_auto_reload_shadow_enable
Function prototype	void timer_auto_reload_shadow_enable(uint32_t timer_periph);
Function descriptions	enable the auto reload shadow function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

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 3-897. Function timer\_auto\_reload\_shadow\_disable**

Function name	timer_auto_reload_shadow_disable
Function prototype	void timer_auto_reload_shadow_disable (uint32_t timer_periph);
Function descriptions	disable the auto reload shadow function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection

Output parameter{out}	
-	-
Return value	
-	-

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 3-898. Function timer\_update\_event\_enable**

Function name	timer_update_event_enable
Function prototype	void timer_update_event_enable(uint32_t timer_periph);
Function descriptions	enable the update event
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

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 3-899. Function timer\_update\_event\_disable**

Function name	timer_update_event_disable
Function prototype	void timer_update_event_disable(uint32_t timer_periph);
Function descriptions	disable the update event
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMERO the update event */
```

```
timer_update_event_disable (TIMERO);
```

## timer\_counter\_alignment

The description of timer\_counter\_alignment is shown as below:

**Table 3-900. 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)	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_CHCTL0 register). 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_CHCTL0 register). Only when the counter is counting up, compare interrupt flag of channels can be set.
TIMER_COUNTER_COUNTER_BOTH	Center-aligned and counting up/down assert mode. The counter counts under center-aligned and channel is configured in output mode (CHxMS=00 in TIMEx_CHCTL0 register). Both when the counter is counting up and counting down, compare interrupt flag of channels can be set.
Output parameter{out}	
-	-
Return value	
-	-

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 3-901. Function timer\_counter\_up\_direction**

<b>Function name</b>	timer_counter_up_direction
<b>Function prototype</b>	void timer_counter_up_direction(uint32_t timer_periph);
<b>Function descriptions</b>	set TIMER counter up direction
<b>Precondition</b>	set TIMER counter no center-aligned mode (edge-aligned mode)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set TIMER0 counter up direction */
```

```
timer_counter_up_direction(TIMER0);
```

## timer\_counter\_down\_direction

The description of timer\_counter\_down\_direction is shown as below:

**Table 3-902. timer\_counter\_down\_direction**

<b>Function name</b>	timer_counter_down_direction
<b>Function prototype</b>	void timer_counter_down_direction(uint32_t timer_periph);
<b>Function descriptions</b>	set TIMER counter down direction
<b>Precondition</b>	set TIMER counter no center-aligned mode (edge-aligned mode)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:



```
/* set TIMER0 counter down direction */
```

```
timer_counter_down_direction(TIMER0);
```

## timer\_prescaler\_config

The description of timer\_prescaler\_config is shown as below:

**Table 3-903. Function timer\_prescaler\_config**

<b>Function name</b>	timer_prescaler_config
<b>Function prototype</b>	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint32_t pscreload);
<b>Function descriptions</b>	configure TIMER prescaler
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..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 3-904. 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 3-905. 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 (0-0xFFFFFFFF)
<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 3-906. 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 (0-0xFFFFFFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 counter register value */
```

```
timer_counter_value_config(TIMER0);
```

## timer\_counter\_read

The description of timer\_counter\_read is shown as below:

**Table 3-907. 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(0~0xFFFFFFFF)

Example:

```
/* read TIMER0 counter value */
```

```
uint32_t i = 0;
```

```
i = timer_counter_read(TIMER0);
```

## timer\_prescaler\_read

The description of timer\_prescaler\_read is shown as below:

**Table 3-908. 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 3-909. 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

Output parameter{out}	
-	-
Return value	
-	-

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 3-910. Function timer\_update\_source\_config**

Function name	timer_update_source_config
Function prototype	void timer_update_source_config(uint32_t timer_periph, uint32_t update);
Function descriptions	configure TIMER update source
Precondition	-
The called functions	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Input parameter{in}	
<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.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER update only by counter overflow/underflow */
```

```
timer_update_source_config(TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

## timer\_dma\_enable

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

**Table 3-911. 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, $TIMERx(x=0..7)$
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable, $TIMERx(x=0..4,7)$
<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable, $TIMERx(x=0..4,7)$
<i>TIMER_DMA_CH2D</i>	channel 2 DMA enable, $TIMERx(x=0..4,7)$
<i>TIMER_DMA_CH3D</i>	channel 3 DMA enable, $TIMERx(x=0..4,7)$
<i>TIMER_DMA_CMTD</i>	commutation DMA request enable, $TIMERx(x=0,7)$
<i>TIMER_DMA_TRGD</i>	trigger DMA enable, $TIMERx(x=0..4,7)$
<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 3-912. 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

Input parameter{in}	
<b>dma</b>	timer DMA source disable
<i>TIMER_DMA_UPD</i>	update DMA disable, $TIMERx(x=0..7)$
<i>TIMER_DMA_CH0D</i>	channel 0 DMA disable, $TIMERx(x=0..4,7)$
<i>TIMER_DMA_CH1D</i>	channel 1 DMA disable, $TIMERx(x=0..4,7)$
<i>TIMER_DMA_CH2D</i>	channel 2 DMA disable, $TIMERx(x=0..4,7)$
<i>TIMER_DMA_CH3D</i>	channel 3 DMA disable, $TIMERx(x=0..4,7)$
<i>TIMER_DMA_CMTD</i>	commutation DMA request disable, $TIMERx(x=0,7)$
<i>TIMER_DMA_TRGD</i>	trigger DMA disable, $TIMERx(x=0..4,7)$
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER0 update DMA */
```

```
timer_dma_disable(TIMER0, TIMER_DMA_UPD);
```

## timer\_channel\_dma\_request\_source\_select

The description of timer\_channel\_dma\_request\_source\_select is shown as below:

**Table 3-913. 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>	-
Input parameter{in}	
<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 3-914. Function timer\_dma\_transfer\_config**

Function name	timer_dma_transfer_config
Function prototype	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
Function descriptions	configure the TIMER DMA transfer
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
dma_baseaddr	DMA transfer access start address
TIMER_DMACFG_DMA TA_CTL0	DMA transfer address is TIMER_CTL0, TIMERx(x=0..4,7)
TIMER_DMACFG_DMA TA_CTL1	DMA transfer address is TIMER_CTL1, TIMERx(x=0..4,7)
TIMER_DMACFG_DMA TA_SMCFG	DMA transfer address is TIMER_SMCFG, TIMERx(x=0..4,7)
TIMER_DMACFG_DMA TA_DMAINTEN	DMA transfer address is TIMER_DMAINTEN, TIMERx(x=0..4,7)
TIMER_DMACFG_DMA TA_INTF	DMA transfer address is TIMER_INTF, TIMERx(x=0..4,7)
TIMER_DMACFG_DMA TA_SWEVG	DMA transfer address is TIMER_SWEVG, TIMERx(x=0..4,7)
TIMER_DMACFG_DMA TA_CHCTL0	DMA transfer address is TIMER_CHCTL0, TIMERx(x=0..4,7)
TIMER_DMACFG_DMA TA_CHCTL1	DMA transfer address is TIMER_CHCTL1, TIMERx(x=0..4,7)
TIMER_DMACFG_DMA TA_CHCTL2	DMA transfer address is TIMER_CHCTL2, TIMERx(x=0..4,7)
TIMER_DMACFG_DMA TA_CNT	DMA transfer address is TIMER_CNT, TIMERx(x=0..4,7)
TIMER_DMACFG_DMA TA_PSC	DMA transfer address is TIMER_PSC, TIMERx(x=0..4,7)
TIMER_DMACFG_DMA	MA transfer address is TIMER_CAR, TIMERx(x=0..4,7)



<i>TA_CAR</i>	
<i>TIMER_DMACFG_DMA</i> <i>TA_CREP</i>	DMA transfer address is <i>TIMER_CREP</i> , <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH0CV</i>	DMA transfer address is <i>TIMER_CH0CV</i> , <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH1CV</i>	DMA transfer address is <i>TIMER_CH1CV</i> , <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH2CV</i>	DMA transfer address is <i>TIMER_CH2CV</i> , <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH3CV</i>	DMA transfer address is <i>TIMER_CH3CV</i> , <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CCHP</i>	DMA transfer address is <i>TIMER_CCHP</i> , <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMACFG_DMA</i> <i>TA_DMACFG</i>	DMA transfer address is <i>TIMER_DMACFG</i> , <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_DMACFG_DMA</i> <i>TA_DMATB</i>	DMA transfer address is <i>TIMER_DMATB</i> , <i>TIMERx</i> ( <i>x</i> =0..4,7)
<b>Input parameter{in}</b>	
<b><i>dma_lenth</i></b>	DMA transfer count
<i>TIMER_DMACFG_DMA</i> <i>TC_xTRANSFER</i>	<i>x</i> =1..18, DMA transfer <i>x</i> time
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the TIMER0 DMA transfer */
```

```
timer_dma_transfer_config(TIMER0, TIMER_DMACFG_DMATA_CTL0,  
TIMER_DMACFG_DMATC_5TRANSFER);
```

### timer\_event\_software\_generate

The description of `timer_event_software_generate` is shown as below:

**Table 3-915. Function `timer_event_software_generate`**

<b>Function name</b>	<code>timer_event_software_generate</code>
<b>Function prototype</b>	<code>void timer_event_software_generate(uint32_t timer_periph, uint16_t event);</code>
<b>Function descriptions</b>	software generate events
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>event</b>	the timer software event generation sources
<i>TIMER_EVENT_SRC_UPG</i>	update event, TIMERx(x=0..13)
<i>TIMER_EVENT_SRC_CH0G</i>	channel 0 capture or compare event generation, TIMERx(x=0..4,7..13)
<i>TIMER_EVENT_SRC_CH1G</i>	channel 1 capture or compare event generation, TIMERx(x=0..4,7,8,11)
<i>TIMER_EVENT_SRC_CH2G</i>	channel 2 capture or compare event generation, TIMERx(x=0..4,7)
<i>TIMER_EVENT_SRC_CH3G</i>	channel 3 capture or compare event generation, TIMERx(x=0..4,7)
<i>TIMER_EVENT_SRC_CMTG</i>	channel commutation event generation, TIMERx(x=0,7)
<i>TIMER_EVENT_SRC_TRIGG</i>	trigger event generation, TIMERx(x=0..4,7,8,11)
<i>TIMER_EVENT_SRC_BREAKG</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 3-916. Function timer\_break\_struct\_para\_init**

<b>Function name</b>	timer_break_struct_para_init
<b>Function prototype</b>	void timer_break_struct_para_init(timer_break_parameter_struct* breakpara);
<b>Function descriptions</b>	initialize the parameters of TIMER break parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>breakpara</b>	TIMER break parameter struct, the structure members can refer to

	<a href="#">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 3-917. Function timer\_break\_config**

<b>Function name</b>	timer_break_config
<b>Function prototype</b>	void timer_break_config(uint32_t timer_periph, timer_break_parameter_struct* breakpara);
<b>Function descriptions</b>	configure TIMER break function
<b>Precondition</b>	-
<b>The called functions</b>	-
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="#">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(TIMERO0, &timer_breakpara);

```

## timer\_break\_enable

The description of timer\_break\_enable is shown as below:

**Table 3-918. 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 TIMERO0 break function*/

timer_break_enable (TIMERO0);

```

## timer\_break\_disable

The description of timer\_break\_disable is shown as below:

**Table 3-919. Function timer\_break\_disable**

<b>Function name</b>	timer_break_disable
<b>Function prototype</b>	void timer_break_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER break function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* disable TIMER0 break function*/
```

```
timer_break_disable(TIMER0);
```

### timer\_automatic\_output\_enable

The description of timer\_automatic\_output\_enable is shown as below:

**Table 3-920. Function timer\_automatic\_output\_enable**

<b>Function name</b>	timer_automatic_output_enable
<b>Function prototype</b>	void timer_automatic_output_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable TIMER output automatic function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7)</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 3-921. Function timer\_automatic\_output\_disable**

<b>Function name</b>	timer_automatic_output_disable
<b>Function prototype</b>	void timer_automatic_output_disable (uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER output automatic function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral

<i>TIMERx(x=0,7)</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 3-922. 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 3-923. Function timer\_channel\_control\_shadow\_config**

<b>Function name</b>	timer_channel_control_shadow_config
<b>Function prototype</b>	void timer_channel_control_shadow_config(uint32_t timer_periph,

	ControlStatus newvalue);
<b>Function descriptions</b>	channel commutation control shadow register enable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* channel capture/compare control shadow register enable */
```

```
timer_channel_control_shadow_config(TIMER0, ENABLE);
```

## timer\_channel\_control\_shadow\_update\_config

The description of timer\_channel\_control\_shadow\_update\_config is shown as below:

**Table 3-924. Function timer\_channel\_control\_shadow\_update\_config**

<b>Function name</b>	timer_channel_control_shadow_update_config
<b>Function prototype</b>	void timer_channel_control_shadow_update_config(uint32_t timer_periph, uint32_t ccuctl);
<b>Function descriptions</b>	configure 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 3-925. Function timer\_channel\_output\_struct\_para\_init**

<b>Function name</b>	timer_channel_output_struct_para_init
<b>Function prototype</b>	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpa);
<b>Function descriptions</b>	initialize the parameters of TIMER channel output parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ocpara</b>	TIMER channel output parameter struct, the structure members can refer to <a href="#">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 3-926. 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="#">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 3-927. Function timer\_channel\_output\_mode\_config**

<b>Function name</b>	timer_channel_output_mode_config
<b>Function prototype</b>	void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint16_t ocmode);
<b>Function descriptions</b>	configure TIMER channel output compare mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral

<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<b>ocmode</b>	channel output compare mode
<i>TIMER_OC_MODE_TIMING</i>	timing mode
<i>TIMER_OC_MODE_ACTIVE</i>	set the channel output
<i>TIMER_OC_MODE_INACTIVE</i>	clear the channel output
<i>TIMER_OC_MODE_TOGGLE</i>	toggle on match
<i>TIMER_OC_MODE_LOW</i>	force low mode
<i>TIMER_OC_MODE_HIGH</i>	force high mode
<i>TIMER_OC_MODE_PWM0</i>	PWM mode 0
<i>TIMER_OC_MODE_PWM1</i>	PWM mode 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel PWM 0 mode */
```

```
timer_channel_output_mode_config(TIMER0, TIMER_CH_0, TIMER_OC_MODE_PWM0);
```

### timer\_channel\_output\_pulse\_value\_config

The description of timer\_channel\_output\_pulse\_value\_config is shown as below:

**Table 3-928. 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 (0~0xFFFFFFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 output pulse value */
```

```
timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

## timer\_channel\_output\_shadow\_config

The description of timer\_channel\_output\_shadow\_config is shown as below:

**Table 3-929. Function timer\_channel\_output\_shadow\_config**

<b>Function name</b>	timer_channel_output_shadow_config
<b>Function prototype</b>	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
<b>Function descriptions</b>	configure TIMER channel output shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	

<b>ocshadow</b>	channel output shadow state
<i>TIMER_OC_SHADOW_ENABLE</i>	channel output shadow state enable
<i>TIMER_OC_SHADOW_DISABLE</i>	channel output shadow state disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*configure TIMER0 channel 0 output shadow function */
```

```
timer_channel_output_shadow_config (TIMER0, TIMER_CH_0,  
TIMER_OC_SHADOW_ENABLE);
```

### timer\_channel\_output\_fast\_config

The description of timer\_channel\_output\_fast\_config is shown as below:

**Table 3-930. 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>	
-	-

Return value	
-	-

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 3-931. Function timer\_channel\_output\_clear\_config**

Function name	timer_channel_output_clear_config
Function prototype	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);
Function descriptions	configure TIMER channel output clear function
Precondition	-
The called functions	-
Input parameter{in}	
<b>timer_periph</b>	TIMER periphera
<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>	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
<b>occlear</b>	channel output clear function
<i>TIMER_OC_CLEAR_ENABLE</i>	channel output clear function enable
<i>TIMER_OC_CLEAR_DISABLE</i>	channel output clear function disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output clear function */
```

```
timer_channel_output_clear_config (TIMER0, TIMER_CH_0,  
TIMER_OC_CLEAR_ENABLE);
```

## timer\_channel\_output\_polarity\_config

The description of timer\_channel\_output\_polarity\_config is shown as below:

**Table 3-932. 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 3-933. 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 3-934. 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 3-935. 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>	
-	-



Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 complementary output enable state */
timer_channel_complementary_output_state_config(TIMER0, TIMER_CH_0,
TIMER_CCXN_ENABLE);
```

### timer\_channel\_input\_struct\_para\_init

The description of timer\_channel\_input\_struct\_para\_init is shown as below:

**Table 3-936. Function timer\_channel\_input\_struct\_para\_init**

Function name	timer_channel_input_struct_para_init
Function prototype	void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);
Function descriptions	initialize the parameters of TIMER channel input parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
icpara	TIMER channel input parameter struct, the structure members can refer to <a href="#">Structure timer_ic_parameter_struct</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER channel input parameter struct with a default value */
timer_ic_parameter_struct timer_icinitpara;
timer_channel_input_struct_para_init(&timer_icinitpara);
```

### timer\_input\_capture\_config

The description of timer\_input\_capture\_config is shown as below:

**Table 3-937. Function timer\_input\_capture\_config**

Function name	timer_input_capture_config
Function prototype	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);
Function descriptions	configure TIMER input capture parameter
Precondition	-

<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="#">Structure timer_ic_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure TIMER0 input capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);

```

## timer\_channel\_input\_capture\_prescaler\_config

The description of timer\_channel\_input\_capture\_prescaler\_config is shown as below:

**Table 3-938. 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

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>prescaler</b>	channel input capture prescaler value
<i>TIMER_IC_PSC_DIV1</i>	no prescaler
<i>TIMER_IC_PSC_DIV2</i>	divided by 2
<i>TIMER_IC_PSC_DIV4</i>	divided by 4
<i>TIMER_IC_PSC_DIV8</i>	divided by 8
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 input capture prescaler value */
```

```
timer_channel_input_capture_prescaler_config(TIMER0, TIMER_CH_0,  
TIMER_IC_PSC_DIV2);
```

## timer\_channel\_capture\_value\_register\_read

The description of timer\_channel\_capture\_value\_register\_read is shown as below:

**Table 3-939. 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>	-
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))
Output parameter{out}	

-	-
<b>Return value</b>	
uint32_t	channel capture compare register value (0~0xFFFFFFFF)

Example:

```
/* read TIMER0 channel 0 capture compare register value */
```

```
uint32_t ch0_value = 0;
```

```
ch0_value = timer_channel_capture_value_register_read(TIMER0, TIMER_CH_0);
```

### timer\_input\_pwm\_capture\_config

The description of timer\_input\_pwm\_capture\_config is shown as below:

**Table 3-940. Function timer\_input\_pwm\_capture\_config**

<b>Function name</b>	timer_input_pwm_capture_config
<b>Function prototype</b>	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
<b>Function descriptions</b>	configure TIMER input pwm capture function
<b>Precondition</b>	-
<b>The called functions</b>	timer_channel_input_capture_prescaler_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0
<i>TIMER_CH_1</i>	TIMER channel 1
<b>Input parameter{in}</b>	
<b>icpwm</b>	TIMER channel input pwm parameter struct, the structure members can refer to <a href="#">Structure timer_ic_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 input pwm capture parameter */
```

```
timer_ic_parameter_struct timer_icinitpara;
```

```
timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
```

```
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;
```

```
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
```

```
timer_icitpara.icfilter    = 0x0;
```

```
timer_input_pwm_capture_config(TIMERO, TIMER_CH_0, &timer_icitpara);
```

## timer\_hall\_mode\_config

The description of timer\_hall\_mode\_config is shown as below:

**Table 3-941. Function timer\_hall\_mode\_config**

<b>Function name</b>	timer_hall_mode_config
<b>Function prototype</b>	void timer_hall_mode_config(uint32_t timer_periph, uint32_t hallmode);
<b>Function descriptions</b>	configure TIMER hall sensor mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7)</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 TIMERO hall sensor mode */
```

```
timer_hall_mode_config (TIMERO, TIMER_HALLINTERFACE_ENABLE);
```

## timer\_input\_trigger\_source\_select

The description of timer\_input\_trigger\_source\_select is shown as below:

**Table 3-942. 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, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI1</i>	Internal trigger input 0 (ITI1, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI2</i>	Internal trigger input 0 (ITI2, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI3</i>	Internal trigger input 0 (ITI3, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOF_ED</i>	CIO edge flag (CIOF_ED, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOFE0</i>	channel 0 input Filtered output (CIOFE0, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_CI1FE1</i>	channel 1 input Filtered output (CI1FE1, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_ETIFP</i>	External trigger input filter output(ETIFP, <i>TIMERx</i> ( <i>x</i> =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 3-943. 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</i> ( <i>x</i> =0,7)	TIMER peripheral selection
<b>Input parameter{in}</b>	

outrigger	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_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 3-944. Function `timer_slave_mode_select`**

Function name	<code>timer_slave_mode_select</code>
Function prototype	<code>void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);</code>
Function descriptions	select TIMER slave mode
Precondition	-
The called functions	-
<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>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 3-945. Function timer\_master\_slave\_mode\_config**

<b>Function name</b>	timer_master_slave_mode_config
<b>Function prototype</b>	void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t masterslave);
<b>Function descriptions</b>	configure TIMER master slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection



Input parameter{in}	
<b>masterslave</b>	master slave mode state
<i>TIMER_MASTER_SLAVE_MODE_ENABLE</i>	master slave mode enable
<i>TIMER_MASTER_SLAVE_MODE_DISABLE</i>	master slave mode disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 master slave mode */
```

```
timer_master_slave_mode_config(TIMER0, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

## timer\_external\_trigger\_config

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

**Table 3-946. Function timer\_external\_trigger\_config**

<b>Function name</b>	timer_external_trigger_config
<b>Function prototype</b>	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER external trigger input
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
Input parameter{in}	
<b>extprescaler</b>	external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
Input parameter{in}	
<b>expolarity</b>	external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active

Input parameter{in}	
<b>extfilter</b>	external trigger filter control (0~15)
Output parameter{out}	
-	-
Return value	
-	-

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 3-947. Function timer\_quadrature\_decoder\_mode\_config**

<b>Function name</b>	timer_quadrature_decoder_mode_config
<b>Function prototype</b>	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity);
<b>Function descriptions</b>	configure TIMER quadrature decoder mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
Input parameter{in}	
<b>decomode</b>	quadrature decoder mode
<i>TIMER_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
Input parameter{in}	
<b>ic0polarity</b>	IC0 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	active both edge
Input parameter{in}	

<b>ic1polarity</b>	IC1 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	active both edge
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 quadrature decoder mode */
timer_quadrature_decoder_mode_config(TIMER0, TIMER_QUAD_DECODER_MODE0,
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

### timer\_internal\_clock\_config

The description of timer\_internal\_clock\_config is shown as below:

**Table 3-948. 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 3-949. 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 EL_ITI0</i>	Internal trigger input 0 (ITI0)
<i>TIMER_SMCFG_TRGS EL_ITI1</i>	Internal trigger input 0 (ITI1)
<i>TIMER_SMCFG_TRGS EL_ITI2</i>	Internal trigger input 0 (ITI2)
<i>TIMER_SMCFG_TRGS 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 3-950. Function timer\_external\_trigger\_as\_external\_clock\_config**

<b>Function name</b>	timer_external_trigger_as_external_clock_config
<b>Function prototype</b>	void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t extrigger, uint16_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external trigger as external clock input
<b>Precondition</b>	-
<b>The called functions</b>	timer_input_trigger_source_select
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection

Input parameter{in}	
<b>extrigger</b>	external trigger selection
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOF_ED</i>	CI0 edge flag (CIOF_ED)
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOFE0</i>	channel 0 input Filtered output (CIOFE0)
<i>TIMER_SMCFG_TRGS</i> <i>EL_C11FE1</i>	channel 1 input Filtered output (C11FE1)
Input parameter{in}	
<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
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	active both edge
Input parameter{in}	
<b>extfilter</b>	external trigger filter control (0~15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 the external trigger 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 3-951. 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
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
Input parameter{in}	

<b>extprescaler</b>	ETI external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
<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 3-952. Function timer\_external\_clock\_mode1\_config**

<b>Function name</b>	timer_external_clock_mode1_config
<b>Function prototype</b>	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external clock mode1
<b>Precondition</b>	-
<b>The called functions</b>	timer_external_trigger_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	ETI external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided

<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
<b>Input parameter{in}</b>	
<b>expolarity</b>	ETI external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
<b>Input parameter{in}</b>	
<b>extfilter</b>	ETI external trigger filter control (0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 0);
```

### timer\_external\_clock\_mode1\_disable

The description of timer\_external\_clock\_mode1\_disable is shown as below:

**Table 3-953. Function timer\_external\_clock\_mode1\_disable**

<b>Function name</b>	timer_external_clock_mode1_disable
<b>Function prototype</b>	void timer_external_clock_mode1_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER the external clock mode1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_disable(TIMER0);
```

## timer\_write\_chxval\_register\_config

The description of timer\_write\_chxval\_register\_config is shown as below:

**Table 3-954. 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 3-955. 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 3-956. Function timer\_flag\_get**

<b>Function name</b>	timer_flag_get
<b>Function prototype</b>	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);
<b>Function descriptions</b>	get TIMER flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>flag</b>	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag, TIMERx(x=0..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>	
-	-

Return value	
FlagStatus	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 3-957. 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(TIMERO, TIMER_FLAG_UP);
```

## timer\_interrupt\_enable

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

**Table 3-958. 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 TIMERO update interrupt */
```

```
timer_interrupt_enable (TIMERO, TIMER_INT_UP);
```

## timer\_interrupt\_disable

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

**Table 3-959. Function timer\_interrupt\_disable**

<b>Function name</b>	timer_interrupt_disable
<b>Function prototype</b>	void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	disable the TIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>interrupt</b>	timer interrupt disable source
<i>TIMER_INT_UP</i>	update interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..13)
<i>TIMER_INT_CH0</i>	channel 0 interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4,7..13)
<i>TIMER_INT_CH1</i>	channel 1 interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)
<i>TIMER_INT_CH2</i>	channel 2 interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_INT_CH3</i>	channel 3 interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_INT_CMT</i>	commutation interrupt disable, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_TRG</i>	trigger interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)
<i>TIMER_INT_BRK</i>	break interrupt disable, <i>TIMERx</i> ( <i>x</i> =0,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER0 update interrupt */
```

```
timer_interrupt_disable(TIMER0, TIMER_INT_UP);
```

## timer\_interrupt\_flag\_get

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

**Table 3-960. Function timer\_interrupt\_flag\_get**

<b>Function name</b>	timer_interrupt_flag_get
<b>Function prototype</b>	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t int_flag);
<b>Function descriptions</b>	get timer interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>int_flag</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..13)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7..13)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)

<i>TIMER_INT_FLAG_CM</i> <i>T</i>	channel commutation interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7,8,11)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TIMER0 update interrupt flag */
```

```
FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = timer_interrupt_flag_get(TIMER0, TIMER_INT_FLAG_UP);
```

### timer\_interrupt\_flag\_clear

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

**Table 3-961. Function timer\_interrupt\_flag\_clear**

<b>Function name</b>	timer_interrupt_flag_clear
<b>Function prototype</b>	void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t int_flag);
<b>Function descriptions</b>	clear TIMER interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>int_flag</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..13)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7..13)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_INT_FLAG_CM</i> <i>T</i>	channel commutation interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7,8,11)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TIMER0 update interrupt flag */

timer_interrupt_flag_clear(TIMER0, TIMER_INT_FLAG_UP);
```

## 3.27. TMU

The Trigonometric Math Unit (TMU) is a fully configurable block that execute common trigonometric and arithmetic operations. The TMU calculation unit can be used to calculate total 9 kinds of operations. The operation data must meet IEEE 32-Bit Single Precision Floating-Point Format. The TMU registers are listed in chapter [3.27.1](#), the TMU firmware functions are introduced in chapter [3.27.2](#).

### 3.27.1. Descriptions of Peripheral registers

TMU registers are listed in the table shown as below:

**Table 3-962. TMU Registers**

Registers	Descriptions
TMU_IDATA0	input data0 register
TMU_IDATA1	input data1 register
TMU_CTL	control register
TMU_DATA0	data0 register
TMU_DATA1	data1 register
TMU_STAT	status register

### 3.27.2. Descriptions of Peripheral functions

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

**Table 3-963. TMU firmware function**

Function name	Function description
tmu_deinit	reset the TMU
tmu_enable	enable the TMU
tmu_mode_set	configure the TMU mode
tmu_idata0_write	write the data to TMU input data0 register
tmu_idata1_write	write the data to TMU input data1 register
tmu_data0_read	read the data from TMU data0 register
tmu_data1_read	read the data from TMU data1 register
tmu_interrupt_enable	enable TTMU interrupt
tmu_interrupt_disable	disable TTMU interrupt
tmu_flag_get	check the TMU status flag
tmu_interrupt_flag_get	check the TMU interrupt flag

## tmu\_deinit

The description of tmu\_deinit is shown as below:

**Table 3-964. Function tmu\_deinit**

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

Example:

```
/* reset the TMU */
```

```
tmu_deinit();
```

## tmu\_enable

The description of tmu\_enable is shown as below:

**Table 3-965. Function tmu\_enable**

<b>Function name</b>	tmu_enable
<b>Function prototype</b>	void tmu_enable (void);
<b>Function descriptions</b>	enable the TMU
<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 TMU */
```

```
void tmu_enable(void);
```

## tmu\_mode\_set

The description of tmu\_mode\_set is shown as below:

**Table 3-966. Function tmu\_mode\_set**

<b>Function name</b>	tmu_mode_set
<b>Function prototype</b>	void tmu_mode_set(uint32_t modex);
<b>Function descriptions</b>	configure the TMU mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>modex</b>	the operation mode of TMU
TMU_MODE0	the operation mode0
TMU_MODE1	the operation mode1
TMU_MODE2	the operation mode2
TMU_MODE3	the operation mode3
TMU_MODE4	the operation mode4
TMU_MODE5	the operation mode5
TMU_MODE6	the operation mode6
TMU_MODE7	the operation mode7
TMU_MODE8	the operation mode8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the TMU mode0 */
tmu_mode_set(TMU_MODE0);
```

## tmu\_idata0\_write

The description of tmu\_idata0\_write is shown as below:

**Table 3-967. Function tmu\_idata0\_write**

<b>Function name</b>	tmu_idata0_write
<b>Function prototype</b>	void tmu_idata0_write(uint32_t idata0);
<b>Function descriptions</b>	write the data to TMU input data0 regisetr
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>idata0</b>	the value write to input data0
<b>Output parameter{out}</b>	
-	-



Return value	
-	-

Example:

```
/* write the idata0 register */
tmu_idata0_write (0XBF000000);
```

### tmu\_idata1\_write

The description of tmu\_idata1\_write is shown as below:

**Table 3-968. Function tmu\_idata1\_write**

Function name	tmu_idata1_write
Function prototype	void tmu_idata1_write(uint32_t idata1);
Function descriptions	write the data to TMU input data1 register
Precondition	-
The called functions	-
Input parameter{in}	
idata1	the value write to input data1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write the idata1 register */
tmu_idata1_write (0XBF000000);
```

### tmu\_data0\_read

The description of tmu\_data0\_read is shown as below:

**Table 3-969. Function tmu\_data0\_read**

Function name	tmu_data0_read
Function prototype	uint32_t tmu_data0_read(void);
Function descriptions	read the data from TMU data0 register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the value of data0 register meet IEEE 32-Bit Single Precision Floating-Point

Example:

```
/* read the data from TMU data0 regisetr*/
```

```
uint32_t tmu_value = 0;
```

```
tmu_value = tmu_data0_read();
```

### tmu\_data1\_read

The description of tmu\_data1\_read is shown as below:

**Table 3-970. Function tmu\_data1\_read**

<b>Function name</b>	tmu_data1_read
<b>Function prototype</b>	uint32_t tmu_data1_read(void);
<b>Function descriptions</b>	read the data from TMU data1 regisetr
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	the value of data1 register meet IEEE 32-Bit Single Precision Floating-Point Format

Example:

```
/* read the data from TMU data1 regisetr*/
```

```
uint32_t tmu_value = 0;
```

```
tmu_value = tmu_data1_read();
```

### tmu\_interrupt\_enable

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

**Table 3-971. Function tmu\_interrupt\_enable**

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

-	-
Return value	
-	-

Example:

```
/* enable TMU interrupt */
void tmu_interrupt_enable();
```

### tmu\_interrupt\_disable

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

**Table 3-972. Function tmu\_interrupt\_disable**

<b>Function name</b>	tmu_interrupt_disable
<b>Function prototype</b>	void tmu_interrupt_disable(void);
<b>Function descriptions</b>	disable TMU 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 TMU interrupt */
void tmu_interrupt_disable();
```

### tmu\_flag\_get

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

**Table 3-973. Function tmu\_flag\_get**

<b>Function name</b>	tmu_flag_get
<b>Function prototype</b>	FlagStatus tmu_flag_get(uint32_t flag);
<b>Function descriptions</b>	check the TMU status flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	the TMU status flag
TMU_FLAG_OVRF	the flag of TMU overflow
TMU_FLAG_UDRF	the flag of TMU underflow
<b>Output parameter{out}</b>	

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

Example:

```
/* check the TMU overflow flag */
tmu_flag_get(TMU_FLAG_OVRF);
```

### tmu\_interrupt\_flag\_get

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

**Table 3-974. Function tmu\_interrupt\_flag\_get**

<b>Function name</b>	tmu_interrupt_flag_get
<b>Function prototype</b>	FlagStatus tmu_interrupt_flag_get (uint32_t flag);
<b>Function descriptions</b>	check teh TMU interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	teh TMU interrupt flag
<i>TMU_INT_FLAG_CFIF</i>	the interrupt flag of calculation finished
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET/RESET

Example:

```
/* check teh TMU interrupt flag */
tmu_interrupt_flag_get(TMU_INT_FLAG_CFIF);
```

## 3.28. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter [3.28.1](#), the USART firmware functions are introduced in chapter [3.28.2](#).

### 3.28.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

**Table 3-975. USART Registers**

Registers	Descriptions
USART_STAT0	Status register 0 (USARTx, x=0..4)

Registers	Descriptions
USART_DATA	Data register (USARTx, x=0..4)
USART_BAUD	Baud rate register (USARTx, x=0..4)
USART_CTL0	Control register 0 (USARTx, x=0..4)
USART_CTL1	Control register 1 (USARTx, x=0..4)
USART_CTL2	Control register 2 (USARTx, x=0..4)
USART_GP	Guard time and prescaler register (USARTx, x=0..4)
USART_CTL3	Control register 3 (USARTx, x=0..4)
USART_RT	Receiver timeout register (USARTx, x=0..4)
USART_STAT1	Status register 1 (USARTx, x=0..4)
USART_GDCTL	GD control register (USARTx, x=0..4)
USART5_CTL0	Control register 0 (USARTx, x=5)
USART5_CTL1	Control register 1 (USARTx, x=5)
USART5_CTL2	Control register 2 (USARTx, x=5)
USART5_BAUD	Baud rate register (USARTx, x=5)
USART5_GP	Guard time and prescaler register (USARTx, x=5)
USART5_RT	Receiver timeout register (USARTx, x=5)
USART5_CMD	Command register (USARTx, x=5)
USART5_STAT	Status register (USARTx, x=5)
USART5_INTC	Interrupt status clear register (USARTx, x=5)
USART5_RDATA	Receive data register (USARTx, x=5)
USART5_TDATA	Transmit data register (USARTx, x=5)
USART5_CHC	Coherence control register (USARTx, x=5)
USART5_RFCFS	Receive FIFO control and status register (USARTx, x=5)

## 3.28.2. Descriptions of Peripheral functions

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

**Table 3-976. USART firmware function**

Function name	Function description
usart_deinit	reset USART/UART (USARTx, x=0..5)
usart_baudrate_set	configure USART baud rate value (USARTx, x=0..5)
usart_parity_config	configure USART parity (USARTx, x=0..5)
usart_word_length_set	configure USART word length (USARTx, x=0..5)
usart_stop_bit_set	configure USART stop bit length (USARTx, x=0..5)
usart_enable	enable USART (USARTx, x=0..5)
usart_disable	disable USART (USARTx, x=0..5)
usart_transmit_config	configure USART transmitter (USARTx, x=0..5)
usart_receive_config	configure USART receiver (USARTx, x=0..5)
usart_oversample_config	configure the USART oversample mode (USARTx, x=0..5)
usart_sample_bit_config	configure sample bit method (USARTx, x=0..5)
usart_receiver_timeout_enable	enable receiver timeout (USARTx, x=0..5)

Function name	Function description
usart_receiver_timeout_disable	disable receiver timeout (USARTx, x=0..5)
usart_receiver_timeout_threshold_config	configure receiver timeout threshold (USARTx, x=0..5)
usart_data_transmit	USART transmit data function (USARTx, x=0..5)
usart_data_receive	USART receive data function (USARTx, x=0..5)
usart_mute_mode_enable	enable mute mode (USARTx, x=0..5)
usart_mute_mode_disable	disable mute mode (USARTx, x=0..5)
usart_mute_mode_wakeup_config	configure wakeup method in mute mode (USARTx, x=0..5)
usart_lin_mode_enable	enable LIN mode (USARTx, x=0..5)
usart_lin_mode_disable	disable LIN mode (USARTx, x=0..5)
usart_lin_break_dection_length_config	configure LIN break frame length (USARTx, x=0..5)
usart_halfduplex_enable	enable half duplex mode (USARTx, x=0..5)
usart_halfduplex_disable	disable half duplex mode (USARTx, x=0..5)
usart_synchronous_clock_enable	enable CK pin in synchronous mode (USARTx, x=0..5)
usart_synchronous_clock_disable	disable CK pin in synchronous mode (USARTx, x=0..5)
usart_synchronous_clock_config	configure USART synchronous mode parameters (USARTx, x=0..5)
usart_guard_time_config	configure guard time value in smartcard mode (USARTx, x=0..5)
usart_smartcard_mode_enable	enable smartcard mode (USARTx, x=0..5)
usart_smartcard_mode_disable	disable smartcard mode (USARTx, x=0..5)
usart_smartcard_mode_nack_enable	enable NACK in smartcard mode (USARTx, x=0..5)
usart_smartcard_mode_nack_disable	disable NACK in smartcard mode (USARTx, x=0..5)
usart_smartcard_autoretry_config	configure smartcard auto-retry number (USARTx, x=0..5)
usart_block_length_config	configure block length (USARTx, x=0..5)
usart_irda_mode_enable	enable IrDA mode (USARTx, x=0..5)
usart_irda_mode_disable	disable IrDA mode (USARTx, x=0..5)
usart_prescaler_config	configure the peripheral clock prescaler in USART IrDA low-power mode (USARTx, x=0..5)
usart_irda_lowpower_config	configure IrDA low-power (USARTx, x=0..5)
usart_dma_receive_config	configure USART DMA reception (USARTx, x=0..5)
usart_dma_transmit_config	configure USART DMA transmission (USARTx, x=0..5)
usart_hardware_flow_rts_config	configure hardware flow control RTS (USARTx, x=0..4)
usart_hardware_flow_cts_config	configure hardware flow control CTS (USARTx, x=0..4)
usart_data_first_config	data is transmitted/received with the LSB/MSB first (USARTx, x=0..4)
usart_invert_config	configure USART inverted (USARTx, x=0..4)
usart_address_config	configure the address of the USART in wake up by address match mode (USARTx, x=0..4)
usart_send_break	send break frame (USARTx, x=0..4)
usart_collision_detected_interrupt_en	enable collision detected interrupt (USARTx, x=0..4)

Function name	Function description
able	
usart_collision_detected_interrupt_disable	disable collision detected interrupt (USARTx, x=0..4)
usart_collision_detection_enable	enable collision detection (USARTx, x=0..4)
usart_collision_detection_disable	disable collision detection (USARTx, x=0..4)
usart_flag_get	get flag in STAT0/STAT1 register (USARTx, x=0..4)
usart_flag_clear	clear flag in STAT0/STAT1 register (USARTx, x=0..4)
usart_interrupt_enable	enable USART interrupt (USARTx, x=0..4)
usart_interrupt_disable	disable USART interrupt (USARTx, x=0..4)
usart_interrupt_flag_get	get USART interrupt flag status (USARTx, x=0..4)
usart_interrupt_flag_clear	clear USART interrupt flag (USARTx, x=0..4)
usart5_data_first_config	data is transmitted/received with the LSB/MSB first (USARTx, x=5)
usart5_invert_config	configure USART5 inverted (USARTx, x=5)
usart5_overrun_enable	enable the USART5 overrun function (USARTx, x=5)
usart5_overrun_disable	disable the USART5 overrun function (USARTx, x=5)
usart5_address_config	configure address of the USART5 (USARTx, x=5)
usart5_address_detection_mode_config	configure address detection mode (USARTx, x=5)
usart5_smartcard_mode_early_nack_enable	enable early NACK in smartcard mode (USARTx, x=5)
usart5_smartcard_mode_early_nack_disable	disable early NACK in smartcard mode (USARTx, x=5)
usart5_reception_error_dma_enable	enable DMA on reception error (USARTx, x=5)
usart5_reception_error_dma_disable	disable DMA on reception error (USARTx, x=5)
usart5_wakeup_enable	enable USART to wakeup the mcu from deep-sleep mode (USARTx, x=5)
usart5_wakeup_disable	disable USART to wakeup the mcu from deep-sleep mode (USARTx, x=5)
usart5_wakeup_mode_config	configure the USART wakeup mode from deep-sleep mode (USARTx, x=5)
usart5_receive_fifo_enable	enable receive FIFO (USARTx, x=5)
usart5_receive_fifo_disable	disable receive FIFO (USARTx, x=5)
usart5_receive_fifo_counter_number	read receive FIFO counter number (USARTx, x=5)
usart5_flag_get	get flag in STAT/RFCs register (USARTx, x=5)
usart5_flag_clear	clear USART status (USARTx, x=5)
usart5_interrupt_enable	enable USART interrupt (USARTx, x=5)
usart5_interrupt_disable	disable USART interrupt (USARTx, x=5)
usart5_command_enable	enable USART command (USARTx, x=5)
usart5_interrupt_flag_get	get USART interrupt and flag status (USARTx, x=5)
usart5_interrupt_flag_clear	clear USART interrupt flag (USARTx, x=5)

## Enum usart\_flag\_enum

**Table 3-977. Enum usart\_flag\_enum**

Member name	Function description
USART_FLAG_CTS	CTS change flag
USART_FLAG_LBD	LIN break detected flag
USART_FLAG_TBE	transmit data buffer empty
USART_FLAG_TC	transmission complete
USART_FLAG_RBNE	read data buffer not empty
USART_FLAG_IDLE	IDLE line detected flag
USART_FLAG_ORERR	overrun error
USART_FLAG_NERR	noise error flag
USART_FLAG_FERR	frame error flag
USART_FLAG_PERR	parity error flag
USART_FLAG_BSY	busy flag
USART_FLAG_EB	end of block flag
USART_FLAG_RT	receiver timeout flag
USART_FLAG_CD	collision detected flag

## Enum usart5\_flag\_enum

**Table 3-978. Enum usart5\_flag\_enum**

Member name	Function description
USART5_FLAG_REA	receive enable acknowledge flag
USART5_FLAG_TEA	transmit enable acknowledge flag
USART5_FLAG_WU	wakeup from deep-sleep mode flag
USART5_FLAG_RWU	receiver wakeup from mute mode
USART5_FLAG_SB	send break flag
USART5_FLAG_AM	ADDR match flag
USART5_FLAG_BSY	busy flag
USART5_FLAG_EB	end of block flag
USART5_FLAG_RT	receiver timeout flag
USART5_FLAG_LBD	LIN break detected flag
USART5_FLAG_TBE	transmit data buffer empty
USART5_FLAG_TC	transmission complete
USART5_FLAG_RBNE	read data buffer not empty
USART5_FLAG_IDLE	IDLE line detected flag
USART5_FLAG_ORERR	overrun error
USART5_FLAG_NERR	noise error flag
USART5_FLAG_FERR	frame error flag
USART5_FLAG_PERR	parity error flag
USART5_FLAG_EPERR	early parity error flag
USART5_FLAG_RFFINT	receive FIFO full interrupt flag



Member name	Function description
USART5_FLAG_RFF	receive FIFO full flag
USART5_FLAG_RFE	receive FIFO empty flag

## Enum usart\_interrupt\_flag\_enum

**Table 3-979. Enum usart\_interrupt\_flag\_enum**

Member name	Function description
USART_INT_FLAG_PERR	parity error interrupt and flag
USART_INT_FLAG_TBE	transmitter buffer empty interrupt and flag
USART_INT_FLAG_TC	transmission complete interrupt and flag
USART_INT_FLAG_RBNE	read data buffer not empty interrupt and flag
USART_INT_FLAG_RBNE_ORE RR	read data buffer not empty interrupt and overrun error flag
USART_INT_FLAG_IDLE	IDLE line detected interrupt and flag
USART_INT_FLAG_LBD	LIN break detected interrupt and flag
USART_INT_FLAG_CTS	CTS interrupt and flag
USART_INT_FLAG_ERR_ORER R	error interrupt and overrun error
USART_INT_FLAG_ERR_NERR	error interrupt and noise error flag
USART_INT_FLAG_ERR_FERR	error interrupt and frame error flag
USART_INT_FLAG_EB	interrupt enable bit of end of block event and flag
USART_INT_FLAG_RT	interrupt enable bit of receive timeout event and flag
USART_INT_FLAG_CD	collision detected interrupt and flag

## Enum usart5\_interrupt\_flag\_enum

**Table 3-980. Enum usart5\_interrupt\_flag\_enum**

Member name	Function description
USART5_INT_FLAG_EB	end of block interrupt and flag
USART5_INT_FLAG_RT	receiver timeout interrupt and flag
USART5_INT_FLAG_AM	address match interrupt and flag
USART5_INT_FLAG_PERR	parity error interrupt and flag
USART5_INT_FLAG_TBE	transmitter buffer empty interrupt and flag
USART5_INT_FLAG_TC	transmission complete interrupt and flag
USART5_INT_FLAG_RBNE	read data buffer not empty interrupt and flag
USART5_INT_FLAG_RBNE_ORE RR	read data buffer not empty interrupt and overrun error flag
USART5_INT_FLAG_IDLE	IDLE line detected interrupt and flag
USART5_INT_FLAG_LBD	LIN break detected interrupt and flag
USART5_INT_FLAG_WU	wakeup from deep-sleep mode interrupt and flag
USART5_INT_FLAG_ERR_NERR	error interrupt and noise error flag
USART5_INT_FLAG_ERR_ORER	error interrupt and overrun error

Member name	Function description
R	
USART5_INT_FLAG_ERR_FERR	error interrupt and frame error flag
USART5_INT_FLAG_RFF	receive FIFO full interrupt and flag

## Enum usart\_interrupt\_enum

Table 3-981. Enum usart\_interrupt\_enum

Member name	Function description
USART_INT_PERR	parity error interrupt
USART_INT_TBE	transmitter buffer empty interrupt
USART_INT_TC	transmission complete interrupt
USART_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt
USART_INT_IDLE	IDLE line detected interrupt
USART_INT_LBD	LIN break detected interrupt
USART_INT_CTS	CTS interrupt
USART_INT_ERR	error interrupt
USART_INT_EB	interrupt enable bit of end of block event
USART_INT_RT	interrupt enable bit of receive timeout event
USART_INT_CD	collision detected interrupt

## Enum usart5\_interrupt\_enum

Table 3-982. Enum usart5\_interrupt\_enum

Member name	Function description
USART5_INT_EB	end of block interrupt
USART5_INT_RT	receiver timeout interrupt
USART5_INT_AM	address match interrupt
USART5_INT_PERR	parity error interrupt
USART5_INT_TBE	transmitter buffer empty interrupt
USART5_INT_TC	transmission complete interrupt
USART5_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt
USART5_INT_IDLE	IDLE line detected interrupt
USART5_INT_LBD	LIN break detected interrupt
USART5_INT_WU	wakeup from deep-sleep mode interrupt
USART5_INT_ERR	error interrupt
USART5_INT_RFF	receive FIFO full interrupt

## Enum usart\_invert\_enum

Table 3-983. Enum usart\_invert\_enum

Member name	Function description
USART_DINV_ENABLE	data bit level inversion
USART_DINV_DISABLE	data bit level not inversion

Member name	Function description
USART_TXPIN_ENABLE	TX pin level inversion
USART_TXPIN_DISABLE	TX pin level not inversion
USART_RXPIN_ENABLE	RX pin level inversion
USART_RXPIN_DISABLE	RX pin level not inversion

### Enum usart5\_invert\_enum

**Table 3-984. Enum usart5\_invert\_enum**

Member name	Function description
USART5_DINV_ENABLE	data bit level inversion
USART5_DINV_DISABLE	data bit level not inversion
USART5_TXPIN_ENABLE	TX pin level inversion
USART5_TXPIN_DISABLE	TX pin level not inversion
USART5_RXPIN_ENABLE	RX pin level inversion
USART5_RXPIN_DISABLE	RX pin level not inversion
USART5_SWAP_ENABLE	swap TX/RX pins
USART5_SWAP_DISABLE	not swap TX/RX pins

### usart\_deinit

The description of usart\_deinit is shown as below:

**Table 3-985. 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}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4
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 3-986. Function usart\_baudrate\_set**

<b>Function name</b>	usart_baudrate_set
<b>Function prototype</b>	void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);
<b>Function descriptions</b>	configure USART baud rate value
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>baudval</b>	baud rate value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 baud rate value */
usart_baudrate_set(USART0, 115200);
```

## usart\_parity\_config

The description of usart\_parity\_config is shown as below:

**Table 3-987. 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
USARTx	x=0,1,2,5
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>paritycfg</b>	configure USART parity
USART_PM_NONE	no parity
USART_PM_ODD	odd parity
USART_PM_EVEN	even parity

Output parameter{out}	
-	-
Return value	
-	-

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 3-988. Function usart\_word\_length\_set**

Function name	usart_word_length_set
Function prototype	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);
Function descriptions	configure USART word length
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4
Input parameter{in}	
wlen	USART word length
USART_WL_8BIT	8 bits
USART_WL_9BIT	9 bits
Output parameter{out}	
-	-
Return value	
-	-

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 3-989. Function usart\_stop\_bit\_set**

Function name	usart_stop_bit_set
Function prototype	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);
Function descriptions	configure USART stop bit length

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>stblen</b>	USART stop bit
<i>USART_STB_1BIT</i>	1 bit
<i>USART_STB_0_5BIT</i>	0.5 bit, not available for UARTx(x=3,4)
<i>USART_STB_2BIT</i>	2 bits
<i>USART_STB_1_5BIT</i>	1.5 bits, not available for UARTx(x=3,4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 stop bit length */
usart_stop_bit_set(USART0, USART_STB_1_5BIT);
```

### usart\_enable

The description of usart\_enable is shown as below:

**Table 3-990. Function usart\_enable**

<b>Function name</b>	usart_enable
<b>Function prototype</b>	void usart_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable USART
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 */
usart_enable(USART0);
```

## usart\_disable

The description of usart\_disable is shown as below:

**Table 3-991. Function usart\_disable**

<b>Function name</b>	usart_disable
<b>Function prototype</b>	void usart_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable USART
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 */
```

```
usart_disable(USART0);
```

## usart\_transmit\_config

The description of usart\_transmit\_config is shown as below:

**Table 3-992. Function usart\_transmit\_config**

<b>Function name</b>	usart_transmit_config
<b>Function prototype</b>	void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);
<b>Function descriptions</b>	configure USART transmitter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>txconfig</b>	enable or disable USART transmitter
<i>USART_TRANSMIT_ENABLE</i>	enable USART transmission
<i>USART_TRANSMIT_DISABLE</i>	disable USART transmission
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

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 3-993. 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
USARTx	x=0,1,2,5
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>rxconfig</b>	enable or disable USART receiver
USART_RECEIVE_ENABLE	enable USART reception
USART_RECEIVE_DISABLE	disable USART reception
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 receiver */
```

```
usart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

### usart\_oversample\_config

The description of usart\_oversample\_config is shown as below:

**Table 3-994. Function usart\_oversample\_config**

<b>Function name</b>	usart_oversample_config
<b>Function prototype</b>	void usart_oversample_config(uint32_t usart_periph, uint32_t oversamp);



<b>Function descriptions</b>	configure the USART oversample mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>oversamp</b>	oversample value
<i>USART_OVSMOD_8</i>	8 bits
<i>USART_OVSMOD_16</i>	16 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 oversample mode */
```

```
usart_oversample_config(USART0, USART_OVSMOD_8);
```

## usart\_sample\_bit\_config

The description of usart\_sample\_bit\_config is shown as below:

**Table 3-995. Function usart\_sample\_bit\_config**

<b>Function name</b>	usart_sample_bit_config
<b>Function prototype</b>	void usart_sample_bit_config(uint32_t usart_periph, uint32_t obsm);
<b>Function descriptions</b>	configure sample bit method
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>obsn</b>	sample bit
<i>USART_OSB_1bit</i>	1 bits
<i>USART_OSB_3bit</i>	3 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 sample bit */
```

```
usart_sample_bit_config(USART0, USART_OSB_1bit);
```

### usart\_receiver\_timeout\_enable

The description of usart\_receiver\_timeout\_enable is shown as below:

**Table 3-996. Function usart\_receiver\_timeout\_enable**

<b>Function name</b>	usart_receiver_timeout_enable
<b>Function prototype</b>	void usart_receiver_timeout_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable receiver timeout
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<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 3-997. 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 3-998. 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>	receiver timeout threshold (0x00000000 - 0x00FFFFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the receiver timeout threshold of USART0 */
```

```
usart_receiver_timeout_threshold_config(USART0, 115200*3);
```

## usart\_data\_transmit

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

**Table 3-999. 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
<b>Input parameter{in}</b>	
<b>data</b>	data of transmission

0-0xFF	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 3-1000. Function usart\_data\_receive**

<b>Function name</b>	usart_data_receive
<b>Function prototype</b>	void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);
<b>Function descriptions</b>	USART receive data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	data of received(0-0xFF)

Example:

```
/* USART0 receive data */
uint16_t temp;
temp = usart_data_receive(USART0);
```

### usart\_mute\_mode\_enable

The description of usart\_mute\_mode\_enable is shown as below:

**Table 3-1001. 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>	-

The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 receiver in mute mode */
```

```
usart_mute_mode_enable(USART0);
```

### usart\_mute\_mode\_disable

The description of usart\_mute\_mode\_disable is shown as below:

**Table 3-1002. Function usart\_mute\_mode\_disable**

Function name	usart_mute_mode_disable
Function prototype	void usart_mute_mode_disable(uint32_t usart_periph);
Function descriptions	disable mute mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4
Output parameter{out}	
-	-
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 3-1003. Function usart\_mute\_mode\_wakeup\_config**

Function name	usart_mute_mode_wakeup_config
Function prototype	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
<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 3-1004. 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
<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 3-1005. 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
<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 3-1006. 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
<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>	
-	-

Return value	
-	-

Example:

```
/* configure LIN break frame length */
```

```
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

### usart\_halfduplex\_enable

The description of usart\_halfduplex\_enable is shown as below:

**Table 3-1007. Function usart\_halfduplex\_enable**

Function name	usart_halfduplex_enable
Function prototype	void usart_halfduplex_enable(uint32_t usart_periph);
Function descriptions	enable half duplex mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 half duplex mode*/
```

```
usart_halfduplex_enable(USART0);
```

### usart\_halfduplex\_disable

The description of usart\_halfduplex\_disable is shown as below:

**Table 3-1008. Function usart\_halfduplex\_disable**

Function name	usart_halfduplex_disable
Function prototype	void usart_halfduplex_disable(uint32_t usart_periph);
Function descriptions	disable half duplex mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4



Output parameter{out}	
-	-
Return value	
-	-

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 3-1009. Function usart\_synchronous\_clock\_enable**

Function name	usart_synchronous_clock_enable
Function prototype	void usart_synchronous_clock_enable(uint32_t usart_periph);
Function descriptions	enable CK pin in synchronous mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

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 3-1010. Function usart\_synchronous\_clock\_disable**

Function name	usart_synchronous_clock_disable
Function prototype	void usart_synchronous_clock_disable(uint32_t usart_periph);
Function descriptions	disable CK pin in synchronous mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5

Output parameter{out}	
-	-
Return value	
-	-

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 3-1011. Function usart\_synchronous\_clock\_config**

Function name	usart_synchronous_clock_config
Function prototype	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
Function descriptions	configure USART synchronous mode parameters
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Input parameter{in}	
clen	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
Input parameter{in}	
cph	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
Input parameter{in}	
cpl	clock polarity
USART_CPL_LOW	steady low value on CK pin
USART_CPL_HIGH	steady high value on CK pin
Output parameter{out}	
-	-
Return value	
-	-

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 3-1012. Function usart\_guard\_time\_config**

<b>Function name</b>	usart_guard_time_config
<b>Function prototype</b>	void usart_guard_time_config(uint32_t usart_periph,uint32_t gaut);
<b>Function descriptions</b>	configure guard time value in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<b>Input parameter{in}</b>	
<b>guat</b>	guard time value (0x00000000 - 0x000000FF)
<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 3-1013. Function usart\_smartcard\_mode\_enable**

<b>Function name</b>	usart_smartcard_mode_enable
<b>Function prototype</b>	void usart_smartcard_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<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 3-1014. 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 3-1015. 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 3-1016. 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 3-1017. 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 (0x00000000 - 0x00000007)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* configure smartcard auto-retry number */
```

```
usart_smartcard_autoretry_config (USART0, 0x00000007);
```

### usart\_block\_length\_config

The description of usart\_block\_length\_config is shown as below:

**Table 3-1018. Function usart\_block\_length\_config**

<b>Function name</b>	usart_block_length_config
<b>Function prototype</b>	void usart_block_length_config(uint32_t usart_periph, uint32_t bl);
<b>Function descriptions</b>	configure block length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<b>Input parameter{in}</b>	
<b>bl</b>	block length (0x00000000 - 0x000000FF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure block length in Smartcard T=1 reception */
```

```
usart_block_length_config(USART0, 0x000000FF);
```

### usart\_irda\_mode\_enable

The description of usart\_irda\_mode\_enable is shown as below:

**Table 3-1019. 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

Output parameter{out}	
-	-
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 3-1020. 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
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 3-1021. 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

<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>psc</b>	0x00000000 - 0x000000FF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the USART0 peripheral clock prescaler */
```

```
usart_prescaler_config(USART0, 0x00000000);
```

### usart\_irda\_lowpower\_config

The description of usart\_irda\_lowpower\_config is shown as below:

**Table 3-1022. Function usart\_irda\_lowpower\_config**

<b>Function name</b>	usart_irda_lowpower_config
<b>Function prototype</b>	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
<b>Function descriptions</b>	configure IrDA low-power
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>irlp</b>	IrDA low-power or normal
<i>USART_IRLP_LOW</i>	low-power
<i>USART_IRLP_NORMAL</i>	normal
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 IrDA low-power */
```

```
usart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```



## usart\_dma\_receive\_config

The description of usart\_dma\_receive\_config is shown as below:

**Table 3-1023. Function usart\_dma\_receive\_config**

<b>Function name</b>	usart_dma_receive_config
<b>Function prototype</b>	void usart_dma_receive_config(uint32_t usart_periph, uint8_t dmacmd);
<b>Function descriptions</b>	configure USART DMA reception
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4
<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_DENR_ENABLE);
```

## usart\_dma\_transmit\_config

The description of usart\_dma\_transmit\_config is shown as below:

**Table 3-1024. Function usart\_dma\_transmit\_config**

<b>Function name</b>	usart_dma_transmit_config
<b>Function prototype</b>	void usart_dma_transmit_config(uint32_t usart_periph, uint8_t dmacmd);
<b>Function descriptions</b>	configure USART DMA transmission
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4
<b>Input parameter{in}</b>	

<b>dmacmd</b>	enable or disable DMA for transmission
<i>USART_TRANSMIT_DMA_ENABLE</i>	DMA enable for transmission
<i>USART_TRANSMIT_DMA_DISABLE</i>	DMA disable for transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 DMA enable for transmission */
```

```
usart_dma_transmit_config(USART0, USART_DENT_ENABLE);
```

### usart\_hardware\_flow\_rts\_config

The description of usart\_hardware\_flow\_rts\_config is shown as below:

**Table 3-1025. Function usart\_hardware\_flow\_rts\_config**

<b>Function name</b>	usart_hardware_flow_rts_config
<b>Function prototype</b>	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
<b>Function descriptions</b>	configure hardware flow control RTS
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>rtsconfig</b>	enable or disable RTS
<i>USART_RTS_ENABLE</i>	enable RTS
<i>USART_RTS_DISABLE</i>	disable RTS
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 hardware flow control RTS */
```

```
usart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);
```

## usart\_hardware\_flow\_cts\_config

The description of usart\_hardware\_flow\_cts\_config is shown as below:

**Table 3-1026. Function usart\_hardware\_flow\_cts\_config**

<b>Function name</b>	usart_hardware_flow_cts_config
<b>Function prototype</b>	void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
<b>Function descriptions</b>	configure hardware flow control RTS
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>ctsconfig</b>	enable or disable CTS
<i>USART_CTS_ENABLE</i>	enable CTS
<i>USART_CTS_DISABLE</i>	disable CTS
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 hardware flow control CTS */
```

```
usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

## usart\_data\_first\_config

The description of usart\_data\_first\_config is shown as below:

**Table 3-1027. Function usart\_data\_first\_config**

<b>Function name</b>	usart_data_first_config
<b>Function prototype</b>	void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);
<b>Function descriptions</b>	data is transmitted/received with the LSB/MSB first
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>msbf</b>	LSB first or MSB first
<i>USART_MSBF_LSB</i>	LSB first

<i>USART_MSBF_MSB</i>	MSB first
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LSB of data first */
```

```
usart_data_first_config(USART0, USART_MSBF_LSB);
```

## usart\_invert\_config

The description of usart\_invert\_config is shown as below:

**Table 3-1028. Function usart\_invert\_config**

<b>Function name</b>	usart_invert_config
<b>Function prototype</b>	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);
<b>Function descriptions</b>	configure USART inversion
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>invertpara</b>	refer to <a href="#">Table 3-983. Enum usart_invert_enum</a>
<i>USART_DINV_ENAB</i> <i>E</i>	data bit level inversion
<i>USART_DINV_DISAB</i> <i>E</i>	data bit level not inversion
<i>USART_TXPIN_ENAB</i> <i>LE</i>	TX pin level inversion
<i>USART_TXPIN_DISAB</i> <i>LE</i>	TX pin level not inversion
<i>USART_RXPIN_ENAB</i> <i>LE</i>	RX pin level inversion
<i>USART_RXPIN_DISAB</i> <i>LE</i>	RX pin level not inversion
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART inversion */
```

```
usart_invert_config(USART0, USART_DINV_ENABLE);
```

## usart\_address\_config

The description of usart\_address\_config is shown as below:

**Table 3-1029. Function usart\_address\_config**

<b>Function name</b>	usart_address_config
<b>Function prototype</b>	void usart_address_config(uint32_t usart_periph, uint8_t addr);
<b>Function descriptions</b>	configure the address of the USART in wake up by address match mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>addr</b>	address of USART/UART (0x00000000 - 0x0000000F)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure address of the USART0 */
```

```
usart_address_config(USART0, 0x00000000);
```

## usart\_send\_break

The description of usart\_send\_break is shown as below:

**Table 3-1030. 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
<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* USART0 send break frame */
```

```
usart_send_break(USART0);
```

## usart\_flag\_get

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

**Table 3-1031. Function usart\_flag\_get**

Function name	usart_flag_get
Function prototype	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
Function descriptions	get flag in STAT0/STAT1 register
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2
UARTx	x=3,4
Input parameter{in}	
flag	USART flags, refer to <a href="#">Table 3-977. Enum usart_flag_enum</a>
USART_FLAG_CTS	CTS change flag
USART_FLAG_LBD	LIN break detected flag
USART_FLAG_TBE	transmit data buffer empty
USART_FLAG_TC	transmission complete
USART_FLAG_RBNE	read data buffer not empty
USART_FLAG_IDLE	IDLE frame detected flag
USART_FLAG_ORER R	overrun error flag
USART_FLAG_NERR	noise error flag
USART_FLAG_FERR	frame error flag
USART_FLAG_PERR	parity error flag
USART_FLAG_BSY	busy flag
USART_FLAG_EB	end of block flag
USART_FLAG_RT	receiver timeout flag
USART_FLAG_CD	collision detected flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get flag USART0 state */
```

```
FlagStatus status;
```

```
status = usart_flag_get(USART0, USART_FLAG_TBE);
```

### usart\_flag\_clear

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

**Table 3-1032. 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
USARTx	x=0,1,2
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags, refer to <a href="#">Table 3-977. Enum usart_flag_enum</a>
USART_FLAG_CTS	CTS change flag
USART_FLAG_LBD	LIN break detected flag
USART_FLAG_TC	transmission complete
USART_FLAG_RBNE	read data buffer not empty
USART_FLAG_EB	end of block flag
USART_FLAG_RT	receiver timeout flag
USART_FLAG_CD	collision detected 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 3-1033. Function usart\_interrupt\_enable**

<b>Function name</b>	usart_interrupt_enable
<b>Function prototype</b>	void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum

	interrupt);
<b>Function descriptions</b>	enable USART interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>interrupt</b>	USART interrupt, refer to <a href="#">Table 3-981. Enum usart_interrupt_enum</a>
<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
<i>USART_INT_CD</i>	collision detected 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 3-1034. Function usart\_interrupt\_disable**

<b>Function name</b>	usart_interrupt_disable
<b>Function prototype</b>	void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);
<b>Function descriptions</b>	disable USART interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral



<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag, refer to <a href="#">Table 3-981. Enum usart_interrupt_enum</a>
<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
<i>USART_INT_CD</i>	collision detected 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 3-1035. Function usart\_interrupt\_flag\_get**

<b>Function name</b>	usart_interrupt_flag_get
<b>Function prototype</b>	FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	get USART interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag, refer to <a href="#">Table 3-979. Enum usart_interrupt_flag_enum</a>
<i>USART_INT_FLAG_PE</i>	parity error interrupt and flag

<i>RR</i>	
<i>USART_INT_FLAG_TBE</i>	transmitter buffer empty interrupt and flag
<i>USART_INT_FLAG_TC</i>	transmission complete interrupt and flag
<i>USART_INT_FLAG_RBNE</i>	read data buffer not empty interrupt and flag
<i>USART_INT_FLAG_RBNE_ORERR</i>	read data buffer not empty interrupt and overrun error flag
<i>USART_INT_FLAG_IDLE</i>	IDLE line detected interrupt and flag
<i>USART_INT_FLAG_LBD</i>	LIN break detected interrupt and flag
<i>USART_INT_FLAG_CTS</i>	CTS interrupt and flag
<i>USART_INT_FLAG_ERORERR</i>	error interrupt and overrun error
<i>USART_INT_FLAG_EROR_NERR</i>	error interrupt and noise error flag
<i>USART_INT_FLAG_EROR_FERR</i>	error interrupt and frame error flag
<i>USART_INT_FLAG_EB</i>	end of block event interrupt flag
<i>USART_INT_FLAG_RT</i>	receive timeout event interrupt flag
<i>USART_INT_FLAG_CD</i>	collision detected interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the USART0 interrupt flag status */
```

```
FlagStatus status;
```

```
status = usart_interrupt_flag_get(USART0, USART_INT_FLAG_RBNE);
```

## usart\_interrupt\_flag\_clear

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

**Table 3-1036. Function usart\_interrupt\_flag\_clear**

<b>Function name</b>	usart_interrupt_flag_clear
<b>Function prototype</b>	void usart_interrupt_flag_clear(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	clear USART interrupt flag

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag, refer to <a href="#">Table 3-979. Enum usart_interrupt_flag_enum</a>
<i>USART_INT_FLAG_CTS</i>	CTS interrupt and flag
<i>USART_INT_FLAG_LBD</i>	LIN break detected interrupt and flag
<i>USART_INT_FLAG_TC</i>	transmission complete interrupt and flag
<i>USART_INT_FLAG_RBNE</i>	read data buffer not empty interrupt and flag
<i>USART_INT_FLAG_EB</i>	interrupt enable bit of end of block event and flag
<i>USART_INT_FLAG_RT</i>	interrupt enable bit of receive timeout event and flag
<i>USART_INT_FLAG_CD</i>	collision detected interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the USART0 interrupt enable bit status */
```

```
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

## usart5\_data\_first\_config

The description of usart\_data\_first\_config is shown as below:

**Table 3-1037. Function usart5\_data\_first\_config**

<b>Function name</b>	usart5_data_first_config
<b>Function prototype</b>	void usart5_data_first_config(uint32_t usart_periph, uint32_t msbf);
<b>Function descriptions</b>	data is transmitted/received with the LSB/MSB first
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=5
<b>Input parameter{in}</b>	

<b>msbf</b>	LSB/MSB
<i>USART5_MSBF_LSB</i>	LSB first
<i>USART5_MSBF_MSB</i>	MSB first
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LSB of data first */
```

```
usart5_data_first_config(USART5, USART5_MSBF_LSB);
```

### usart5\_invert\_config

The description of usart5\_invert\_config is shown as below:

**Table 3-1038. Function usart5\_invert\_config**

<b>Function name</b>	usart5_invert_config
<b>Function prototype</b>	void usart5_invert_config(uint32_t usart_periph, usart5_invert_enum invertpara);
<b>Function descriptions</b>	configure USART5 inverted
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=5
<b>Input parameter{in}</b>	
<b>invertpara</b>	refer to <a href="#">Table 3-984. Enum usart5_invert_enum</a>
<i>USART5_DINV_ENABLE</i>	data bit level inversion
<i>USART5_DINV_DISABLE</i>	data bit level not inversion
<i>USART5_TXPIN_ENABLE</i>	TX pin level inversion
<i>USART5_TXPIN_DISABLE</i>	TX pin level not inversion
<i>USART5_RXPIN_ENABLE</i>	RX pin level inversion
<i>USART5_RXPIN_DISABLE</i>	RX pin level not inversion
<i>USART5_SWAP_ENABLE</i>	swap TX/RX pins
<i>USART5_SWAP_DISABLE</i>	not swap TX/RX pins

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

Example:

```
/* configure USART5 inversion */
usart5_invert_config(USART5, USART5_DINV_ENABLE);
```

## usart5\_overshoot\_enable

The description of usart5\_overshoot\_enable is shown as below:

**Table 3-1039. Function usart5\_overshoot\_enable**

<b>Function name</b>	usart5_overshoot_enable
<b>Function prototype</b>	void usart5_overshoot_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable the USART5 overshoot function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART5 overshoot */
usart5_overshoot_enable(USART5);
```

## usart5\_overshoot\_disable

The description of usart5\_overshoot\_disable is shown as below:

**Table 3-1040. Function usart5\_overshoot\_disable**

<b>Function name</b>	usart5_overshoot_disable
<b>Function prototype</b>	void usart5_overshoot_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable the USART5 overshoot function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral

USARTx	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART5 overrun */
usart5_overrun_disable(USART5);
```

### usart5\_address\_config

The description of usart5\_address\_config is shown as below:

**Table 3-1041. Function usart5\_address\_config**

Function name	usart5_address_config
Function prototype	void usart5_address_config(uint32_t usart_periph, uint8_t addr);
Function descriptions	configure the address of the USART5 terminal
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=5
Input parameter{in}	
addr	address of USART terminal (0x00000000-0x000000FF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure address of the USART5 */
usart5_address_config(USART5, 0x00000000);
```

### usart5\_address\_detection\_mode\_config

The description of usart5\_address\_detection\_mode\_config is shown as below:

**Table 3-1042. Function usart5\_address\_detection\_mode\_config**

Function name	usart5_address_detection_mode_config
Function prototype	void usart5_address_detection_mode_config(uint32_t usart_periph, uint32_t addmod);
Function descriptions	configure address detection mode
Precondition	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=5
<b>Input parameter{in}</b>	
<b>addmod</b>	address detection mode
<i>USART5_ADDM_4BIT</i>	4 bits
<i>USART5_ADDM_FULL BIT</i>	full bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*configure address detection mode */
```

```
usart5_address_detection_mode_config(USART5, USART5_ADDM_4BIT);
```

## usart5\_smartcard\_mode\_early\_nack\_enable

The description of usart5\_smartcard\_mode\_early\_nack\_enable is shown as below:

**Table 3-1043. Function usart5\_smartcard\_mode\_early\_nack\_enable**

<b>Function name</b>	usart5_smartcard_mode_early_nack_enable
<b>Function prototype</b>	void usart5_smartcard_mode_early_nack_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable early NACK in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART5 early NACK in smartcard mode */
```

```
usart5_smartcard_mode_early_nack_enable(USART5);
```

## usart5\_smartcard\_mode\_early\_nack\_disable

The description of usart5\_smartcard\_mode\_early\_nack\_disable is shown as below:

**Table 3-1044. Function usart5\_smartcard\_mode\_early\_nack\_disable**

<b>Function name</b>	usart5_smartcard_mode_early_nack_disable
<b>Function prototype</b>	void usart5_smartcard_mode_early_nack_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable early NACK in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART5 early NACK in smartcard mode */
usart5_smartcard_mode_early_nack_disable(USART5);
```

### usart5\_reception\_error\_dma\_enable

The description of usart5\_reception\_error\_dma\_enable is shown as below:

**Table 3-1045. Function usart5\_reception\_error\_dma\_enable**

<b>Function name</b>	usart5_reception_error_dma_enable
<b>Function prototype</b>	void usart5_reception_error_dma_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable DMA on reception error
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMA on reception error */
usart5_reception_error_dma_enable(USART5);
```

### usart5\_reception\_error\_dma\_disable

The description of usart5\_reception\_error\_dma\_disable is shown as below:



**Table 3-1046. Function usart5\_reception\_error\_dma\_disable**

<b>Function name</b>	usart5_reception_error_dma_disable
<b>Function prototype</b>	void usart5_reception_error_dma_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable DMA on reception error
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA on reception error */
usart5_reception_error_dma_disable(USART5);
```

## usart5\_wakeup\_enable

The description of usart5\_wakeup\_enable is shown as below:

**Table 3-1047. Function usart5\_wakeup\_enable**

<b>Function name</b>	usart5_wakeup_enable
<b>Function prototype</b>	void usart5_wakeup_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable USART to wakeup the mcu from deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART5 wake up enable */
usart5_wakeup_enable(USART5);
```

## usart5\_wakeup\_disable

The description of usart5\_wakeup\_disable is shown as below:

Table 3-1048. Function usart5\_wakeup\_disable

Function name	usart5_wakeup_disable
Function prototype	void usart5_wakeup_disable(uint32_t usart_periph);
Function descriptions	disable USART to wakeup the mcu from deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART5 wake up disable */
```

```
usart5_wakeup_disable(USART5);
```

### usart5\_wakeup\_mode\_config

The description of usart5\_wakeup\_mode\_config is shown as below:

Table 3-1049. Function usart5\_wakeup\_mode\_config

Function name	usart5_wakeup_mode_config
Function prototype	void usart5_wakeup_mode_config(uint32_t usart_periph, uint32_t wum);
Function descriptions	configure the USART wakeup mode from deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=5
Input parameter{in}	
wum	wakeup mode
USART5_WUM_ADDR	WUF active on address match
USART5_WUM_STAR TB	WUF active on start bit
USART5_WUM_RBNE	WUF active on RBNE
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART5 wake up mode */
```

```
usart5_wakeup_mode_config(USART5, USART5_WUM_ADDR);
```

## usart5\_receive\_fifo\_enable

The description of usart5\_receive\_fifo\_enable is shown as below:

**Table 3-1050. Function usart5\_receive\_fifo\_enable**

<b>Function name</b>	usart5_receive_fifo_enable
<b>Function prototype</b>	void usart5_receive_fifo_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable receive FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable receive FIFO */
```

```
usart5_receive_fifo_enable(USART5);
```

## usart5\_receive\_fifo\_disable

The description of usart5\_receive\_fifo\_disable is shown as below:

**Table 3-1051. Function usart5\_receive\_fifo\_disable**

<b>Function name</b>	Usart5_receive_fifo_disable
<b>Function prototype</b>	void usart5_receive_fifo_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable receive FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable receive FIFO */
```

```
usart5_receive_fifo_disable(USART5);
```

### usart5\_receive\_fifo\_counter\_number

The description of usart5\_receive\_fifo\_counter\_number is shown as below:

**Table 3-1052. Function usart5\_receive\_fifo\_counter\_number**

<b>Function name</b>	usart5_receive_fifo_counter_number
<b>Function prototype</b>	uint8_t usart5_receive_fifo_counter_number(uint32_t usart_periph);
<b>Function descriptions</b>	read receive FIFO counter number
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint8_t	receive FIFO counter number

Example:

```
/* read receive FIFO counter number */
```

```
uint8_t temp;
```

```
temp = usart5_receive_fifo_counter_number(USART5);
```

### usart5\_flag\_get

The description of usart5\_flag\_get is shown as below:

**Table 3-1053. Function usart5\_flag\_get**

<b>Function name</b>	usart5_flag_get
<b>Function prototype</b>	FlagStatus usart5_flag_get(uint32_t usart_periph, usart5_flag_enum flag);
<b>Function descriptions</b>	get flag in STAT/CHC/RFCs register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=5
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags, refer to <a href="#">Table 3-978. Enum usart5_flag_enum</a> only one among these parameters can be selected
USART5_FLAG_PERR	parity error flag

<code>USART5_FLAG_FERR</code>	frame error flag
<code>USART5_FLAG_NERR</code>	noise error flag
<code>USART5_FLAG_ORER</code> <code>R</code>	overrun error
<code>USART5_FLAG_IDLE</code>	idle line detected flag
<code>USART5_FLAG_RBNE</code>	read data buffer not empty
<code>USART5_FLAG_TC</code>	transmission completed
<code>USART5_FLAG_TBE</code>	transmit data register empty
<code>USART5_FLAG_LBD</code>	LIN break detected flag
<code>USART5_FLAG_RT</code>	receiver timeout flag
<code>USART5_FLAG_EB</code>	end of block flag
<code>USART5_FLAG_BSY</code>	busy flag
<code>USART5_FLAG_AM</code>	address match flag
<code>USART5_FLAG_SB</code>	send break flag
<code>USART5_FLAG_RWU</code>	receiver wakeup from mute mode
<code>USART5_FLAG_WU</code>	wakeup from deep-sleep mode flag
<code>USART5_FLAG_TEA</code>	transmit enable acknowledge flag
<code>USART5_FLAG_REA</code>	receive enable acknowledge flag
<code>USART5_FLAG_EPER</code> <code>R</code>	early parity error flag
<code>USART5_FLAG_RFE</code>	receive FIFO empty flag
<code>USART5_FLAG_RFF</code>	receive FIFO full flag
<code>USART5_FLAG_RFFIN</code> <code>T</code>	receive FIFO full interrupt flag
<b>Output</b> <b>parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get flag USART5 state */
```

```
FlagStatus status;
```

```
status = usart5_flag_get(USART5, USART5_FLAG_TBE);
```

## **usart5\_flag\_clear**

The description of `usart5_flag_clear` is shown as below:

**Table 3-1054. Function `usart5_flag_clear`**

<b>Function name</b>	<code>usart5_flag_clear</code>
----------------------	--------------------------------

<b>Function prototype</b>	void usart5_flag_clear(uint32_t usart_periph, usart5_flag_enum flag);
<b>Function descriptions</b>	clear flag in STAT register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=5
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags, refer to <a href="#">Table 3-978. Enum usart5_flag_enum</a> only one among these parameters can be selected
USART5_FLAG_PERR	parity error flag
USART5_FLAG_FERR	frame error flag
USART5_FLAG_NERR	noise detected flag
USART5_FLAG_ORER R	overrun error flag
USART5_FLAG_IDLE	idle line detected flag
USART5_FLAG_TC	transmission complete flag
USART5_FLAG_LBD	LIN break detected flag
USART5_FLAG_RT	receiver timeout flag
USART5_FLAG_EB	end of block flag
USART5_FLAG_AM	address match flag
USART5_FLAG_WU	wakeup from deep-sleep mode flag
USART5_FLAG_EPER R	early parity error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear USART5 flag */
```

```
usart5_flag_clear(USART5, USART5_FLAG_TC);
```

### usart5\_interrupt\_enable

The description of usart5\_interrupt\_enable is shown as below:

**Table 3-1055. Function usart5\_interrupt\_enable**

<b>Function name</b>	usart5_interrupt_enable
<b>Function prototype</b>	void usart5_interrupt_enable(uint32_t usart_periph, usart5_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>	usart peripheral
<i>USARTx</i>	x=5
<b>Input parameter{in}</b>	
<b>interrupt</b>	USART5 interrupts, refer to <a href="#">Table 3-982. Enum usart5_interrupt_enum</a>
<i>USART5_INT_IDLE</i>	idle interrupt
<i>USART5_INT_RBNE</i>	read data buffer not empty interrupt and overrun error interrupt enable interrupt
<i>USART5_INT_TC</i>	transmission complete interrupt
<i>USART5_INT_TBE</i>	transmit data register empty interrupt
<i>USART5_INT_PERR</i>	parity error interrupt
<i>USART5_INT_AM</i>	address match interrupt
<i>USART5_INT_RT</i>	receiver timeout interrupt
<i>USART5_INT_EB</i>	end of block interrupt
<i>USART5_INT_LBD</i>	LIN break detection interrupt
<i>USART5_INT_ERR</i>	error interrupt enable in multibuffer communication
<i>USART5_INT_WU</i>	wakeup from deep-sleep mode interrupt
<i>USART5_INT_RFF</i>	receive FIFO full interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART5 TBE interrupt */
```

```
usart5_interrupt_enable(USART5, USART5_INT_TBE);
```

## usart5\_interrupt\_disable

The description of usart5\_interrupt\_disable is shown as below:

**Table 3-1056. Function usart5\_interrupt\_disable**

<b>Function name</b>	usart5_interrupt_disable
<b>Function prototype</b>	void usart5_interrupt_disable(uint32_t usart_periph, usart5_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>	usart peripheral
<i>USARTx</i>	x=5
<b>Input parameter{in}</b>	

interrupt	USART5 interrupts, refer to <a href="#">Table 3-982. Enum usart5_interrupt_enum</a>
USART5_INT_IDLE	idle interrupt
USART5_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt enable interrupt
USART5_INT_TC	transmission complete interrupt
USART5_INT_TBE	transmit data register empty interrupt
USART5_INT_PERR	parity error interrupt
USART5_INT_AM	address match interrupt
USART5_INT_RT	receiver timeout interrupt
USART5_INT_EB	end of block interrupt
USART5_INT_LBD	LIN break detection interrupt
USART5_INT_ERR	error interrupt enable in multibuffer communication
USART5_INT_WU	wakeup from deep-sleep mode interrupt
USART5_INT_RFF	receive FIFO full interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART5 TBE interrupt */
```

```
usart5_interrupt_disable(USART5, USART5_INT_TBE);
```

### usart5\_command\_enable

The description of usart5\_command\_enable is shown as below:

**Table 3-1057. Function usart5\_command\_enable**

Function name	usart5_command_enable
Function prototype	void usart5_command_enable(uint32_t usart_periph, uint32_t cmdtype);
Function descriptions	enable USART command
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=5
Input parameter{in}	
cmdtype	command type
USART5_CMD_SBKC MD	send break command
USART5_CMD_MMCM D	mute mode command
USART5_CMD_RXFC	receive data flush command



<i>MD</i>	
<i>USART5_CMD_TXFC</i> <i>MD</i>	transmit data flush request
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART5 command */
```

```
usart5_command_enable(USART5, USART5_CMD_SBKCMD);
```

### usart5\_interrupt\_flag\_get

The description of usart5\_interrupt\_flag\_get is shown as below:

**Table 3-1058. Function usart5\_interrupt\_flag\_get**

<b>Function name</b>	usart5_interrupt_flag_get
<b>Function prototype</b>	FlagStatus usart5_interrupt_flag_get(uint32_t usart_periph, usart5_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>	usart peripheral
<i>USARTx</i>	x=5
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag, refer to <a href="#">Table 3-980. Enum usart5_interrupt_flag_enum</a>
<i>USART5_INT_FLAG_EB</i>	end of block interrupt and flag
<i>USART5_INT_FLAG_RT</i>	receiver timeout interrupt and flag
<i>USART5_INT_FLAG_AM</i>	address match interrupt and flag
<i>USART5_INT_FLAG_PERR</i>	parity error interrupt and flag
<i>USART5_INT_FLAG_TBE</i>	transmitter buffer empty interrupt and flag
<i>USART5_INT_FLAG_TC</i>	transmission complete interrupt and flag
<i>USART5_INT_FLAG_RBNE</i>	read data buffer not empty interrupt and flag

<i>USART5_INT_FLAG_RBNE_ORERR</i>	read data buffer not empty interrupt and overrun error flag
<i>USART5_INT_FLAG_IDLE</i>	IDLE line detected interrupt and flag
<i>USART5_INT_FLAG_LIN_BD</i>	LIN break detected interrupt and flag
<i>USART5_INT_FLAG_WU</i>	wakeup from deep-sleep mode interrupt and flag
<i>USART5_INT_FLAG_ERR_NERR</i>	error interrupt and noise error flag
<i>USART5_INT_FLAG_ERR_ORERR</i>	error interrupt and overrun error
<i>USART5_INT_FLAG_ERR_FERR</i>	error interrupt and frame error flag
<i>USART5_INT_FLAG_RX_FF</i>	receive FIFO full interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the USART5 interrupt flag status */
```

```
FlagStatus status;
```

```
status = usart5_interrupt_flag_get(USART5, USART5_INT_FLAG_RBNE);
```

## usart5\_interrupt\_flag\_clear

The description of usart5\_interrupt\_flag\_clear is shown as below:

**Table 3-1059. Function usart5\_interrupt\_flag\_clear**

<b>Function name</b>	usart5_interrupt_flag_clear
<b>Function prototype</b>	void usart5_interrupt_flag_clear(uint32_t usart_periph, usart5_interrupt_flag_enum flag);
<b>Function descriptions</b>	clear USART interrupt flag in STAT register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=5
<b>Input parameter{in}</b>	
<b>flag</b>	USART interrupt flag, refer to <a href="#">Table 3-980. Enum usart5_interrupt_flag_enum</a>

<i>USART5_INT_FLAG_PERR</i>	parity error interrupt and flag
<i>USART5_INT_FLAG_ERR_FERR</i>	error interrupt and frame error flag
<i>USART5_INT_FLAG_ERR_NERR</i>	error interrupt and noise error flag
<i>USART5_INT_FLAG_RBNE_ORERR</i>	read data buffer not empty interrupt and overrun error flag
<i>USART5_INT_FLAG_ERR_ORERR</i>	error interrupt and overrun error
<i>USART5_INT_FLAG_IDLE</i>	IDLE line detected interrupt and flag
<i>USART5_INT_FLAG_TC</i>	transmission complete interrupt and flag
<i>USART5_INT_FLAG_LBD</i>	LIN break detected interrupt and flag
<i>USART5_INT_FLAG_RT</i>	receiver timeout interrupt and flag
<i>USART5_INT_FLAG_EOB</i>	end of block interrupt and flag
<i>USART5_INT_FLAG_AM</i>	address match interrupt and flag
<i>USART5_INT_FLAG_WU</i>	wakeup from deep-sleep mode interrupt and flag
<i>USART5_INT_FLAG_RFF</i>	receive FIFO full interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the USART5 interrupt flag */
```

```
usart5_interrupt_flag_clear(USART5, USART5_INT_FLAG_TC);
```

## 3.29. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.29.1](#), the WWDGT firmware functions are introduced in chapter [3.29.2](#).

### 3.29.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

**Table 3-1060. WWDGT Registers**

Registers	Descriptions
WWDGT_CTL	Control register
WWDGT_CFG	Configuration register
WWDGT_STAT	Status register

### 3.29.2. Descriptions of Peripheral functions

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

**Table 3-1061. 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 3-1062. 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 3-1063. 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 3-1064. 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 WWDGT value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>counter_value</b>	0x00 - 0x7F
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* update WWDGT counter to 0x7F */
```

```
wwdgt_counter_update(127);
```

## wwdgt\_config

The description of wwdgt\_config is shown as below:

**Table 3-1065. Function wwdgt\_config**

<b>Function name</b>	wwdgt_config
<b>Function prototype</b>	void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler);
<b>Function descriptions</b>	configure counter value, window value, and prescaler divider value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>counter</b>	0x00 - 0x7F
<b>Input parameter{in}</b>	
<b>window</b>	0x00 - 0x7F
<b>Input parameter{in}</b>	
<b>prescaler</b>	wwdgt prescaler value
WWDGT_CFG_PSC_DIV1	the time base of window watchdog counter = (PCLK1/4096)/1
WWDGT_CFG_PSC_DIV2	the time base of window watchdog counter = (PCLK1/4096)/2
WWDGT_CFG_PSC_DIV4	the time base of window watchdog counter = (PCLK1/4096)/4
WWDGT_CFG_PSC_DIV8	the time base of window watchdog counter = (PCLK1/4096)/8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
```

```
wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

## wwdgt\_interrupt\_enable

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

**Table 3-1066. 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>	-

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

Example:

```
/* enable early wakeup interrupt of WWDGT */
```

```
wwdgt_interrupt_enable ( );
```

### wwdgt\_flag\_get

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

**Table 3-1067. Function wwdgt\_flag\_get**

Function name	wwdgt_flag_get
Function prototype	FlagStatus wwdgt_flag_get(void);
Function descriptions	check early wakeup interrupt state of WWDGT
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* test if the counter value update has reached the 0x40 */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get ( );
```

```
if(status == RESET)
```

```
{
```

```
...
```

```
}else
```

```
{
```

```
...
```

```
}
```

## wwdgt\_flag\_clear

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

**Table 3-1068. Function wwdgt\_flag\_clear**

<b>Function name</b>	wwdgt_flag_clear
<b>Function prototype</b>	void wwdgt_flag_clear(void);
<b>Function descriptions</b>	clear early wakeup interrupt state of WWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear early wakeup interrupt state of WWDGT */
wwdgt_flag_clear( );
```



## 4. Revision history

Table 4-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Mar.10, 2020
1.1	Module information modified	Aug.26, 2020
1.2	Module information modified	Mar.31, 2021
1.3	Module information modified	Dec.21, 2021
1.4	<ol style="list-style-type: none"> <li>Add function fwdgt_prescaler_value_config and fwdgt_reload_value_config in <b><u>3.15.2</u></b>.</li> <li>Change function i2c_dma_enable / i2c_smbus_issue_alert / i2c_smbus_arp_enable to i2c_dma_config / i2c_smbus_alert_config / i2c_smbus_arp_config in <b><u>3.18.2</u></b>.</li> <li>Change function pmu_to_standbymode(WFI_CMD) to pmu_to_standbymode() in <b><u>3.20.2</u></b>.</li> <li>Change function qspi_enable / qspi_disable / qspi_write_enable / qspi_read_enable / qspi_io23_output_enable / qspi_io23_output_disable to spi_quad_enable / spi_quad_disable / spi_quad_write_enable / spi_quad_read_enable / spi_quad_output_enable / spi_quad_io23_output_disable in <b><u>3.24.2</u></b>.</li> <li>Delete the description of GD32E50X_XD.</li> </ol>	Aug.08, 2022
1.5	<ol style="list-style-type: none"> <li>Modify the parameters of the function interfaces usart_dma_receive_config and usart_dma_transmit_config in <b><u>3.28.2</u></b>.</li> </ol>	Dec.28, 2022
1.6	<ol style="list-style-type: none"> <li>DAC consistency modification mainly involves modification of bit domain name, register name, macro definition, function, and function comment</li> </ol>	Dec.25, 2023

## Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company under the intellectual property laws and treaties of the People's Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as its suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.