

**GigaDevice Semiconductor Inc.**

**GD32F30x  
ARM® Cortex™-M4 32-bit MCU**

**Firmware Library  
User Guide**

Revison 1.0

(Dec. 2018)

## Table of Contents

<b>Table of Contents .....</b>	<b>2</b>
<b>List of Figures .....</b>	<b>5</b>
<b>List of Tables .....</b>	<b>6</b>
<b>1. Introduction .....</b>	<b>24</b>
<b>1.1. Rules of User Manual and Firmware Library .....</b>	<b>24</b>
1.1.1. Peripherals .....	24
1.1.2. Naming rules .....	25
<b>2. Firmware Library Overview .....</b>	<b>26</b>
<b>2.1. File Structure of Firmware Library .....</b>	<b>26</b>
2.1.1. Examples Folder .....	27
2.1.2. Firmware Folder .....	27
2.1.3. Template Folder .....	28
2.1.4. Utilities Folder .....	30
<b>2.2. File descriptions of Firmware Library .....</b>	<b>31</b>
<b>3. Firmware Library of Standard Peripherals .....</b>	<b>32</b>
<b>3.1. Overview of Firmware Library of Standard Peripherals .....</b>	<b>32</b>
<b>3.2. ADC .....</b>	<b>32</b>
3.2.1. Descriptions of Peripheral registers .....	32
3.2.2. Descriptions of Peripheral functions .....	33
<b>3.3. BKP .....</b>	<b>68</b>
3.3.1. Descriptions of Peripheral registers .....	68
3.3.2. Descriptions of Peripheral functions .....	69
<b>3.4. CAN .....</b>	<b>83</b>
3.4.1. Descriptions of Peripheral registers .....	83
3.4.2. Descriptions of Peripheral functions .....	84
<b>3.5. CRC .....</b>	<b>123</b>
3.5.1. Descriptions of Peripheral registers .....	123
3.5.2. Descriptions of Peripheral functions .....	124
<b>3.6. CTC .....</b>	<b>129</b>
3.6.1. Descriptions of Peripheral registers .....	129
3.6.2. Descriptions of Peripheral functions .....	129
<b>3.7. DAC .....</b>	<b>146</b>
3.7.1. Descriptions of Peripheral registers .....	146
3.7.2. Descriptions of Peripheral functions .....	146

---

<b>3.8. DBG .....</b>	<b>165</b>
3.8.1. Descriptions of Peripheral registers.....	165
3.8.2. Descriptions of Peripheral functions .....	166
<b>3.9. DMA .....</b>	<b>173</b>
3.9.1. Descriptions of Peripheral registers.....	174
3.9.2. Descriptions of Peripheral functions .....	174
<b>3.10. ENET .....</b>	<b>200</b>
3.10.1. Descriptions of Peripheral registers.....	200
3.10.2. Descriptions of Peripheral functions .....	203
<b>3.11. EXMC.....</b>	<b>301</b>
3.11.1. Descriptions of Peripheral registers.....	301
3.11.2. Descriptions of Peripheral functions .....	302
<b>3.12. EXTI.....</b>	<b>325</b>
3.12.1. Descriptions of Peripheral registers.....	326
3.12.2. Descriptions of Peripheral functions .....	326
<b>3.13. FMC .....</b>	<b>335</b>
3.13.1. Descriptions of Peripheral registers.....	335
3.13.2. Descriptions of Peripheral functions .....	335
<b>3.14. FWDGT .....</b>	<b>376</b>
3.14.1. Descriptions of Peripheral registers.....	376
3.14.2. Descriptions of Peripheral functions .....	376
<b>3.15. GPIO .....</b>	<b>381</b>
3.15.1. Descriptions of Peripheral registers.....	381
3.15.2. Descriptions of Peripheral functions .....	382
<b>3.16. I2C .....</b>	<b>400</b>
3.16.1. Descriptions of Peripheral registers.....	400
3.16.2. Descriptions of Peripheral functions .....	401
<b>3.17. MISC .....</b>	<b>429</b>
3.17.1. Descriptions of Peripheral registers.....	429
3.17.2. Descriptions of Peripheral functions .....	430
<b>3.18. PMU .....</b>	<b>438</b>
3.18.1. Descriptions of Peripheral registers.....	438
3.18.2. Descriptions of Peripheral functions .....	438
<b>3.19. RCU .....</b>	<b>453</b>
3.19.1. Descriptions of Peripheral registers.....	453
3.19.2. Descriptions of Peripheral functions .....	454
<b>3.20. RTC.....</b>	<b>494</b>
3.20.1. Descriptions of Peripheral registers.....	494
3.20.2. Descriptions of Peripheral functions .....	494



## GD32F30x Firmware Library User Guide

---

<b>3.21. SDIO .....</b>	<b>504</b>
3.21.1. Descriptions of Peripheral registers.....	504
3.21.2. Descriptions of Peripheral functions .....	505
<b>3.22. SPI .....</b>	<b>544</b>
3.22.1. Descriptions of Peripheral registers.....	544
3.22.2. Descriptions of Peripheral functions .....	544
<b>3.23. TIMER .....</b>	<b>577</b>
3.23.1. Descriptions of Peripheral registers.....	578
3.23.2. Descriptions of Peripheral functions .....	578
<b>3.24. USART.....</b>	<b>650</b>
3.24.1. Descriptions of Peripheral registers.....	650
3.24.2. Descriptions of Peripheral functions .....	650
<b>3.25. WWDGT.....</b>	<b>693</b>
3.25.1. Descriptions of Peripheral registers.....	693
3.25.2. Descriptions of Peripheral functions .....	693
<b>3.26. USBFS.....</b>	<b>699</b>
<b>4. Revision history .....</b>	<b>700</b>

## List of Figures

Figure 2-1. File structure of firmware library of GD32F30x .....	26
Figure 2-2. Select peripheral example files.....	28
Figure 2-3. Copy the peripheral example files.....	29
Figure 2-4. Open the project file.....	29
Figure 2-5. Configure project files .....	30
Figure 2-6. Compile-debug-download .....	30

## List of Tables

<b>Table 1-1. Peripherals .....</b>	<b>24</b>
<b>Table 2-1. Function descriptions of Firmware Library .....</b>	<b>31</b>
<b>Table 3-1. Peripheral function format of Firmware Library.....</b>	<b>32</b>
<b>Table 3-2. ADC Registers.....</b>	<b>33</b>
<b>Table 3-3. ADC firmware function.....</b>	<b>33</b>
<b>Table 3-4. Function adc_deinit.....</b>	<b>35</b>
<b>Table 3-5. Function adc_enable.....</b>	<b>35</b>
<b>Table 3-6. Function adc_disable.....</b>	<b>36</b>
<b>Table 3-7. Function adc_calibration_enable .....</b>	<b>37</b>
<b>Table 3-8. Function adc_dma_mode_enable .....</b>	<b>37</b>
<b>Table 3-9. Function adc_dma_mode_disable .....</b>	<b>38</b>
<b>Table 3-10. Function adc_tempsensor_vrefint_enable .....</b>	<b>39</b>
<b>Table 3-11. Function adc_tempsensor_vrefint_disable .....</b>	<b>39</b>
<b>Table 3-12. Function adc_resolution_config .....</b>	<b>40</b>
<b>Table 3-13. Function adc_discontinuous_mode_config .....</b>	<b>41</b>
<b>Table 3-14. Function adc_mode_config .....</b>	<b>42</b>
<b>Table 3-15. Function adc_special_function_config .....</b>	<b>43</b>
<b>Table 3-16. Function adc_data_alignment_config .....</b>	<b>44</b>
<b>Table 3-17. Function adc_channel_length_config .....</b>	<b>45</b>
<b>Table 3-18. Function adc_regular_channel_config.....</b>	<b>46</b>
<b>Table 3-19. Function adc_inserted_channel_config.....</b>	<b>47</b>
<b>Table 3-20. Function adc_inserted_channel_offset_config.....</b>	<b>49</b>
<b>Table 3-21. Function adc_external_trigger_config .....</b>	<b>50</b>
<b>Table 3-22. Function adc_external_trigger_source_config .....</b>	<b>51</b>
<b>Table 3-23. Function adc_software_trigger_enable.....</b>	<b>53</b>
<b>Table 3-24. Function adc_regular_data_read .....</b>	<b>54</b>
<b>Table 3-25. Function adc_inserted_data_read .....</b>	<b>55</b>
<b>Table 3-26. Function adc_sync_mode_convert_value_read .....</b>	<b>56</b>
<b>Table 3-27. Function adc_flag_get .....</b>	<b>56</b>
<b>Table 3-28. Function adc_flag_clear .....</b>	<b>57</b>
<b>Table 3-29. Function adc_interrupt_flag_get.....</b>	<b>58</b>
<b>Table 3-30. Function adc_interrupt_flag_clear.....</b>	<b>59</b>
<b>Table 3-31. Function adc_interrupt_enable .....</b>	<b>60</b>
<b>Table 3-32. Function adc_interrupt_disable .....</b>	<b>61</b>
<b>Table 3-33. Function adc_watchdog_single_channel_enable.....</b>	<b>62</b>
<b>Table 3-34. Function adc_watchdog_group_channel_enable.....</b>	<b>62</b>
<b>Table 3-35. Function adc_watchdog_disable .....</b>	<b>63</b>
<b>Table 3-36. Function adc_watchdog_threshold_config .....</b>	<b>64</b>
<b>Table 3-37. Function adc_oversample_mode_config.....</b>	<b>65</b>
<b>Table 3-38. Function adc_oversample_mode_enable .....</b>	<b>67</b>

Table 3-39. Function adc_oversample_mode_disable .....	67
Table 3-40. BKP Registers .....	68
Table 3-41. BKP firmware function.....	69
Table 3-42. Function bkp_deinit .....	70
Table 3-43. Function bkp_write_data.....	70
Table 3-44. Function bkp_data_read.....	71
Table 3-45. Function bkp_rtc_calibration_output_enable .....	72
Table 3-46. Function bkp_rtc_calibration_output_disable .....	72
Table 3-47. Function bkp_rtc_signal_output_enable .....	73
Table 3-48. Function bkp_rtc_signal_output_disable .....	74
Table 3-49. Function bkp_rtc_output_select .....	74
Table 3-50. Function bkp_rtc_clock_output_select.....	75
Table 3-51. Function bkp_rtc_clock_calibration_direction .....	76
Table 3-52. Function bkp_rtc_calibration_value_set.....	76
Table 3-53. Function bkp_tamper_detection_enable.....	77
Table 3-54. Function bkp_tamper_detection_disable.....	78
Table 3-55. Function bkp_tamper_active_level_set.....	78
Table 3-56. Function bkp_tamper_interrupt_enable.....	79
Table 3-57. Function bkp_tamper_interrupt_disable.....	80
Table 3-58. Function bkp_flag_get .....	80
Table 3-59. Function bkp_flag_clear .....	81
Table 3-60. Function bkp_interrupt_flag_get.....	82
Table 3-61. Function bkp_interrupt_flag_clear.....	82
Table 3-62. CAN Registers.....	83
Table 3-63. CAN firmware function .....	84
Table 3-64. can_parameter_struct.....	85
Table 3-65. can_trasnmit_message_struct .....	86
Table 3-66. can_receive_message_struct .....	86
Table 3-67. can_filter_parameter_struct.....	86
Table 3-68. Function can_deinit.....	87
Table 3-69. Function can_struct_para_init.....	88
Table 3-70. Function can_init.....	89
Table 3-71. Function can_filter_init.....	92
Table 3-72. Function can1_filter_start_bank.....	95
Table 3-73. Function can_debug_freeze_enable .....	95
Table 3-74. Function can_debug_freeze_disable.....	96
Table 3-75. Function can_time_trigger_mode_enable .....	97
Table 3-76. Function can_time_trigger_mode_disable .....	98
Table 3-77. Function can_message_transmit.....	100
Table 3-78. Function can_transmit_states .....	102
Table 3-79. Function can_transmission_stop.....	103
Table 3-80. Function can_message_receive.....	105
Table 3-81. Function can_fifo_release.....	107
Table 3-82. Function can_receive_message_length_get .....	109

Table 3-83. Function can_working_mode_set.....	110
Table 3-84. Function can_wakeup.....	111
Table 3-85. Function can_error_get .....	112
Table 3-86. Function can_receive_error_number_get.....	113
Table 3-87. Function can_transmit_error_number_get.....	114
Table 3-88. Function can_interrupt_enable .....	115
Table 3-89. Function can_interrupt_disable .....	118
Table 3-90. Function can_flag_get .....	119
Table 3-91. Function can_flag_clear .....	120
Table 3-92. Function can_interrupt_flag_get.....	121
Table 3-93. Function can_interrupt_flag_clear.....	122
Table 3-94. CRC Registers.....	123
Table 3-95. CRC firmware function .....	124
Table 3-96. Function crc_deinit .....	124
Table 3-97. Function crc_data_register_reset .....	125
Table 3-98. Function crc_data_register_read .....	125
Table 3-99. Function crc_free_data_register_read .....	126
Table 3-100. Function crc_free_data_register_write .....	127
Table 3-101. Function crc_single_data_calculate.....	127
Table 3-102. Function crc_block_data_calculate.....	128
Table 3-103. CTC Registers .....	129
Table 3-104. CTC firmware function .....	129
Table 3-105. Function ctc_deinit.....	130
Table 3-106. Function ctc_counter_enable .....	131
Table 3-107. Function ctc_counter_disable .....	132
Table 3-108. Function ctc_irc48m_trim_value_config.....	132
Table 3-109. Function ctc_software_refresource_pulse_generate .....	133
Table 3-110. Function ctc_hardware_trim_mode_config .....	133
Table 3-111. Function ctc_refresource_polarity_config .....	134
Table 3-112. Function ctc_refresource_signal_select .....	135
Table 3-113. Function ctc_refresource_prescaler_config .....	136
Table 3-114. Function ctc_clock_limit_value_config.....	137
Table 3-115. Function ctc_counter_reload_value_config .....	137
Table 3-116. Function ctc_counter_capture_value_read .....	138
Table 3-117. Function ctc_counter_direction_read .....	139
Table 3-118. Function ctc_counter_reload_value_read .....	139
Table 3-119. Function ctc_irc48m_trim_value_read .....	140
Table 3-120. Function ctc_interrupt_enable .....	141
Table 3-121. Function ctc_interrupt_disable .....	141
Table 3-122. Function ctc_interrupt_flag_get .....	142
Table 3-123. Function ctc_interrupt_flag_clear .....	143
Table 3-124. Function ctc_flag_get .....	144
Table 3-125. Function ctc_flag_clear .....	145
Table 3-126. DAC Registers.....	146

Table 3-127. DAC firmware function .....	147
Table 3-128. Function <code>dac_deinit</code> .....	148
Table 3-129. Function <code>dac_enable</code> .....	148
Table 3-130. Function <code>dac_disable</code> .....	149
Table 3-131. Function <code>dac_dma_enable</code> .....	150
Table 3-132. Function <code>dac_dma_disable</code> .....	150
Table 3-133. Function <code>dac_output_buffer_enable</code> .....	151
Table 3-134. Function <code>dac_output_buffer_disable</code> .....	151
Table 3-135. Function <code>dac_output_value_get</code> .....	152
Table 3-136. Function <code>dac_data_set</code> .....	153
Table 3-137. Function <code>dac_trigger_enable</code> .....	154
Table 3-138. Function <code>dac_trigger_disable</code> .....	154
Table 3-139. Function <code>dac_trigger_source_config</code> .....	155
Table 3-140. Function <code>dac_software_trigger_enable</code> .....	156
Table 3-141. Function <code>dac_software_trigger_disable</code> .....	157
Table 3-142. Function <code>dac_wave_mode_config</code> .....	157
Table 3-143. Function <code>dac_wave_bit_width_config</code> .....	158
Table 3-144. Function <code>dac_lfsr_noise_config</code> .....	159
Table 3-145. Function <code>dac_triangle_noise_config</code> .....	160
Table 3-146. Function <code>dac_concurrent_enable</code> .....	161
Table 3-147. Function <code>dac_concurrent_disable</code> .....	161
Table 3-148. Function <code>dac_concurrent_software_trigger_enable</code> .....	162
Table 3-149. Function <code>dac_concurrent_software_trigger_disable</code> .....	163
Table 3-150. Function <code>dac_concurrent_output_buffer_enable</code> .....	163
Table 3-151. Function <code>dac_concurrent_output_buffer_disable</code> .....	164
Table 3-152. Function <code>dac_concurrent_data_set</code> .....	164
Table 3-153. DBG Registers .....	165
Table 3-154. DBG firmware function .....	166
Table 3-155. Enum <code>dbg_periph_enum</code> .....	166
Table 3-156. Function <code>dbg_deinit</code> .....	167
Table 3-157. Function <code>dbg_id_get</code> .....	168
Table 3-158. Function <code>dbg_low_power_enable</code> .....	168
Table 3-159. Function <code>dbg_low_power_disable</code> .....	169
Table 3-160. Function <code>dbg_periph_enable</code> .....	170
Table 3-161. Function <code>dbg_periph_disable</code> .....	171
Table 3-162. Function <code>dbg_trace_pin_enable</code> .....	171
Table 3-163. Function <code>dbg_trace_pin_disable</code> .....	172
Table 3-164. Function <code>dbg_trace_pin_mode_set</code> .....	173
Table 3-165. DMA Registers .....	174
Table 3-166. DMA firmware function .....	174
Table 3-167. Structure <code>dma_parameter_struct</code> .....	175
Table 3-168. Function <code>dma_deinit</code> .....	176
Table 3-169. Function <code>dma_struct_para_init</code> .....	176
Table 3-170. Function <code>dma_init</code> .....	177

Table 3-171. Function <code>dma_circulation_enable</code> .....	178
Table 3-172. Function <code>dma_circulation_disable</code> .....	179
Table 3-173. Function <code>dma_memory_to_memory_enable</code> .....	180
Table 3-174. Function <code>dma_memory_to_memory_disable</code> .....	181
Table 3-175. Function <code>dma_channel_enable</code> .....	181
Table 3-176. Function <code>dma_channel_disable</code> .....	182
Table 3-177. Function <code>dma_periph_address_config</code> .....	183
Table 3-178. Function <code>dma_memory_address_config</code> .....	184
Table 3-179. Function <code>dma_transfer_number_config</code> .....	185
Table 3-180. Function <code>dma_transfer_number_get</code> .....	186
Table 3-181. Function <code>dma_priority_config</code> .....	186
Table 3-182. Function <code>dma_memory_width_config</code> .....	187
Table 3-183. Function <code>dma_periph_width_config</code> .....	188
Table 3-184. Function <code>dma_memory_increase_enable</code> .....	189
Table 3-185. Function <code>dma_memory_increase_disable</code> .....	190
Table 3-186. Function <code>dma_periph_increase_enable</code> .....	191
Table 3-187. Function <code>dma_periph_increase_disable</code> .....	192
Table 3-188. Function <code>dma_transfer_direction_config</code> .....	193
Table 3-189. Function <code>dma_flag_get</code> .....	193
Table 3-190. Function <code>dma_flag_clear</code> .....	194
Table 3-191. Function <code>dma_interrupt_flag_get</code> .....	195
Table 3-192. Function <code>dma_interrupt_flag_clear</code> .....	197
Table 3-193. Function <code>dma_interrupt_enable</code> .....	198
Table 3-194. Function <code>dma_interrupt_disable</code> .....	199
Table 3-195. ENET Registers.....	200
Table 3-196. ENET firmware function .....	203
Table 3-197. Structure <code>enet_initpara_struct</code> .....	207
Table 3-198. Structure <code>enet_descriptors_struct</code> .....	208
Table 3-199. Structure <code>enet_ptp_systime_struct</code> .....	209
Table 3-200. Function <code>enet_deinit</code> .....	209
Table 3-201. Function <code>enet_initpara_config</code> .....	209
Table 3-202. Function <code>enet_init</code> .....	214
Table 3-203. Function <code>enet_software_reset</code> .....	216
Table 3-204. Function <code>enet_rxframe_size_get</code> .....	216
Table 3-205. Function <code>enet_descriptors_chain_init</code> .....	217
Table 3-206. Function <code>enet_descriptors_ring_init</code> .....	218
Table 3-207. Function <code>enet_frame_receive</code> .....	218
Table 3-208. Function <code>enet_frame_transmit</code> .....	219
Table 3-209. Function <code>enet_transmit_checksum_config</code> .....	220
Table 3-210. Function <code>enet_enable</code> .....	221
Table 3-211. Function <code>enet_disable</code> .....	221
Table 3-212. Function <code>enet_mac_address_set</code> .....	222
Table 3-213. Function <code>enet_mac_address_get</code> .....	223
Table 3-214. Function <code>enet_flag_get</code> .....	224

Table 3-215. Function enet_flag_clear.....	227
Table 3-216. Function enet_interrupt_enable .....	228
Table 3-217. Function enet_interrupt_disable .....	230
Table 3-218. Function enet_interrupt_flag_get.....	232
Table 3-219. Function enet_interrupt_flag_clear.....	234
Table 3-220. Function enet_tx_enable .....	236
Table 3-221. Function enet_tx_disable .....	236
Table 3-222. Function enet_rx_enable.....	237
Table 3-223. Function enet_rx_disable.....	238
Table 3-224. Function enet_registers_get.....	238
Table 3-225. Function enet_debug_status_get .....	239
Table 3-226. Function enet_address_filter_enable .....	240
Table 3-227. Function enet_address_filter_disable .....	241
Table 3-228. Function enet_address_filter_config.....	242
Table 3-229. Function enet_phy_config .....	244
Table 3-230. Function enet_phy_write_read .....	244
Table 3-231. Function enet_phyloopback_enable .....	245
Table 3-232. Function enet_phyloopback_disable .....	246
Table 3-233. Function enet_forward_feature_enable .....	246
Table 3-234. Function enet_forward_feature_disable .....	247
Table 3-235. Function enet_fliter_feature_enable.....	248
Table 3-236. Function enet_fliter_feature_disable .....	249
Table 3-237. Function enet_pauseframe_generate .....	250
Table 3-238. Function enet_pauseframe_detect_config .....	251
Table 3-239. Function enet_pauseframe_config .....	252
Table 3-240. Function enet_flowcontrol_threshold_config .....	253
Table 3-241. Function enet_flowcontrol_feature_enable .....	254
Table 3-242. Function enet_flowcontrol_feature_disable .....	255
Table 3-243. Function enet_dmaprocess_state_get .....	256
Table 3-244. Function enet_dmaprocess_resume .....	257
Table 3-245. Function enet_rxprocess_check_recovery .....	258
Table 3-246. Function enet_txfifo_flush .....	259
Table 3-247. Function enet_current_desc_address_get.....	259
Table 3-248. Function enet_desc_information_get.....	260
Table 3-249. Function enet_missed_frame_counter_get.....	261
Table 3-250. Function enet_desc_flag_get .....	262
Table 3-251. Function enet_desc_flag_set.....	265
Table 3-252. Function enet_desc_flag_clear .....	266
Table 3-253. Function enet_rx_desc_immediate_receive_complete_interrupt .....	267
Table 3-254. Function enet_rx_desc_delay_receive_complete_interrupt .....	268
Table 3-255. Function enet_rxframe_drop .....	269
Table 3-256. Function enet_dma_feature_enable .....	269
Table 3-257. Function enet_dma_feature_disable .....	270
Table 3-258. Function enet_rx_desc_enhanced_status_get .....	271

---

Table 3-259. Function enet_desc_select_enhanced_mode .....	272
Table 3-260. Function enet_ptp_enhanced_descriptors_chain_init .....	272
Table 3-261. Function enet_ptp_enhanced_descriptors_ring_init .....	273
Table 3-262. Function enet_ptpframe_receive_enhanced_mode .....	274
Table 3-263. Function enet_ptpframe_transmit_enhanced_mode .....	275
Table 3-264. Function enet_desc_select_normal_mode .....	276
Table 3-265. Function enet_ptp_normal_descriptors_chain_init .....	276
Table 3-266. Function enet_ptp_normal_descriptors_ring_init .....	277
Table 3-267. Function enet_ptpframe_receive_normal_mode .....	278
Table 3-268. Function enet_ptpframe_transmit_normal_mode .....	279
Table 3-269. Function enet_wum_filter_register_pointer_reset .....	280
Table 3-270. Function enet_wum_filter_config .....	280
Table 3-271. Function enet_wum_feature_enable .....	281
Table 3-272. Function enet_wum_feature_disable .....	282
Table 3-273. Function enet_msc_counters_reset .....	283
Table 3-274. Function enet_msc_feature_enable .....	283
Table 3-275. Function enet_msc_feature_disable .....	284
Table 3-276. Function enet_msc_counters_preset_config .....	285
Table 3-277. Function enet_msc_counters_get .....	286
Table 3-278. Function enet_ptp_subsecond_2_nanosecond .....	287
Table 3-279. Function enet_ptp_nanosecond_2_subsecond .....	287
Table 3-280. Function enet_ptp_feature_enable .....	288
Table 3-281. Function enet_ptp_feature_disable .....	289
Table 3-282. Function enet_ptp_timestamp_function_config .....	290
Table 3-283. Function enet_ptp_subsecond_increment_config .....	292
Table 3-284. Function enet_ptp_timestamp_addend_config .....	293
Table 3-285. Function enet_ptp_timestamp_update_config .....	293
Table 3-286. Function enet_ptp_expected_time_config .....	294
Table 3-287. Function enet_ptp_system_time_get .....	295
Table 3-288. enet_ptp_pps_output_frequency_config .....	296
Table 3-289. enet_ptp_start .....	297
Table 3-290. enet_ptp_finecorrection_adjfreq .....	298
Table 3-291. enet_ptp_coarsecorrection_systime_update .....	299
Table 3-292. enet_ptp_finecorrection_settime .....	299
Table 3-293. enet_ptp_flag_get .....	300
Table 3-294. Function enet_initpara_reset .....	301
Table 3-295. EXMC Registers .....	301
Table 3-296. EXMC firmware function .....	302
Table 3-297. Structure exmc_norsram_timing_parameter_struct .....	303
Table 3-298. Structure exmc_norsram_parameter_struct .....	303
Table 3-299. Structure exmc_nand_pccard_timing_parameter_struct .....	304
Table 3-300. Structure exmc_nand_parameter_struct .....	304
Table 3-301. Structure exmc_pccard_parameter_struct .....	305
Table 3-302. Function exmc_norsram_deinit .....	305

---

Table 3-303. Function exmc_norsram_struct_para_init.....	306
Table 3-304. Function exmc_norsram_init.....	307
Table 3-305. Function exmc_norsram_enable.....	308
Table 3-306. Function exmc_norsram_disable.....	309
Table 3-307. Function exmc_nand_deinit .....	310
Table 3-308. Function exmc_nand_struct_para_init.....	310
Table 3-309. Function exmc_nand_init.....	311
Table 3-310. Function exmc_nand_enable.....	312
Table 3-311. Function exmc_nand_disable .....	313
Table 3-312. Function exmc_pccard_deinit .....	313
Table 3-313. Function exmc_pccard_struct_para_init .....	314
Table 3-314. Function exmc_pccard_init.....	315
Table 3-315. Function exmc_pccard_enable .....	316
Table 3-316. Function exmc_pccard_disable .....	316
Table 3-317. Function exmc_norsram_page_size_config.....	317
Table 3-318. Function exmc_nand_ecc_config .....	318
Table 3-319. Function exmc_ecc_get .....	319
Table 3-320. Function exmc_interrupt_enable .....	319
Table 3-321. Function exmc_interrupt_disable .....	320
Table 3-322. Function exmc_flag_get .....	321
Table 3-323. Function exmc_flag_clear .....	322
Table 3-324. Function exmc_interrupt_flag_get.....	323
Table 3-325. Function exmc_interrupt_flag_clear.....	324
Table 3-326. EXTI Registers .....	326
Table 3-327. EXTI firmware function .....	326
Table 3-328. Function exti_deinit .....	327
Table 3-329. Function exti_init.....	327
Table 3-330. Function exti_interrupt_enable .....	328
Table 3-331. Function exti_interrupt_disable .....	329
Table 3-332. Function exti_event_enable.....	329
Table 3-333. Function exti_event_disable.....	330
Table 3-334. Function exti_software_interrupt_enable .....	331
Table 3-335. Function exti_software_interrupt_disable .....	331
Table 3-336. Function exti_flag_get .....	332
Table 3-337. Function exti_flag_clear .....	333
Table 3-338. Function exti_interrupt_flag_get.....	333
Table 3-339. Function exti_interrupt_flag_clear.....	334
Table 3-340. FMC Registers.....	335
Table 3-341. FMC firmware function .....	335
Table 3-342. fmc_state_enum .....	337
Table 3-343. fmc_int_enum .....	337
Table 3-344. fmc_flag_enum .....	338
Table 3-345. fmc_interrupt_flag_enum.....	338
Table 3-346. Function fmc_wsct_set .....	339

---

Table 3-347. Function fmc_prefetch_disable .....	340
Table 3-348. Function fmc_ibus_enable .....	341
Table 3-349. Function fmc_ibus_disable .....	342
Table 3-350. Function fmc_dbus_enable .....	342
Table 3-351. Function fmc_dbus_disable .....	343
Table 3-352. Function fmc_ibus_reset .....	344
Table 3-353. Function fmc_dbus_reset .....	344
Table 3-354. Function fmc_program_width_set .....	345
Table 3-355. Function fmc_unlock .....	345
Table 3-356. Function fmc_bank0_unlock .....	346
Table 3-357. Function fmc_bank1_unlock .....	347
Table 3-358. Function fmc_lock .....	347
Table 3-359. Function fmc_bank0_lock .....	348
Table 3-360. Function fmc_bank1_lock .....	348
Table 3-361. Function fmc_page_erase .....	349
Table 3-362. Function fmc_mass_erase .....	350
Table 3-363. Function fmc_bank0_erase .....	351
Table 3-364. Function fmc_bank1_erase .....	351
Table 3-365. Function fmc_word_program .....	352
Table 3-366. Function fmc_halfword_program .....	353
Table 3-367. Function fmc_word_reprogram .....	354
Table 3-368. Function ob_unlock .....	355
Table 3-369. Function ob_lock .....	355
Table 3-370. Function ob_erase .....	356
Table 3-371. Function ob_write_protection_enable .....	357
Table 3-372. Function ob_security_protection_config .....	358
Table 3-373. Function ob_user_write .....	359
Table 3-374. Function ob_data_program .....	361
Table 3-375. Function ob_user_get .....	362
Table 3-376. Function ob_data_program .....	363
Table 3-377. Function ob_write_protection_get .....	363
Table 3-378. Function ob_spc_get .....	364
Table 3-379. Function fmc_interrupt_enable .....	364
Table 3-380. Function fmc_interrupt_disable .....	366
Table 3-381. Function fmc_flag_get .....	367
Table 3-382. Function fmc_flag_clear .....	369
Table 3-383. Function fmc_interrupt_flag_get .....	370
Table 3-384. Function fmc_interrupt_flag_clear .....	371
Table 3-385. Function fmc_bank0_state_get .....	372
Table 3-386. Function fmc_bank1_state_get .....	373
Table 3-387. Function fmc_bank0_ready_wait .....	374
Table 3-388. Function fmc_bank1_ready_wait .....	375
Table 3-389. FWDGT Registers .....	376
Table 3-390. FWDGT firmware function .....	376

Table 3-391. Function fwdgt_write_ensable .....	376
Table 3-392. Function fwdgt_write_disable .....	377
Table 3-393. Function fwdgt_enable .....	378
Table 3-394. Function fwdgt_counter_reload .....	378
Table 3-395. Function fwdgt_config .....	379
Table 3-396. Function fwdgt_flag_get fwdgt_write_disable .....	380
Table 3-397. GPIO Registers .....	381
Table 3-398. GPIO firmware function .....	382
Table 3-399. Function gpio_deinit .....	383
Table 3-400. Function gpio_afio_deinit .....	383
Table 3-401. Function gpio_init .....	384
Table 3-402. Function gpio_bit_set .....	385
Table 3-403. Function gpio_bit_reset .....	386
Table 3-404. Function gpio_bit_write .....	387
Table 3-405. Function gpio_port_write .....	388
Table 3-406. Function gpio_input_bit_get .....	389
Table 3-407. Function gpio_input_port_get .....	389
Table 3-408. Function gpio_output_bit_get .....	390
Table 3-409. Function gpio_output_port_get .....	391
Table 3-410. Function gpio_pin_remap_config .....	392
Table 3-411. Function gpio_ethernet_phy_select .....	395
Table 3-412. Function gpio_exti_source_select .....	395
Table 3-413. Function gpio_event_output_config .....	396
Table 3-414. Function gpio_event_output_enable .....	397
Table 3-415. Function gpio_event_output_disable .....	397
Table 3-416. Function gpio_pin_lock .....	398
Table 3-417. Function gpio_compensation_config .....	399
Table 3-418. Function gpio_compensation_flag_get .....	400
Table 3-419. I2C Registers .....	400
Table 3-420. I2C firmware function .....	401
Table 3-421. Function i2c_deinit .....	402
Table 3-422. Function i2c_clock_config .....	403
Table 3-423. Function i2c_mode_addr_config .....	404
Table 3-424. Function i2c_smbus_type_config .....	405
Table 3-425. Function i2c_ack_config .....	406
Table 3-426. Function i2c_ackpos_config .....	406
Table 3-427. Function i2c_master_addressing .....	407
Table 3-428. Function i2c_dualaddr_enable .....	408
Table 3-429. Function i2c_enable .....	409
Table 3-430. Function i2c_disable .....	410
Table 3-431. Function i2c_start_on_bus .....	410
Table 3-432. Function i2c_stop_on_bus .....	411
Table 3-433. Function i2c_data_transmit .....	412
Table 3-434. Function i2c_data_receive .....	412

Table 3-435. Function i2c_dma_enable .....	413
Table 3-436. Function i2c_dma_last_transfer_enable .....	414
Table 3-437. Function i2c_stretch_scl_low_config .....	415
Table 3-438. Function i2c_slave_response_to_gcall_config .....	415
Table 3-439. Function i2c_software_reset_config .....	416
Table 3-440. Function i2c_pec_enable .....	417
Table 3-441. Function i2c_pec_transfer_enable .....	418
Table 3-442. Function i2c_pec_value_get .....	419
Table 3-443. Function i2c_smbus_issue_alert .....	419
Table 3-444. Function i2c_smbus_arp_enable .....	420
Table 3-445. Function i2c_flag_get .....	421
Table 3-446. Function i2c_flag_clear .....	423
Table 3-447. Function i2c_interrupt_enable .....	424
Table 3-448. Function i2c_interrupt_disable .....	425
Table 3-449. Function i2c_interrupt_flag_get .....	426
Table 3-450. Function i2c_interrupt_flag_clear .....	427
Table 3-451. NVIC Registers .....	429
Table 3-452. SysTick Registers .....	430
Table 3-453. IRQn_Type .....	430
Table 3-454. MISC firmware function .....	432
Table 3-455. Function nvic_priority_group_set .....	432
Table 3-456. Function nvic_irq_enable .....	433
Table 3-457. Function nvic_irq_disable .....	434
Table 3-458. Function nvic_vector_table_set .....	435
Table 3-459. Function system_lowpower_set .....	435
Table 3-460. Function system_lowpower_reset .....	436
Table 3-461. Function systick_clksource_set .....	437
Table 3-462. PMU Registers .....	438
Table 3-463. PMU firmware function .....	438
Table 3-464. Function pmu_deinit .....	439
Table 3-465. Function pmu_lvd_select .....	440
Table 3-466. Function pmu_ldo_output_select .....	441
Table 3-467. Function pmu_lvd_disable .....	441
Table 3-468. Function pmu_highdriver_switch_select .....	442
Table 3-469. Function pmu_highdriver_mode_enable .....	443
Table 3-470. Function pmu_highdriver_mode_disable .....	443
Table 3-471. Function pmu_lowdriver_mode_enable .....	444
Table 3-472. Function pmu_lowdriver_mode_disable .....	444
Table 3-473. Function pmu_lowpower_driver_config .....	445
Table 3-474. Function pmu_normalpower_driver_config .....	446
Table 3-475. Function pmu_to_sleepmode .....	447
Table 3-476. Function pmu_to_deepsleepmode .....	447
Table 3-477. Function pmu_to_standbymode .....	448
Table 3-478. Function pmu_backup_write_enable .....	449

---

Table 3-479. Function pmu_backup_write_disable .....	449
Table 3-480. Function pmu_wakeup_pin_enable .....	450
Table 3-481. Function pmu_wakeup_pin_disable .....	451
Table 3-482. Function pmu_flag_get.....	451
Table 3-483. Function pmu_flag_clear.....	452
Table 3-484. RCU Registers.....	453
Table 3-485. RCU firmware function .....	454
Table 3-486. Function rcu_deinit .....	456
Table 3-487. Function rcu_periph_clock_enable .....	456
Table 3-488. Function rcu_periph_clock_disable .....	458
Table 3-489. Function rcu_periph_clock_sleep_enable.....	459
Table 3-490. Function rcu_periph_clock_sleep_disable.....	460
Table 3-491. Function rcu_periph_reset_enable .....	460
Table 3-492. Function rcu_periph_reset_disable .....	462
Table 3-493. Function rcu_bkp_reset_enable.....	463
Table 3-494. Function rcu_bkp_reset_disable.....	463
Table 3-495. Function rcu_system_clock_source_config .....	464
Table 3-496. Function rcu_system_clock_source_get .....	465
Table 3-497. Function rcu_ahb_clock_config.....	465
Table 3-498. Function rcu_apb1_clock_config.....	466
Table 3-499. Function rcu_apb2_clock_config.....	467
Table 3-500. Function rcu_ckout0_config .....	468
Table 3-501. Function rcu_pll_config .....	469
Table 3-502. Function rcu_pllpresel_config.....	470
Table 3-503. Function rcu_pdrv0_config.....	471
Table 3-504. Function rcu_pdrv0_config.....	471
Table 3-505. Function rcu_pdrv1_config.....	472
Table 3-506. Function rcu_pll1_config .....	473
Table 3-507. Function rcu_pll2_config .....	473
Table 3-508. Function rcu_adc_clock_config .....	474
Table 3-509. Function rcu_usb_clock_config.....	475
Table 3-510. Function rcu_rtc_clock_config.....	476
Table 3-511. Function rcu_i2s1_clock_config .....	477
Table 3-512. Function rcu_i2s2_clock_config .....	478
Table 3-513. Function rcu_ck48m_clock_config .....	479
Table 3-514. Function rcu_flag_get.....	479
Table 3-515. Function rcu_all_reset_flag_clear.....	481
Table 3-516. Function rcu_interrupt_flag_get.....	481
Table 3-517. Function rcu_interrupt_flag_clear .....	482
Table 3-518. Function rcu_interrupt_enable .....	483
Table 3-519. Function rcu_interrupt_disable .....	484
Table 3-520. Function rcu_lxtal_drive_capability_config .....	485
Table 3-521. Function rcu_osc_stab_wait.....	486
Table 3-522. Function rcu_osc_on .....	487

Table 3-523. Function rcu_osc1_off .....	488
Table 3-524. Function rcu_osc1_bypass_mode_enable .....	489
Table 3-525. Function rcu_osc1_bypass_mode_disable .....	489
Table 3-526. Function rcu_hxtal_clock_monitor_enable .....	490
Table 3-527. Function rcu_hxtal_clock_monitor_disable .....	491
Table 3-528. Function rcu_irc8m_adjust_value_set .....	491
Table 3-529. Function rcu_deepsleep_voltage_set .....	492
Table 3-530. Function rcu_clock_freq_get .....	493
Table 3-531. RTC Registers .....	494
Table 3-532. RTC firmware function .....	494
Table 3-533. Function rtc_interrupt_enable .....	495
Table 3-534. Function rtc_interrupt_disable .....	496
Table 3-535. Function rtc_configuration_mode_enter .....	497
Table 3-536. Function rtc_configuration_mode_exit .....	497
Table 3-537. Function rtc_lwoff_wait .....	498
Table 3-538. Function rtc_register_sync_wait .....	498
Table 3-539. Function rtc_counter_get .....	499
Table 3-540. Function rtc_counter_set .....	500
Table 3-541. Function rtc_prescaler_set .....	500
Table 3-542. Function rtc_alarm_config .....	501
Table 3-543. Function rtc_divider_get .....	502
Table 3-544. Function rtc_flag_get .....	502
Table 3-545. Function rtc_flag_clear .....	503
Table 3-546. SDIO Registers .....	504
Table 3-547. SDIO firmware function .....	505
Table 3-548. Function sdio_deinit .....	507
Table 3-549. Function sdio_clock_config .....	508
Table 3-550. Function sdio_hardware_clock_enable .....	509
Table 3-551. Function sdio_hardware_clock_disable .....	509
Table 3-552. Function sdio_bus_mode_set .....	510
Table 3-553. Function sdio_power_state_set .....	511
Table 3-554. Function sdio_power_state_get .....	512
Table 3-555. Function sdio_clock_enable .....	512
Table 3-556. Function sdio_clock_disable .....	513
Table 3-557. Function sdio_command_response_config .....	513
Table 3-558. Function sdio_wait_type_set .....	514
Table 3-559. Function sdio_csm_enable .....	515
Table 3-560. Function sdio_csm_disable .....	516
Table 3-561. Function sdio_command_index_get .....	516
Table 3-562. Function sdio_response_get .....	517
Table 3-563. Function sdio_data_config .....	518
Table 3-564. Function sdio_data_transfer_config .....	519
Table 3-565. Function sdio_dsm_enable .....	520
Table 3-566. Function sdio_dsm_disable .....	521

Table 3-567. Function <code>sdio_data_write</code> .....	522
Table 3-568. Function <code>sdio_data_read</code> .....	522
Table 3-569. Function <code>sdio_data_counter_get</code> .....	523
Table 3-570. Function <code>sdio_data_counter_get</code> .....	524
Table 3-571. Function <code>sdio_dma_enable</code> .....	524
Table 3-572. Function <code>sdio_dma_disable</code> .....	525
Table 3-573. Function <code>sdio_flag_get</code> .....	525
Table 3-574. Function <code>sdio_flag_clear</code> .....	527
Table 3-575. Function <code>sdio_interrupt_enable</code> .....	528
Table 3-576. Function <code>sdio_interrupt_disable</code> .....	530
Table 3-577. Function <code>sdio_interrupt_flag_get</code> .....	531
Table 3-578. Function <code>sdio_interrupt_flag_clear</code> .....	533
Table 3-579. Function <code>sdio_readwait_enable</code> .....	535
Table 3-580. Function <code>sdio_readwait_disable</code> .....	535
Table 3-581. Function <code>sdio_stop_readwait_enable</code> .....	536
Table 3-582. Function <code>sdio_stop_readwait_disable</code> .....	536
Table 3-583. Function <code>sdio_readwait_type_set</code> .....	537
Table 3-584. Function <code>sdio_operation_enable</code> .....	538
Table 3-585. Function <code>sdio_operation_disable</code> .....	538
Table 3-586. Function <code>sdio_suspend_enable</code> .....	539
Table 3-587. Function <code>sdio_suspend_disable</code> .....	540
Table 3-588. Function <code>sdio_ceata_command_enable</code> .....	540
Table 3-589. Function <code>sdio_ceata_command_disable</code> .....	541
Table 3-590. Function <code>sdio_ceata_interrupt_enable</code> .....	541
Table 3-591. Function <code>sdio_ceata_interrupt_disable</code> .....	542
Table 3-592. Function <code>sdio_ceata_command_completion_enable</code> .....	543
Table 3-593. Function <code>sdio_ceata_command_completion_disable</code> .....	543
Table 3-594. SPI/I2S Registers .....	544
Table 3-595. SPI/I2S firmware function.....	545
Table 3-596. <code>spi_parameter_struct</code> .....	546
Table 3-597. Function <code>spi_i2s_deinit</code> .....	547
Table 3-598. Function <code>spi_struct_para_init</code> .....	547
Table 3-599. Function <code>spi_init</code> .....	548
Table 3-600. Function <code>spi_enable</code> .....	549
Table 3-601. Function <code>spi_disable</code> .....	550
Table 3-602. Function <code>i2s_init</code> .....	550
Table 3-603. Function <code>i2s_psc_config</code> .....	552
Table 3-604. Function <code>i2s_enable</code> .....	553
Table 3-605. Function <code>i2s_disable</code> .....	554
Table 3-606. Function <code>spi_nss_output_enable</code> .....	555
Table 3-607. Function <code>spi_nss_output_disable</code> .....	555
Table 3-608. Function <code>spi_nss_internal_high</code> .....	556
Table 3-609. Function <code>spi_nss_internal_low</code> .....	557
Table 3-610. Function <code>spi_dma_enable</code> .....	557

---

Table 3-611. Function spi_dma_disable .....	558
Table 3-612. Function spi_i2s_data_frame_format_config.....	559
Table 3-613. Function spi_i2s_data_transmit.....	560
Table 3-614. Function spi_i2s_data_receive.....	560
Table 3-615. Function spi_bidirectional_transfer_config .....	561
Table 3-616. Function spi_crc_polynomial_set.....	562
Table 3-617. Function spi_crc_polynomial_get.....	563
Table 3-618. Function spi_crc_on .....	563
Table 3-619. Function spi_crc_off .....	564
Table 3-620. Function spi_crc_next .....	565
Table 3-621. Function spi_crc_get .....	565
Table 3-622. Function spi_ti_mode_enable.....	566
Table 3-623. Function spi_ti_mode_disable .....	567
Table 3-624. Function spi_nssp_mode_enable .....	567
Table 3-625. Function spi_nssp_mode_disable .....	568
Table 3-626. Function qspi_enable .....	569
Table 3-627. Function qspi_disable .....	569
Table 3-628. Function qspi_write_enable.....	570
Table 3-629. Function qspi_read_enable.....	571
Table 3-630. Function qspi_io23_output_enable .....	571
Table 3-631. Function qspi_io23_output_disable .....	572
Table 3-632. Function spi_i2s_interrupt_enable .....	573
Table 3-633. Function spi_i2s_interrupt_disable .....	573
Table 3-634. Function spi_i2s_interrupt_flag_get.....	574
Table 3-635. Function spi_i2s_flag_get .....	576
Table 3-636. Function spi_crc_error_clear.....	577
Table 3-637. TIMERx Registers .....	578
Table 3-638. TIMERx firmware function.....	579
Table 3-639. Structure timer_parameter_struct.....	582
Table 3-640. Structure timer_break_parameter_struct.....	582
Table 3-641. Structure timer_oc_parameter_struct .....	583
Table 3-642. Structure timer_ic_parameter_struct .....	583
Table 3-643. Function timer_deinit.....	583
Table 3-644. Function timer_struct_para_init.....	584
Table 3-645. Function timer_init .....	585
Table 3-646. Function timer_enable .....	586
Table 3-647. Function timer_disable .....	586
Table 3-648. Function timer_auto_reload_shadow_enable .....	587
Table 3-649. Function timer_auto_reload_shadow_disable .....	588
Table 3-650. Function timer_update_event_enable .....	588
Table 3-651. Function timer_update_event_disable .....	589
Table 3-652. Function timer_counter_alignment.....	590
Table 3-653. Function timer_counter_up_direction.....	591
Table 3-654. timer_counter_down_direction .....	591

Table 3-655. Function timer_prescaler_config .....	592
Table 3-656. Function timer_repetition_value_config .....	593
Table 3-657. Function timer_autoreload_value_config .....	594
Table 3-658. Function timer_counter_value_config .....	594
Table 3-659. Function timer_counter_read .....	595
Table 3-660. Function timer_prescaler_read .....	596
Table 3-661. Function timer_single_pulse_mode_config .....	596
Table 3-662. Function timer_update_source_config .....	597
Table 3-663. Function timer_dma_enable .....	598
Table 3-664. Function timer_dma_disable .....	599
Table 3-665. Function timer_channel_dma_request_source_select .....	600
Table 3-666. Function timer_dma_transfer_config .....	601
Table 3-667. Function timer_event_software_generate .....	603
Table 3-668. Function timer_break_struct_para_init .....	604
Table 3-669. Function timer_break_config .....	605
Table 3-670. Function timer_break_enable .....	606
Table 3-671. Function timer_break_disable .....	607
Table 3-672. Function timer_automatic_output_enable .....	607
Table 3-673. Function timer_automatic_output_disable .....	608
Table 3-674. Function timer_primary_output_config .....	609
Table 3-675. Function timer_channel_control_shadow_config .....	609
Table 3-676. Function timer_channel_control_shadow_update_config .....	610
Table 3-677. Function timer_channel_output_struct_para_init .....	611
Table 3-678. Function timer_channel_output_config .....	612
Table 3-679. Function timer_channel_output_mode_config .....	613
Table 3-680. Function timer_channel_output_pulse_value_config .....	614
Table 3-681. Function timer_channel_output_shadow_config .....	615
Table 3-682. Function timer_channel_output_fast_config .....	616
Table 3-683. Function timer_channel_output_clear_config .....	618
Table 3-684. Function timer_channel_output_polarity_config .....	619
Table 3-685. Function timer_channel_complementary_output_polarity_config .....	620
Table 3-686. Function timer_channel_output_state_config .....	621
Table 3-687. Function timer_channel_complementary_output_state_config .....	622
Table 3-688. Function timer_channel_input_struct_para_init .....	623
Table 3-689. Function timer_input_capture_config .....	623
Table 3-690. Function timer_channel_input_capture_prescaler_config .....	625
Table 3-691. Function timer_channel_capture_value_register_read .....	626
Table 3-692. Function timer_input_pwm_capture_config .....	627
Table 3-693. Function timer_hall_mode_config .....	628
Table 3-694. Function timer_input_trigger_source_select .....	628
Table 3-695. Function timer_master_output_trigger_source_select .....	630
Table 3-696. Function timer_slave_mode_select .....	631
Table 3-697. Function timer_master_slave_mode_config .....	632
Table 3-698. Function timer_external_trigger_config .....	633

Table 3-699. Function timer_quadrature_decoder_mode_config.....	634
Table 3-700. Function timer_internal_clock_config.....	636
Table 3-701. Function timer_internal_trigger_as_external_clock_config .....	636
Table 3-702. Function timer_external_trigger_as_external_clock_config.....	637
Table 3-703. Function timer_external_clock_mode0_config.....	638
Table 3-704. Function timer_external_clock_mode1_config.....	640
Table 3-705. Function timer_external_clock_mode1_disable .....	641
Table 3-706. Function timer_write_chxval_register_config.....	641
Table 3-707. Function timer_output_value_selection_config .....	642
Table 3-708. Function timer_interrupt_enable.....	643
Table 3-709. Function timer_interrupt_disable.....	644
Table 3-710. Function timer_interrupt_flag_get .....	645
Table 3-711. Function timer_interrupt_flag_clear .....	646
Table 3-712. Function timer_flag_get .....	647
Table 3-713. Function timer_flag_clear .....	648
Table 3-714. USART Registers.....	650
Table 3-715. USART firmware function.....	650
Table 3-716. Function usart_deinit.....	652
Table 3-717. Function usart_baudrate_set.....	653
Table 3-718. Function usart_parity_config .....	654
Table 3-719. Function usart_word_length_set .....	655
Table 3-720. Function usart_stop_bit_set.....	656
Table 3-721. Function usart_enable .....	656
Table 3-722. Function usart_disable .....	657
Table 3-723. Function usart_transmit_config .....	658
Table 3-724. Function usart_receive_config.....	659
Table 3-725. Function usart_data_first_config.....	660
Table 3-726. Function usart_invert_config .....	660
Table 3-727. Function usart_receiver_timeout_enable .....	661
Table 3-728. Function usart_receiver_timeout_disable .....	662
Table 3-729. Function usart_receiver_timeout_threshold_config .....	663
Table 3-730. Function usart_data_transmit .....	663
Table 3-731. Function usart_data_receive .....	664
Table 3-732. Function usart_address_config .....	665
Table 3-733. Function usart_mute_mode_enable .....	666
Table 3-734. Function usart_mute_mode_disable .....	666
Table 3-735. Function usart_mute_mode_wakeup_config .....	667
Table 3-736. Function usart_lin_mode_enable.....	668
Table 3-737. Function usart_lin_mode_disable.....	669
Table 3-738. Function usart_lin_break_dection_length_config .....	669
Table 3-739. Function usart_send_break .....	670
Table 3-740. Function usart_halfduplex_enable .....	671
Table 3-741. Function usart_halfduplex_disable .....	672
Table 3-742. Function usart_synchronous_clock_enable .....	672

Table 3-743. Function usart_synchronous_clock_disable .....	673
Table 3-744. Function usart_synchronous_clock_config .....	674
Table 3-745. Function usart_guard_time_config .....	675
Table 3-746. Function usart_smartcard_mode_enable .....	675
Table 3-747. Function usart_smartcard_mode_disable .....	676
Table 3-748. Function usart_smartcard_mode_nack_enable .....	677
Table 3-749. Function usart_smartcard_mode_nack_disable .....	677
Table 3-750. Function usart_smartcard_autoretry_config .....	678
Table 3-751. Function usart_block_length_config .....	679
Table 3-752. Function usart_irda_mode_enable .....	680
Table 3-753. Function usart_irda_mode_disable .....	680
Table 3-754. Function usart_prescaler_config .....	681
Table 3-755. Function usart_irda_lowpower_config .....	682
Table 3-756. Function usart_hardware_flow_rts_config .....	683
Table 3-757. Function usart_hardware_flow_cts_config .....	683
Table 3-758. Function usart_dma_receive_config .....	684
Table 3-759. Function usart_dma_transmit_config .....	685
Table 3-760. Function usart_flag_get .....	686
Table 3-761. Function usart_flag_clear .....	687
Table 3-762. Function usart_interrupt_enable .....	688
Table 3-763. Function usart_interrupt_disable .....	689
Table 3-764. Function usart_interrupt_flag_get .....	690
Table 3-765. Function usart_interrupt_flag_clear .....	692
Table 3-766. WWDGT Registers .....	693
Table 3-767. WWDGT firmware function .....	694
Table 3-768. Function wwdgt_deinit .....	694
Table 3-769. Function wwdgt_enable .....	695
Table 3-770. Function wwdgt_counter_update .....	695
Table 3-771. Function wwdgt_config .....	696
Table 3-772. Function wwdgt_interrupt_enable .....	697
Table 3-773. Function wwdgt_flag_get .....	697
Table 3-774. Function wwdgt_flag_clear .....	698
Table 4-1. Revision history .....	700

## 1. Introduction

This manual introduces firmware library of GD32F30x 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 GD32F30x 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
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/APIO	General-purpose and alternate-function I/Os
I2C	Inter-integrated circuit interface
MISC	Nested Vectored Interrupt Controller
PMU	Power management unit
RCU	Reset and clock unit
RTC	Real-time Clock
SDIO	Secure digital input/output interface
SPI/I2S	Serial peripheral interface/Inter-IC sound
TIMER	TIMER
USART	Universal synchronous/asynchronous receiver /transmitter
WWDGT	Window watchdog timer
USBFS	Universal serial bus full-speed interface

## 1.1.2. Naming rules

The firmware library naming rules are shown as below:

- The peripherals are shortened in XXX format, such as: ADC. More shorten information of peripherals refer to [Peripherals](#);
- The name of sourcefile and header file are started with “gd32f30x\_”, such as: gd32f30x\_adc.h;
- The constants used only in one file should be defined in the used file; the constants used in many files should be defined in corresponding header file. All the constants are written in uppercase of English letters;
- Registers are handled as constants. The naming of them are written in uppercase of English letters. In most cases, register names are shortened accord with the user manual;
- Variables are written in lowercase, when concluded by several words, underlines should be adapted among words;
- The naming of peripheral functions are started with the peripheral abbreviation added with an underline, when the function name is concluded by several words, underlines should be adapted among words, and all the peripheral functions are written in lowercase.

## 2. Firmware Library Overview

### 2.1. File Structure of Firmware Library

GD32F30x\_Firmware\_Library, the file structure is shown as below:

**Figure 2-1. File structure of firmware library of GD32F30x**

- ▲  GD32F30x\_Firmware\_Library
  - ▲  Examples
    - ▷  ADC
    - ▷  BKP
    - ▷  CAN
    - ▷  CRC
    - ▷  CTC
    - ▷  DAC
    - ▷  DBG
    - ▷  DMA
    - ▷  ENET
    - ▷  EXMC
    - ▷  EXTI
    - ▷  FMC
    - ▷  FWDGT
    - ▷  GPIO
    - ▷  I2C
    - ▷  PMU
    - ▷  RTC
    - ▷  SDIO
    - ▷  SPI
    - ▷  TIMER
    - ▷  USART
    - ▷  USBD
    - ▷  USBFS
    - ▷  WWDGT
  - ▲  Firmware
    - ▷  CMSIS
    - ▷  GD32F30x\_standard\_peripheral
    - ▷  GD32F30x\_usbd\_driver
    - ▷  GD32F30x\_usbfs\_driver
  - ▲  Template
    - ▷  IAR\_project
    - ▷  Keil\_project
  - ▲  Utilities
    - ▷  Binary
    - ▷  LCD\_Common

### 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;
- gd32f30x\_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);
- gd32f30x\_it.c: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- gd32f30x\_it.h: the header file include all the prototypes of the interrupt service routines;
- systick.c: the source file include the precise time delay functions by using systick;
- systick.h: the header file include the prototype of the precise time delay functions by using systick;
- main.c: example code. Note: all the examples are not influenced by software IDEs.

### 2.1.2. Firmware Folder

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

- CMSIS subfolder includes the Cortex M4 kernel support files, the startup file based on the Cortex M4 kernel processor, the global header file of GD32F30x and system configuration file;
- GD32F30x\_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;
- GD32F30x\_usbd\_driver subfolder includes all the related files about USBD peripheral:
  - Include subfolder includes the header files of USBD peripheral, users need not modify this folder;
  - Source subfolder includes the source files of USBD peripheral, users need not modify this folder;
- GD32F30x\_usbfs\_driver subfolder includes all the related files about USBFS peripheral:
  - Include subfolder includes the header files of USBFS peripheral, users need not modify this folder;
  - Source subfolder includes the source files of USBFS peripheral, users need not modify this folder;

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

different software IDEs.

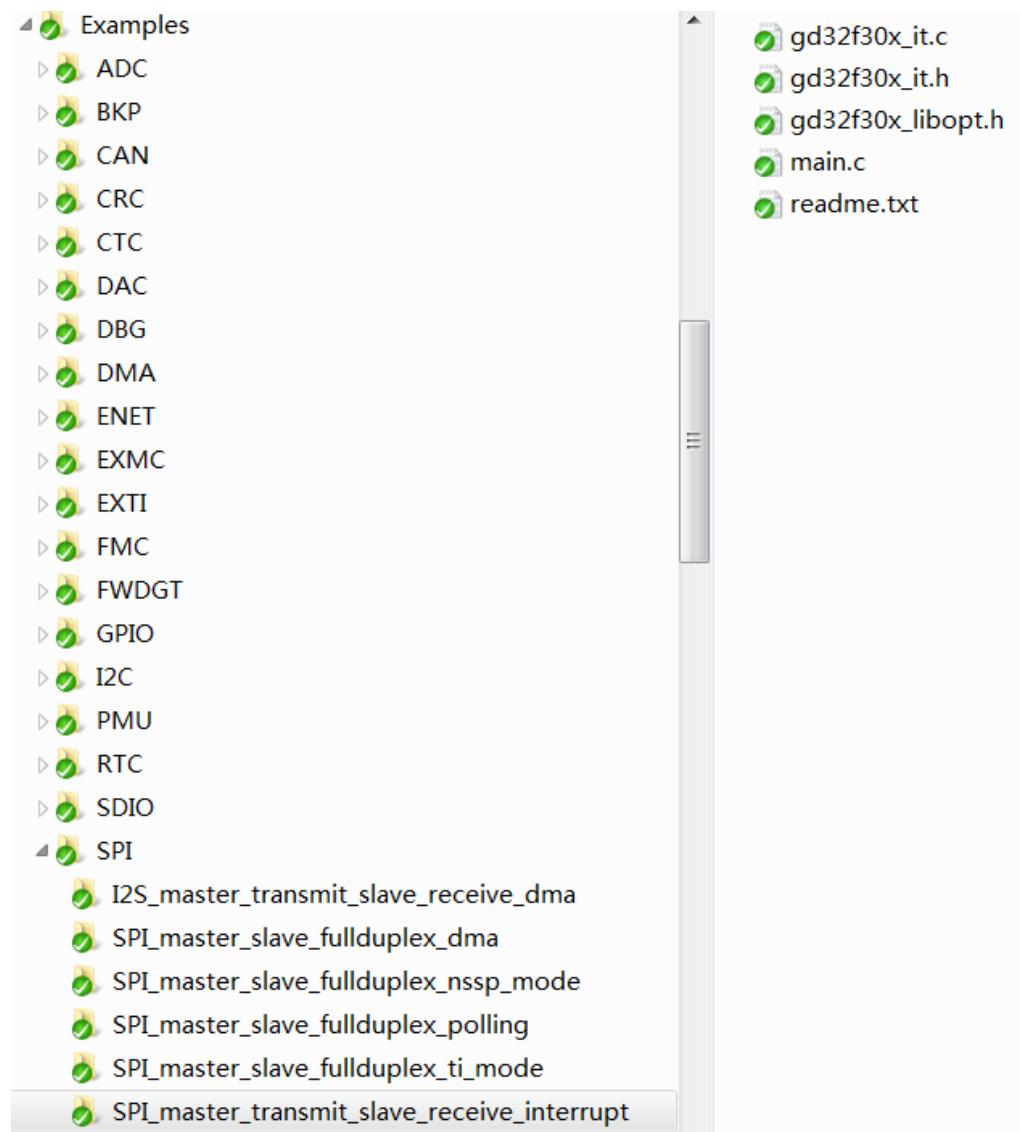
### 2.1.3. Template Folder

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

#### Select files

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

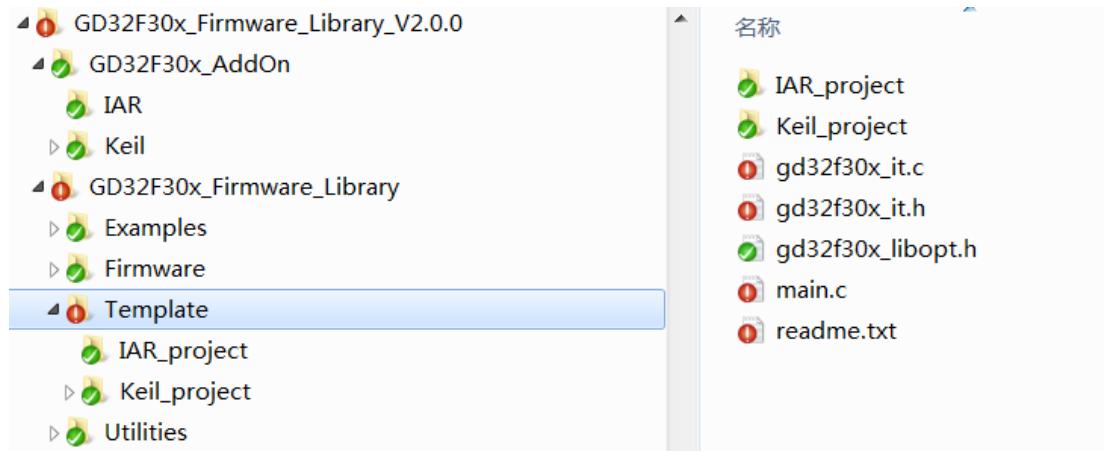
**Figure 2-2. Select peripheral example files**



## Copy files

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

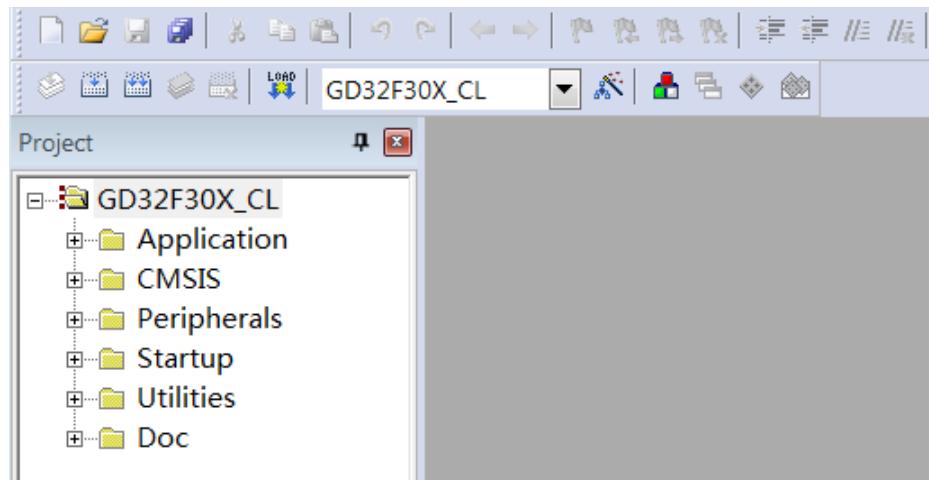
**Figure 2-3. Copy the peripheral example files**



## Open a project

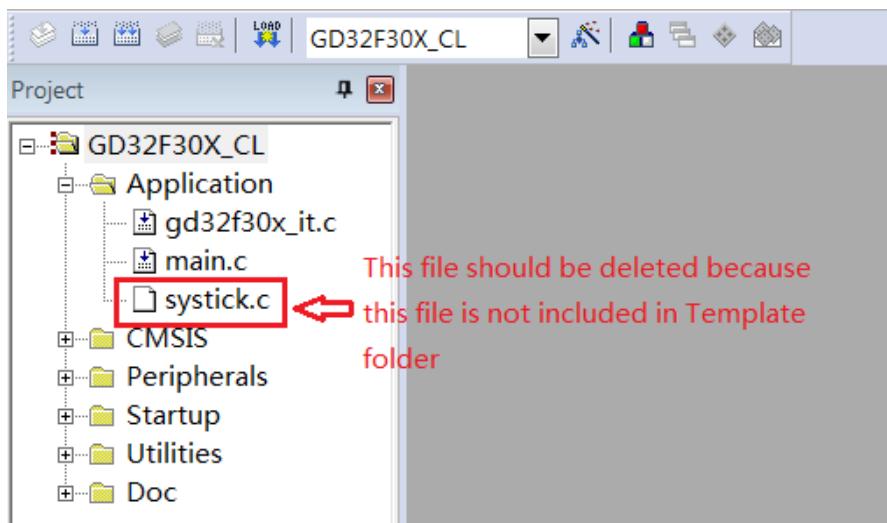
GD provides project in Keil and IAR, users can open project in different IDEs according to their need, such as ”Keil\_project”, open \Template\Keil\_project\Project.uvproj, shown as below:

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



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

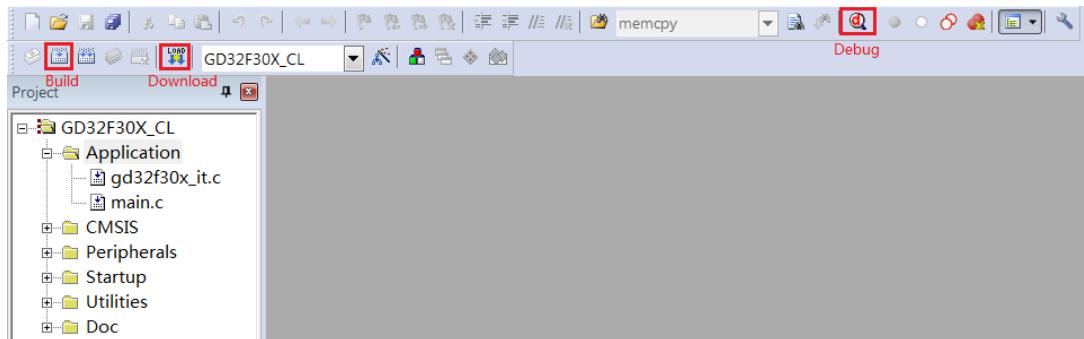
**Figure 2-5. Configure project files**



## Compile-Debug-Download

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

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



### 2.1.4. Utilities Folder

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

- Binary, LCD\_Common subfolders include files for USB tests;
- gd32f30x\_eval.h and gd32f307c\_lcd\_eval.h are related header files of the evaluation board about running the firmware examples;
- gd32f30x\_eval.c and gd32f307c\_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
gd32f30x_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.
gd32f30x_it.h	Header file, including all the prototypes of interrupt service routines.
gd32f30x_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.
gd32f30x_xxx.h	The header file of peripheral PPP, including functions about peripheral PPP, and the variables used for functions.
gd32f30x_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 functin
<b>Input parameter{in}</b>	
<b>Input parameter name</b>	Description
xxxx	Description of input parameters
<b>Output parameter{out}</b>	
<b>Output parameter name</b>	Description
xxxx	Description of output parameters
<b>Return value</b>	
<b>Return value type</b>	The range of return value

#### 3.2. ADC

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

##### 3.2.1. Descriptions of Peripheral registers

ADC registers are listed in the table shown as below:

**Table 3-2. ADC Registers**

Registers	Descriptions
ADC_STAT	Status register
ADC_CTL0	Control register 0
ADC_CTL1	Control register 1
ADC_SAMPT0	Sample time register 0
ADC_SAMPT1	Sample time register 1
ADC_IOFFx (x=0..3)	Inserted channel data offset register x(x=0..3)
ADC_WDHT	Watchdog high threshold register
ADC_WDLT	Watchdog low threshold register
ADC_RSQ0	Regular sequence register 0
ADC_RSQ1	Regular sequence register 1
ADC_RSQ2	Regular sequence register 2
ADC_ISQ	Inserted sequence register
ADC_IDATAx	Inserted data register x(x=0..3)
ADC_RDATA	Regular data register
ADC_OVSAMPCTL	Oversample control register

### 3.2.2. Descriptions of Peripheral functions

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

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

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

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

Function name	Function description
adc_watchdog_threshold_config	configure ADC analog watchdog threshold
adc_oversample_mode_config	configure ADC oversample mode
adc_oversample_mode_enable	enable ADC oversample mode
adc_oversample_mode_disable	disable ADC oversample mode

## adc\_deinit

The description of adc\_deinit is shown as below:

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

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

Example:

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

## adc\_enable

The description of adc\_enable is shown as below:

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

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

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

Example:

```
/* enable ADC0 */
adc_enable(ADC0);
```

### adc\_disable

The description of adc\_disable is shown as below:

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

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

Example:

```
/* disable ADC0 */

adc_disable(ADC0);
```

### **adc\_calibration\_enable**

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

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

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

Example:

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

adc_calibration_enable(ADC0);
```

### **adc\_dma\_mode\_enable**

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

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

<b>Function name</b>	adc_dma_mode_enable
<b>Function prototype</b>	void adc_dma_mode_enable(uint32_t adc_periph);
<b>Function descriptions</b>	enable ADCx DMA request
<b>Precondition</b>	-

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

Example:

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

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

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

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

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

Example:

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

adc\_dma\_mode\_disable(ADC0);

### **adc\_tempsensor\_vrefint\_enable**

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

**Table 3-10. Function adc\_tempsensor\_vrefint\_enable**

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

Example:

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

### **adc\_tempsensor\_vrefint\_disable**

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

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

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

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

adc_tempsensor_vrefint_disable();
```

### adc\_resolution\_config

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

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

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

-	-
Return value	
-	-

Example:

```
/* configure ADC0 data alignment */
adc_data_alignment_config(ADC0, ADC_DATAALIGN_RIGHT);
```

### adc\_discontinuous\_mode\_config

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

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

<b>Function name</b>	adc_discontinuous_mode_config
<b>Function prototype</b>	void adc_discontinuous_mode_config(uint32_t adc_periph, uint8_t adc_channel_group, uint8_t length);
<b>Function descriptions</b>	configure ADC discontinuous mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<b>ADCx(x=0..2)</b>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_channel_group</b>	select the channel group
<b>ADC_REGULAR_CHA_NNEL</b>	regular channel group
<b>ADC_INSERTED_CHA_NNEL</b>	inserted channel group
<b>ADC_CHANNEL_DISC_ON_DISABLE</b>	disable discontinuous mode of regular and inserted channel
<b>Input parameter{in}</b>	
<b>length</b>	number of conversions in discontinuous mode, the number can be 1..8 for regular channel, the number has no effect for inserted channel
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* configure ADC0 regular channel group discontinuous mode */
adc_discontinuous_mode_config(ADC0, ADC_REGULAR_CHANNEL, 6);
```

### adc\_mode\_config

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

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

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

<i>D_PARALLEL</i>	
<i>ADC_DAUL_REGULAR_PARALLEL</i>	ADC0 and ADC1 work in regular parallel mode only
<i>ADC_DAUL_REGULAR_FOLLOWUP_FAST</i>	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_TRIGGER_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>	adc_special_function_config
<b>Function prototype</b>	void adc_special_function_config(uint32_t adc_periph, uint32_t function, ControlStatus newvalue);
<b>Function descriptions</b>	enable or disable ADC special function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>adc_periph</i>	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>	
<i>newvalue</i>	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
<i>ADCx(x=0..2)</i>	ADC peripheral selection

Input parameter{in}	
<b>data_alignment</b>	data alignment select
<i>ADC_DATAALIGN_RIG HT</i>	LSB alignment
<i>ADC_DATAALIGN_LE FT</i>	MSB alignment
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 data alignment */
adc_data_alignment_config(ADC0, ADC_DATAALIGN_RIGHT);
```

### adc\_channel\_length\_config

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

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

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

<b>ADC_INSERTED_CHANNEL</b>	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
<b>ADCx(x=0..2)</b>	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
<b>ADC_CHANNEL_x(x=0..17)</b>	ADC Channelx (x=0..17)(x=16 and x=17 are only for ADC0)

Input parameter{in}	
<b>sample_time</b>	the sample time value
<i>ADC_SAMPLETIME_1POINT5</i>	1.5 cycles
<i>ADC_SAMPLETIME_7POINT5</i>	7.5 cycles
<i>ADC_SAMPLETIME_13POINT5</i>	13.5 cycles
<i>ADC_SAMPLETIME_28POINT5</i>	28.5 cycles
<i>ADC_SAMPLETIME_41POINT5</i>	41.5 cycles
<i>ADC_SAMPLETIME_55POINT5</i>	55.5 cycles
<i>ADC_SAMPLETIME_71POINT5</i>	71.5 cycles
<i>ADC_SAMPLETIME_239POINT5</i>	239.5 cycles
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 regular channel */
adc_regular_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_7POINT5);
```

### adc\_inserted\_channel\_config

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

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

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

<b>Function descriptions</b>		configure ADC inserted channel
<b>Precondition</b>		-
<b>The called functions</b>		-
<b>Input parameter{in}</b>		
<b>adc_periph</b>	ADC peripheral	
<i>ADCx(0,1)</i>	ADC peripheral selection	
<b>Input parameter{in}</b>		
<b>rank</b>	the inserted group sequencer rank, this parameter must be between 0 to 3	
<b>Input parameter{in}</b>		
<b>adc_channel</b>	the selected ADC channel	
<i>ADC_CHANNEL_x(x=0..17)</i>	ADC Channelx (x=0..17)(x=16 and x=17 are only for ADC0)	
<b>Input parameter{in}</b>		
<b>sample_time</b>	the sample time value	
<i>ADC_SAMPLETIME_1POINT5</i>	1.5 cycles	
<i>ADC_SAMPLETIME_7POINT5</i>	7.5 cycles	
<i>ADC_SAMPLETIME_13POINT5</i>	13.5 cycles	
<i>ADC_SAMPLETIME_28POINT5</i>	28.5 cycles	
<i>ADC_SAMPLETIME_41POINT5</i>	41.5 cycles	
<i>ADC_SAMPLETIME_55POINT5</i>	55.5 cycles	
<i>ADC_SAMPLETIME_71POINT5</i>	71.5 cycles	
<i>ADC_SAMPLETIME_239POINT5</i>	239.5 cycles	
<b>Output parameter{out}</b>		

-	-
Return value	
-	-

Example:

```
/* configure ADC0 inserted channel */
adc_inserted_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_7POINT5);
```

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

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

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

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

Example:

---

/\* configure ADC0 inserted channel offset \*/

adc\_inserted\_channel\_offset\_config(ADC0, ADC\_INSERTED\_CHANNEL\_0, 100);

### **adc\_external\_trigger\_config**

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

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

<b>Function name</b>	adc_external_trigger_config
<b>Function prototype</b>	void adc_external_trigger_config(uint32_t adc_periph, uint8_t adc_channel_group, ControlStatus newvalue);
<b>Function descriptions</b>	configure ADC external trigger
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<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>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 inserted channel group external trigger \*/

---

adc\_external\_trigger\_config(ADC0, ADC\_INSERTED\_CHANNEL\_0, ENABLE);

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

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

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

<b>Function name</b>	adc_external_trigger_source_config
<b>Function prototype</b>	void adc_external_trigger_source_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t external_trigger_source);
<b>Function descriptions</b>	configure ADC external trigger source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
<b>Input parameter{in}</b>	
adc_channel_group	select the channel group
ADC_REGULAR_CHANNEL	regular channel group
ADC_INSERTED_CHANNEL	inserted channel group
<b>Input parameter{in}</b>	
external_trigger_source	regular or inserted group trigger source
ADC0_1_EXTTRIG_REGULAR_T0_CH0	TIMER0 CH0 event select for regular channel
ADC0_1_EXTTRIG_REGULAR_T0_CH1	TIMER0 CH1 event select for regular channel
ADC0_1_EXTTRIG_REGULAR_T0_CH2	TIMER0 CH2 event select for regular channel
ADC0_1_EXTTRIG_REGULAR_T1_CH1	TIMER1 CH1 event select for regular channel
ADC0_1_EXTTRIG_REGULAR_T2_CH0	TIMER2 TRGO event select for regular channel

<i>GULAR_T2_TRGO</i>	
<i>ADC0_1_EXTTRIG_REGULAR_T3_CH3</i>	TIMER3 CH3 event select for regular channel
<i>ADC0_1_EXTTRIG_REGULAR_T7_TRGO</i>	TIMER7 TRGO event select for regular channel
<i>ADC0_1_EXTTRIG_REGULAR_EXTI_11</i>	external interrupt line 11 for regular channel
<i>ADC2_EXTTRIG_REGULAR_T2_CH0</i>	TIMER2 CH0 event select for regular channel
<i>ADC2_EXTTRIG_REGULAR_T1_CH2</i>	TIMER1 CH2 event select for regular channel
<i>ADC2_EXTTRIG_REGULAR_T0_CH2</i>	TIMER0 CH2 event select for regular channel
<i>ADC2_EXTTRIG_REGULAR_T7_CH0</i>	TIMER7 CH0 event select for regular channel
<i>ADC2_EXTTRIG_REGULAR_T7_TRGO</i>	TIMER7 TRGO event select for regular channel
<i>ADC2_EXTTRIG_REGULAR_T4_CH0</i>	TIMER4 CH0 event select for regular channel
<i>ADC2_EXTTRIG_REGULAR_T4_CH2</i>	TIMER4 CH2 event select for regular channel
<i>ADC0_1_2_EXTTRIG_REGULAR_NONE</i>	software trigger for regular channel
<i>ADC0_1_EXTTRIG_INSERTED_T0_TRGO</i>	TIMER0 TRGO event select for inserted channel
<i>ADC0_1_EXTTRIG_INSERTED_T0_CH3</i>	TIMER0 CH3 event select for inserted channel
<i>ADC0_1_EXTTRIG_INSERTED_T1_TRGO</i>	TIMER1 TRGO event select for inserted channel
<i>ADC0_1_EXTTRIG_INSERTED_T1_CH0</i>	TIMER1 CH0 event select for inserted channel
<i>ADC0_1_EXTTRIG_INSERTED_T2_CH3</i>	TIMER2 CH3 event select for inserted channel
<i>ADC0_1_EXTTRIG_INSERTED_T3_TRGO</i>	TIMER3 TRGO event select for inserted channel

<i>ADC0_1_EXTTRIG_IN SERTED_EXTI_15</i>	external interrupt line 15 for inserted channel
<i>ADC0_1_EXTTRIG_IN SERTED_T7_CH3</i>	TIMER7 CH3 event select for inserted channel
<i>ADC2_EXTTRIG_INSE RTED_T0_TRGO</i>	TIMER0 TRGO event select for inserted channel
<i>ADC2_EXTTRIG_INSE RTED_T0_CH3</i>	TIMER0 CH3 event select for inserted channel
<i>ADC2_EXTTRIG_INSE RTED_T3_CH2</i>	TIMER3 CH2 event select for inserted channel
<i>ADC2_EXTTRIG_INSE RTED_T7_CH1</i>	TIMER7 CH1 event select for inserted channel
<i>ADC2_EXTTRIG_INSE RTED_T7_CH3</i>	TIMER7 CH3 event select for inserted channel
<i>ADC2_EXTTRIG_INSE RTED_T4_TRGO</i>	TIMER4 TRGO event select for inserted channel
<i>ADC2_EXTTRIG_INSE RTED_T4_CH3</i>	TIMER4 CH3 event select for inserted channel
<i>ADC0_1_2_EXTTRIG_I NSERTED_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 3-23. Function adc\_software\_trigger\_enable**

<b>Function name</b>	adc_software_trigger_enable
----------------------	-----------------------------

<b>Function prototype</b>	void adc_software_trigger_enable(uint32_t adc_periph, uint8_t adc_channel_group);
<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 3-24. 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>	-

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

Example:

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

uint16_t adc_value = 0;

adc_value = adc_regular_data_read(ADC0);
```

### **adc\_inserted\_data\_read**

The description of `adc_inserted_data_read` is shown as below:

**Table 3-25. Function `adc_inserted_data_read`**

<b>Function name</b>	adc_inserted_data_read
<b>Function prototype</b>	<code>uint16_t adc_inserted_data_read(uint32_t adc_periph, uint8_t inserted_channel);</code>
<b>Function descriptions</b>	read ADC inserted group data register
<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>	insert channel select
<i>ADC_INSERTED_CHANNEL(x=0..3)</i>	inserted Channelx, x=0,1,2,3
Output parameter{out}	
-	-

Return value	
uint16_t	ADC conversion value (0-0xFFFF)

Example:

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

uint16_t adc_value = 0;

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

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

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

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

<b>Function name</b>	adc_sync_mode_convert_value_read
<b>Function prototype</b>	uint32_t adc_sync_mode_convert_value_read(void);
<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>	
uint32_t	ADC conversion value (0-0xFFFFFFFF)

Example:

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

uint32_t adc_value = 0;

adc_value = adc_sync_mode_convert_value_read();
```

### adc\_flag\_get

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

**Table 3-27. Function adc\_flag\_get**

<b>Function name</b>	adc_flag_get
----------------------	--------------

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

Example:

```
/* get the ADC0 analog watchdog flag bits*/
FlagStatus flag_value;
flag_value = adc_flag_get(ADC0, ADC_FLAG_WDE);
```

### adc\_flag\_clear

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

**Table 3-28. Function adc\_flag\_clear**

<b>Function name</b>	adc_flag_clear
<b>Function prototype</b>	void adc_flag_clear(uint32_t adc_periph, uint32_t adc_flag);
<b>Function descriptions</b>	clear the ADC flag bits

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<b>ADCx(x=0..2)</b>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_flag</b>	the adc flag bits
<b>ADC_FLAG_WDE</b>	analog watchdog event flag
<b>ADC_FLAG_EOC</b>	end of group conversion flag
<b>ADC_FLAG_EOIC</b>	end of inserted group conversion flag
<b>ADC_FLAG_STIC</b>	start flag of inserted channel group
<b>ADC_FLAG_STRC</b>	start flag of regular channel group
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the ADC0 analog watchdog flag bits*/
adc_flag_clear(ADC0, ADC_FLAG_WDE);
```

### **adc\_interrupt\_flag\_get**

The description of `adc_interrupt_flag_get` is shown as below:

**Table 3-29. Function `adc_interrupt_flag_get`**

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

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

Example:

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

### adc\_interrupt\_flag\_clear

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

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

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

<b>adc_interrupt</b>	the adc interrupt bits
<i>ADC_INT_FLAG_WDE</i>	analog watchdog interrupt
<i>ADC_INT_FLAG_EOC</i>	end of group conversion interrupt
<i>ADC_INT_FLAG_EOIC</i>	end of inserted group conversion interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the ADC0 analog watchdog interrupt bits*/
adc_interrupt_flag_clear(ADC0, ADC_INT_WDE);
```

### **adc\_interrupt\_enable**

The description of `adc_interrupt_enable` is shown as below:

**Table 3-31. Function `adc_interrupt_enable`**

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

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

Example:

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

adc_interrupt_enable(ADC0, ADC_INT_WDE);
```

### adc\_interrupt\_disable

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

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

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

Example:

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

---

```
adc_interrupt_disable(ADC0, ADC_INT_WDE);
```

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

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

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

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

Example:

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

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

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

**Table 3-34. Function adc\_watchdog\_group\_channel\_enable**

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

	adc_channel_group);
<b>Function descriptions</b>	configure ADC analog watchdog group channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
<b>Input parameter{in}</b>	
adc_channel_group	the channel group use analog watchdog
ADC_REGULAR_CHANNEL	regular channel group
ADC_INSERTED_CHANNEL	inserted channel group
ADC_REGULAR_INSERTED_CHANNEL	both regular and inserted group
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### adc\_watchdog\_disable

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

**Table 3-35. Function adc\_watchdog\_disable**

<b>Function name</b>	adc_watchdog_disable
<b>Function prototype</b>	void adc_watchdog_disable(uint32_t adc_periph);
<b>Function descriptions</b>	disable ADC analog watchdog
<b>Precondition</b>	-

<b>The called functions</b>	
<b>Input parameter{in}</b>	
adc_periph	ADC peripheral
ADCx(0, 1)	ADC peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### adc\_watchdog\_threshold\_config

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

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

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

-	-
Return value	
-	-

Example:

```
/* configure ADC0 analog watchdog threshold */
adc_watchdog_threshold_config(ADC0, 0x0400, 0x0A00);
```

### adc\_oversample\_mode\_config

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

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

<b>Function name</b>	adc_oversample_mode_config
<b>Function prototype</b>	void adc_oversample_mode_config(uint32_t adc_periph, uint32_t mode, uint16_t shift, uint8_t ratio);
<b>Function descriptions</b>	configure ADC oversample mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<b>ADCx(x=0..2)</b>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>mode</b>	ADC oversampling mode
<b>ADC_OVERSAMPLING_ALL_CONVERT</b>	all oversampled conversions for a channel are done consecutively after a trigger
<b>ADC_OVERSAMPLING_ONE_CONVERT</b>	each oversampled conversion for a channel needs a trigger
<b>Input parameter{in}</b>	
<b>shift</b>	ADC oversampling shift
<b>ADC_OVERSAMPLING_SHIFT_NONE</b>	no oversampling shift
<b>ADC_OVERSAMPLING_SHIFT_1B</b>	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
<i>ADC_OVERSAMPLING_RATIO_MUL32</i>	oversampling ratio multiple 32
<i>ADC_OVERSAMPLING_RATIO_MUL64</i>	oversampling ratio multiple 64
<i>ADC_OVERSAMPLING_RATIO_MUL128</i>	oversampling ratio multiple 128
<i>ADC_OVERSAMPLING_RATIO_MUL256</i>	oversampling ratio multiple 256
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

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

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

### adc\_oversample\_mode\_enable

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

**Table 3-38. Function adc\_oversample\_mode\_enable**

Function name	adc_oversample_mode_enable
Function prototype	void adc_oversample_mode_enable(uint32_t adc_periph);
Function descriptions	enable ADC oversample mode
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 oversample mode */

adc_oversample_mode_enable (ADC0);
```

### adc\_oversample\_mode\_disable

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

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

Function name	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>	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### 3.3. BKP

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

#### 3.3.1. Descriptions of Peripheral registers

BKP registers are listed in the table shown as below:

**Table 3-40. BKP Registers**

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

Registers	Descriptions
BKP_TPCS	Tamper control and status register

### 3.3.2. Descriptions of Peripheral functions

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

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

Function name	Function description
bkp_deinit	reset data registers
bkp_write_data	write BKP data register
bkp_read_data	read BKP data register
bkp_RTC_calibration_output_enable	enable RTC clock calibration output
bkp_RTC_calibration_output_disable	disable RTC clock calibration output
bkp_RTC_signal_output_enable	enable RTC alarm or second signal output
bkp_RTC_signal_output_disable	disable RTC alarm or second signal output
bkp_RTC_output_select	select RTC output, the RTC output can be select as alarm pulse or second pulse
bkp_RTC_clock_output_select	select RTC clock output
bkp_RTC_clock_calibration_direction	select RTC clock calibration direction
bkp_RTC_calibration_value_set	set RTC clock calibration value
bkp_tamper_detection_enable	enable tamper detection
bkp_tamper_detection_disable	disable tamper detection
bkp_tamper_active_level_set	set tamper pin active level
bkp_tamper_interrupt_enable	enable tamper interrupt
bkp_tamper_interrupt_disable	disable tamper interrupt
bkp_flag_get	get bkp flag state
bkp_flag_clear	clear bkp flag state
bkp_interrupt_flag_get	get bkp interrupt flag state
bkp_interrupt_flag_clear	clear bkp interrupt flag state

### bkp\_deinit

The description of bkp\_deinit is shown as below:

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

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

Example:

```
/* reset BKP registers */

bkp_deinit();
```

### bkp\_write\_data

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

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

<b>Function name</b>	bkp_write_data
<b>Function prototype</b>	void bkp_write_data(bkp_data_register_enum register_number, uint16_t data);
<b>Function descriptions</b>	write BKP data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>register_number</b>	refer to bkp_data_register_enum
<b>BKP_DATA_x(x =</b>	bkp data register number x

	0..41)
<b>Input parameter{in}</b>	
<b>data</b>	the data to be write in BKP data register
0-0xffff	data value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### bkp\_read\_data

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

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

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

Example:

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

```

uint16_t data;

data = bkp _read_data (BKP_DATA_0);

```

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

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

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

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

Example:

```

/* enable RTC clock calibration output */

bkp_rtc_calibration_output_enable();

```

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

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

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

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

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

Example:

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

bkp_rtc_calibration_output_disable();
```

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

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

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

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

Example:

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

bkp_rtc_signal_output_enable();
```

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

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

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

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

Example:

```
/* disable RTC alarm or second signal output */
bkp_rtc_signal_output_disable();
```

### **bkp\_rtc\_output\_select**

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

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

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

<i>ND_PULSE</i>	
<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 3-50. Function bkp\_rtc\_clock\_output\_select**

<b>Function name</b>	bkp_rtc_clock_output_select
<b>Function prototype</b>	void bkp_rtc_clock_output_select(uint16_t clocksel);
<b>Function descriptions</b>	select RTC clock output, the RTC clock output can be select as divided 64 or no division
<b>Precondition</b>	-
<b>The called functions</b>	-
<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 devided 64 to output */
bkp_rtc_clock_output_select (RTC_CLOCK_DIV_64);
```

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

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

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

<b>Function name</b>	bkp_rtc_clock_calibration_direction
<b>Function prototype</b>	void bkp_rtc_clock_calibration_direction(uint16_t direction);
<b>Function descriptions</b>	select RTC clock calibration direction, the RTC clock calibration direction can be select as slowed down or speed up
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>direction</b>	RTC clock calibration direction
<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 3-52. 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
0x00 - 0x7F	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 3-53. 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 3-54. 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 3-55. 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
<b>TAMPER_PIN_ACTIVE_HIGH</b>	the tamper pin is active high

<i>TAMPER_PIN_ACTIVE_LOW</i>	the tamper pin is active low
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

bkp_tamper_active_level_set (TAMPER_PIN_ACTIVE_HIGH);
```

### **bkp\_tamper\_interrupt\_enable**

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

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

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

Example:

```
/* enable tamper pin interrupt */

bkp_tamper_interrupt_enable ();
```

### **bkp\_tamper\_interrupt\_disable**

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

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

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

Example:

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

### **bkp\_flag\_get**

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

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

<b>Function name</b>	bkp_flag_get
<b>Function prototype</b>	FlagStatus bkp_flag_get(uint16_t flag);
<b>Function descriptions</b>	get bkp flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	bkp flag state
<b>BKP_FLAG_TAMPER</b>	tamper event flag
<b>Output parameter{out}</b>	
-	-

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

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

Example:

```
/* clear BKP flag state */

bkp_flag_clear (BKP_FLAG_TAMPER);
```

### bkp\_interrupt\_flag\_get

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

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

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

Example:

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

bkp_interrupt_flag_get (BKP_INT_FLAG_TAMPER);
```

### **bkp\_interrupt\_flag\_clear**

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

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

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

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear BKP interrupt flag state */
bkp_interrupt_flag_clear (BKP_INT_FLAG_TAMPER);
```

## 3.4. CAN

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

### 3.4.1. Descriptions of Peripheral registers

CAN registers are listed in the table shown as below:

**Table 3-62. CAN Registers**

Registers	Descriptions
CAN_CTL	Control register
CAN_STAT	Status register
CAN_TSTAT	Transmit status register
CAN_RFIFO0	Receive message FIFO0 register
CAN_RFIFO1	Receive message FIFO1 register
CAN_INTEN	Interrupt enable register
CAN_ERR	Error register
CAN_BT	Bit timing register
CAN_TMIx	Transmit mailbox identifier register
CAN_TMPx	Transmit mailbox property register
CAN_TMDATA0x	Transmit mailbox data0 register
CAN_TMDATA1x	Transmit mailbox data1 register

Registers	Descriptions
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_FA FIFO	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 3-63. CAN firmware function**

Function name	Function description
can_deinit	deinitialize CAN
can_struct_para_init	initialize CAN parameter struct with a default value
can_init	initialize CAN
can_filter_init	initialize CAN filter
can1_filter_start_bank	set can1 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

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

## Structure can\_parameter\_struct

Table 3-64. can\_parameter\_struct

Member name	Function description
working_mode	CAN working mode
resync_jump_width	CAN resynchronization jump width
time_segment_1	time segment 1
time_segment_2	time segment 2
time_triggered	time triggered communication mode
auto_bus_off_recovery	automatic bus-off recovery
auto_wake_up	automatic wake-up mode
auto_retrans	automatic retransmission mode

rec_fifo_overwrite	receive FIFO overwrite mode
trans_fifo_order	transmit FIFO order
prescaler	baudrate prescaler

## Structure can\_trasnmit\_message\_struct

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

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

## Structure can\_receive\_message\_struct

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

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

## Structure can\_filter\_parameter\_struct

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

Member name	Function description
filter_list_high	filter list number high bits
filter_list_low	filter list number low bits

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

### can\_deinit

The description of can\_deinit is shown as below:

**Table 3-68. Function can\_deinit**

<b>Function name</b>	can_deinit
<b>Function prototype</b>	void can_deinit(uint32_t can_periph);
<b>Function descriptions</b>	deinitialize CAN
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable/ rcu_periph_reset_disable
<b>Input parameter{in}</b>	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-
<b>Function name</b>	can_deinit
<b>Function prototype</b>	void can_deinit(uint32_t can_periph);
<b>Function descriptions</b>	deinitialize CAN
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable/ rcu_periph_reset_disable
<b>Input parameter{in}</b>	

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

Example:

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

#### **can\_struct\_para\_init**

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

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

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

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

Example:

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

### can\_init

The description of can\_init is shown as below:

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

<b>Function name</b>	can_init
<b>Function prototype</b>	ErrStatus can_init(uint32_t can_periph, can_parameter_struct* can_parameter_init);
<b>Function descriptions</b>	initialize CAN

<b>Precondition</b>	can_struct_para_init()
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
CANx(x=0,1)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Input parameter{in}</b>	
<b>can_parameter_init</b>	CAN parameter initialization stuct, the structure members can refer to members of the structure <a href="#">Table 3-64. can_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS / ERROR
<b>Function name</b>	can_init
<b>Function prototype</b>	ErrStatus can_init(uint32_t can_periph, can_parameter_struct* can_parameter_init);
<b>Function descriptions</b>	initialize CAN
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>can_parameter_init</b>	CAN parameter initialization stuct, the structure members can refer to members of the structure <a href="#">Table 3-64. can_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS / ERROR
<b>Function name</b>	can_init

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

ErrStatus	SUCCESS / ERROR
-----------	-----------------

Example:

```
/* CAN0 initialize*/
can_init (CAN0);
```

### can\_filter\_init

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

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

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

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

Example:

```
/* initialize CAN filter */

can_filter_init(&can_filter);
```

<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>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, only one parameter can be selected which is shown as below
<b>CAN_STANDARD_FIFO0</b>	standard format and store to FIFO0
<b>CAN_STANDARD_FIFO1</b>	standard format and store to FIFO1
<b>CAN_EXTENDED_FIFO0</b>	extended format and store to FIFO0
<b>CAN_EXTENDED_FIFO1</b>	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(0x11, 0x11, CAN_STANDARD_FIFO0, 0);
```

### **can1\_filter\_start\_bank**

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

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

<b>Function name</b>	can1_filter_start_bank
<b>Function prototype</b>	void can1_filter_start_bank(uint8_t start_bank);
<b>Function descriptions</b>	set CAN1 filter start bank number
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>start_bank</b>	CAN1 start bank number
1..27	start number
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set CAN1 filter start bank number 15*/
can1_filter_start_bank (15);
```

### **can\_debug\_freeze\_enable**

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

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

<b>Function name</b>	can_debug_freeze_enable
<b>Function prototype</b>	void can_debug_freeze_enable(uint32_t can_periph);
<b>Function descriptions</b>	enable CAN debug freeze
<b>Precondition</b>	-
<b>The called functions</b>	dbg_periph_enable
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral

CANx(x=0,1)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-
<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>	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable CAN0 debug freeze */

can_debug_freeze_enable (CAN0);
```

### **can\_debug\_freeze\_disable**

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

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

<b>Function name</b>	can_debug_freeze_disable
<b>Function prototype</b>	void can_debug_freeze_disable(uint32_t can_periph);
<b>Function descriptions</b>	disable CAN debug freeze
<b>Precondition</b>	-
<b>The called functions</b>	dbg_periph_disable

Input parameter{in}	
<b>can_periph</b>	CAN peripheral
CANx(x=0,1)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
Output parameter{out}	
-	-
Return value	
-	-
<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
Input parameter{in}	
<b>can_periph</b>	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable CAN0 debug freeze */
can_debug_freeze_disable (CAN0);
```

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

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

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

<b>Function name</b>	can_time_trigger_mode_enable
<b>Function prototype</b>	void can_time_trigger_mode_enable(uint32_t can_periph);
<b>Function descriptions</b>	enable CAN time trigger mode

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<b>CANx(x=0,1)</b>	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<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
<b>CANx(x=0,1)</b>	CAN peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

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

<b>Function name</b>	can_time_trigger_mode_disable
----------------------	-------------------------------

<b>Function prototype</b>	void can_time_trigger_mode_disable(uint32_t can_periph);
<b>Function descriptions</b>	disable CAN time trigger mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-
<b>Function name</b>	can_time_trigger_mode_disable
<b>Function prototype</b>	void can_time_trigger_mode_disable(uint32_t can_periph);
<b>Function descriptions</b>	disable CAN time trigger mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable CAN0 time trigger mode */
can_time_trigger_mode_disable (CAN0);
```

### can\_message\_transmit

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

Table 3-77. Function can\_message\_transmit

<b>Function name</b>	can_message_transmit
<b>Function prototype</b>	uint8_t can_message_transmit(uint32_t can_periph, can_trasnmit_message_struct* transmit_message);
<b>Function descriptions</b>	transmit CAN message
<b>Precondition</b>	can_struct_para_init()
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Input parameter{in}</b>	
transmit_message	CAN transmit message stuct, the structure members can refer to members of the structure <a href="#">Table 3-65. can_trasnmit_message_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint8_t	0x00-0x03
<b>Function name</b>	can_message_transmit
<b>Function prototype</b>	uint8_t can_message_transmit(uint32_t can_periph, can_trasnmit_message_struct* transmit_message);
<b>Function descriptions</b>	transmit CAN message
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection
<b>Input parameter{in}</b>	
transmit_message	CAN transmit message stuct, the structure members can refer to members of the structure <a href="#">Table 3-65. can_trasnmit_message_struct</a>
<b>Output parameter{out}</b>	

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

<b>transmit_message</b>	CAN transmit message stuct, the structure members can refer to members of the structure <a href="#">Table 3-65. can_trasnmit_message_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	0x00-0x03

Example:

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

### can\_transmit\_states

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

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

<b>Function name</b>	can_transmit_states
<b>Function prototype</b>	can_transmit_state_enum can_transmit_states(uint32_t can_periph, uint8_t mailbox_number);
<b>Function descriptions</b>	get CAN transmit state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Input parameter{in}</b>	
<b>mailbox_number</b>	Mailbox number
CAN_MAILBOXX	CAN_MAILBOXX(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>can_transmit_state_e</b>	0..4

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

Example:

```

/* CAN0 mailbox0 transmit state */

uint8_t transmit_state = 0;

transmit_state = can_transmit_states (CAN0, CAN_MAILBOX0);

```

### can\_transmission\_stop

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

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

<b>Function name</b>	can_transmission_stop
<b>Function prototype</b>	void can_transmission_stop(uint32_t can_periph, uint8_t mailbox_number);
<b>Function descriptions</b>	stop CAN transmission
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<b>CANx(x=0, 1)</b>	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Input parameter{in}</b>	
<b>mailbox_number</b>	Mailbox number
<b>CAN_MAILBOXx</b>	CAN_MAILBOXx(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-
<b>Function name</b>	can_transmission_stop
<b>Function prototype</b>	void can_transmission_stop(uint32_t can_periph, uint8_t mailbox_number);
<b>Function descriptions</b>	stop CAN transmission
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<b>CANx(x=0, 1)</b>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>mailbox_number</b>	Mailbox number
<b>CAN_MAILBOXx</b>	CAN_MAILBOXx(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

can\_transmission\_stop (CAN0, CAN\_MAILBOX0);

## can\_message\_receive

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

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

<b>Function name</b>	can_message_receive
<b>Function prototype</b>	void can_message_receive(uint32_t can_periph, uint8_t fifo_number, can_receive_message_struct* receive_message);
<b>Function descriptions</b>	CAN receive message
<b>Precondition</b>	can_struct_para_init()
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Input parameter{in}</b>	
fifo_number	Fifo number
CAN_FIFOx	CAN_FIFOx(x=0,1)
<b>Input parameter{in}</b>	
receive_message	CAN message receive stuct, the structure members can refer to members of the structure <a href="#">Table 3-66. can_receive_message_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-
<b>Function name</b>	can_message_receive
<b>Function prototype</b>	void can_message_receive(uint32_t can_periph, uint8_t fifo_number, can_receive_message_struct* receive_message);
<b>Function descriptions</b>	CAN receive message
<b>Precondition</b>	-
<b>The called functions</b>	-

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

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

Example:

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

### can\_fifo\_release

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

**Table 3-81. 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)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<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>	
-	-
<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)	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 3-82. Function can\_receive\_message\_length\_get**

<b>Function name</b>	can_receive_message_length_get
<b>Function prototype</b>	uint8_t can_receive_message_length_get(uint32_t can_periph, uint8_t fifo_number);
<b>Function descriptions</b>	CAN receive message length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<b>CANx(x=0,1)</b>	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Input parameter{in}</b>	
<b>fifo_number</b>	Fifo number
<b>CAN_FIFOx</b>	CAN_FIFOx(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	0..3
<b>Function name</b>	can_receive_message_length_get
<b>Function prototype</b>	uint8_t can_receive_message_length_get(uint32_t can_periph, uint8_t fifo_number);
<b>Function descriptions</b>	CAN receive message length
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>can_periph</b>	CAN peripheral
<b>CANx(x=0,1)</b>	CAN peripheral selection
Input parameter{in}	
<b>fifo_number</b>	Fifo number
<b>CAN_FIFOx</b>	CAN_FIFOx(x=0,1)
Output parameter{out}	
-	-
Return value	
<b>uint8_t</b>	0..3

Example:

```
/* CAN0 FIFO0 receive message length */
uint8_t frame_number = 0;
frame_number = can_receive_message_length_get (CAN0, CAN_FIFO0);
```

### **can\_working\_mode\_set**

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

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

<b>Function name</b>	can_working_mode_set
<b>Function prototype</b>	ErrStatus can_working_mode_set(uint32_t can_periph, uint8_t working_mode);
<b>Function descriptions</b>	set CAN working mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>can_periph</b>	CAN peripheral
<b>CANx(x=0,1)</b>	CAN peripheral selection, the CAN1 only for GD32F30X_CL
Input parameter{in}	
<b>can_working_mode</b>	Mode select

<code>CAN_MODE_INITIALIZE</code>	Initialize mode
<code>CAN_MODE_NORMAL</code>	Normal mode
<code>CAN_MODE_SLEEP</code>	Sleep mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<code>ErrStatus</code>	SUCCESS / ERROR

Example:

```
/* set CAN0 working at initialize mode */
can_working_mode_set (CAN0, CAN_MODE_INITIALIZE);
```

### **can\_wakeup**

The description of can\_wakeup is shown as below:

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

<b>Function name</b>	can_wakeup
<b>Function prototype</b>	<code>ErrStatus can_wakeup(uint32_t can_periph);</code>
<b>Function descriptions</b>	wake up CAN
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>can_periph</code>	CAN peripheral
<code>CANx(x=0,1)</code>	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<code>ErrStatus</code>	SUCCESS / ERROR
<b>Function name</b>	can_wakeup
<b>Function prototype</b>	<code>ErrStatus can_wakeup(uint32_t can_periph);</code>

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

Example:

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

### can\_error\_get

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

**Table 3-85. Function can\_error\_get**

<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>	
can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>can_error_enum</b>	0..7

<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>	
can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
can_error_enum	0..7

Example:

```
/* get CAN0 error type */
can_error_get (CAN0);
```

#### can\_receive\_error\_number\_get

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

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

<b>Function name</b>	can_receive_error_number_get
<b>Function prototype</b>	uint8_t can_receive_error_number_get(uint32_t can_periph);
<b>Function descriptions</b>	get CAN receive error number
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Output parameter{out}</b>	
-	-

Return value	
<b>uint8_t</b>	0..255
<b>Function name</b>	can_receive_error_number_get
<b>Function prototype</b>	uint8_t can_receive_error_number_get(uint32_t can_periph);
<b>Function descriptions</b>	get CAN receive error number
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>can_periph</b>	CAN peripheral
CANx(x=0,1)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
Output parameter{out}	
-	-
Return value	
<b>uint8_t</b>	0..255

Example:

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

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

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

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

Return value	
<b>uint8_t</b>	0..255
<b>Function name</b>	can_transmit_error_number_get
<b>Function prototype</b>	uint8_t can_transmit_error_number_get(uint32_t can_periph);
<b>Function descriptions</b>	get CAN transmit error number
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>can_periph</b>	CAN peripheral
CANx(x=0,1)	CAN peripheral selection, the CAN1 only for GD32F30X_CL

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

Example:

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

### can\_interrupt\_enable

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

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

<b>Function name</b>	can_interrupt_enable
<b>Function prototype</b>	void can_interrupt_enable(uint32_t can_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable CAN interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	

<b>can_periph</b>	CAN peripheral
<b>CANx(x=0,1)</b>	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Input parameter{in}</b>	
<b>interrupt</b>	Interrupt type
<b>CAN_INT_TME</b>	transmit mailbox empty interrupt enable
<b>CAN_INT_RFNE0</b>	receive FIFO0 not empty interrupt enable
<b>CAN_INT_RFF0</b>	receive FIFO0 full interrupt enable
<b>CAN_INT_RFO0</b>	receive FIFO0 overfull interrupt enable
<b>CAN_INT_RFNE1</b>	receive FIFO1 not empty interrupt enable
<b>CAN_INT_RFF1</b>	receive FIFO1 full interrupt enable
<b>CAN_INT_RFO1</b>	receive FIFO1 overfull interrupt enable
<b>CAN_INT_WERR</b>	warning error interrupt enable
<b>CAN_INT_PERR</b>	passive error interrupt enable
<b>CAN_INT_BO</b>	bus-off interrupt enable
<b>CAN_INT_ERRN</b>	error number interrupt enable
<b>CAN_INT_ERR</b>	error interrupt enable
<b>CAN_INT_WU</b>	wakeup interrupt enable
<b>CAN_INT_SLPW</b>	sleep working interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-
<b>Function name</b>	can_interrupt_enable
<b>Function prototype</b>	void can_interrupt_enable(uint32_t can_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable CAN interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>can_periph</b>	CAN peripheral
<b>CANx(x=0,1)</b>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>interrupt</b>	Interrupt type
<b>CAN_INT_TME</b>	transmit mailbox empty interrupt enable
<b>CAN_INT_RFNE0</b>	receive FIFO0 not empty interrupt enable
<b>CAN_INT_RFF0</b>	receive FIFO0 full interrupt enable
<b>CAN_INT_RF00</b>	receive FIFO0 overfull interrupt enable
<b>CAN_INT_RFNE1</b>	receive FIFO1 not empty interrupt enable
<b>CAN_INT_RFF1</b>	receive FIFO1 full interrupt enable
<b>CAN_INT_RF01</b>	receive FIFO1 overfull interrupt enable
<b>CAN_INT_WERR</b>	warning error interrupt enable
<b>CAN_INT_PERR</b>	passive error interrupt enable
<b>CAN_INT_BO</b>	bus-off interrupt enable
<b>CAN_INT_ERRN</b>	error number interrupt enable
<b>CAN_INT_ERR</b>	error interrupt enable
<b>CAN_INT_WU</b>	wakeup interrupt enable
<b>CAN_INT_SLPW</b>	sleep working interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### **can\_interrupt\_disable**

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

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

<b>Function name</b>	can_interrupt_disable
<b>Function prototype</b>	void can_interrupt_disable(uint32_t can_periph, uint32_t interrupt);
<b>Function descriptions</b>	disable CAN interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Input parameter{in}</b>	
interrupt	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_RF00	receive FIFO0 overfull interrupt enable
CAN_INT_RFNE1	receive FIFO1 not empty interrupt enable
CAN_INT_RFF1	receive FIFO1 full interrupt enable
CAN_INT_RF01	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 disable */

can_interrupt_disable (CAN0, CAN_INT_TME);
```

### can\_flag\_get

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

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

<b>Function name</b>	can_flag_get
<b>Function prototype</b>	FlagStatus can_flag_get(uint32_t can_periph, can_flag_enum flag);
<b>Function descriptions</b>	get CAN flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Input parameter{in}</b>	
flag	CAN flags
CAN_FLAG_MTE2	mailbox 2 transmit error
CAN_FLAG_MTE1	mailbox 1 transmit error
CAN_FLAG_MTE0	mailbox 0 transmit error
CAN_FLAG_MTF2	mailbox 2 transmit finished
CAN_FLAG_MTF1	mailbox 1 transmit finished
CAN_FLAG_MTF0	mailbox 0 transmit finished
CAN_FLAG_RFO0	receive FIFO0 overfull
CAN_FLAG_RFF0	receive FIFO0 full
CAN_FLAG_RFO1	receive FIFO1 overfull
CAN_FLAG_RFF1	receive FIFO1 full
CAN_FLAG_BOERR	bus-off error
CAN_FLAG_PERR	passive error

<b>CAN_FLAG_WERR</b>	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 3-91. Function can\_flag\_clear**

<b>Function name</b>	can_flag_clear
<b>Function prototype</b>	void can_flag_clear(uint32_t can_periph, can_flag_enum flag);
<b>Function descriptions</b>	clear CAN flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
CANx(x=0,1)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Input parameter{in}</b>	
<b>flag</b>	CAN flags
<b>CAN_FLAG_MTE2</b>	mailbox 2 transmit error
<b>CAN_FLAG_MTE1</b>	mailbox 1 transmit error
<b>CAN_FLAG_MTE0</b>	mailbox 0 transmit error
<b>CAN_FLAG_MTF2</b>	mailbox 2 transmit finished
<b>CAN_FLAG_MTF1</b>	mailbox 1 transmit finished
<b>CAN_FLAG_MTF0</b>	mailbox 0 transmit finished
<b>CAN_FLAG_RFO0</b>	receive FIFO0 overfull

<i>CAN_FLAG_RFF0</i>	receive FIFO0 full
<i>CAN_FLAG_RFO1</i>	receive FIFO1 overfull
<i>CAN_FLAG_RFF1</i>	receive FIFO1 full
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear CAN0 mailbox 0 transmit error flag*/
can_flag_clear (CAN0, CAN_FLAG_MTE0);
```

### **can\_interrupt\_flag\_get**

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

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

<b>Function name</b>	can_interrupt_flag_get
<b>Function prototype</b>	FlagStatus can_interrupt_flag_get(uint32_t can_periph, can_interrupt_flag_enum flag);
<b>Function descriptions</b>	get CAN interrupt flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
CANx(x=0,1)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Input parameter{in}</b>	
<b>flag</b>	CAN interrupt flags
<i>CAN_INT_FLAG_SLPI_F</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

<code>CAN_INT_FLAG_MTF2</code>	mailbox 2 transmit finished interrupt flag
<code>CAN_INT_FLAG_MTF1</code>	mailbox 1 transmit finished interrupt flag
<code>CAN_INT_FLAG_MTF0</code>	mailbox 0 transmit finished interrupt flag
<code>CAN_INT_FLAG_RF00</code>	receive FIFO0 overfull interrupt flag
<code>CAN_INT_FLAG_RFF0</code>	receive FIFO0 full interrupt flag
<code>CAN_INT_FLAG_RF01</code>	receive FIFO1 overfull interrupt flag
<code>CAN_INT_FLAG_RFF1</code>	receive FIFO1 full interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

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

### **can\_interrupt\_flag\_clear**

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

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

<b>Function name</b>	can_interrupt_flag_clear
<b>Function prototype</b>	void can_interrupt_flag_clear(uint32_t can_periph, can_interrupt_flag_enum flag);
<b>Function descriptions</b>	clear CAN interrupt flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<code>CANx(x=0,1)</code>	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Input parameter{in}</b>	
<b>flag</b>	CAN interrupt flags

<code>CAN_INT_FLAG_SLPIF</code>	status change interrupt flag of sleep working mode entering
<code>CAN_INT_FLAG_WUIF</code>	status change interrupt flag of wakeup from sleep working mode
<code>CAN_INT_FLAG_ERRIF</code>	error interrupt flag
<code>CAN_INT_FLAG_MTF2</code>	mailbox 2 transmit finished interrupt flag
<code>CAN_INT_FLAG_MTF1</code>	mailbox 1 transmit finished interrupt flag
<code>CAN_INT_FLAG_MTF0</code>	mailbox 0 transmit finished interrupt flag
<code>CAN_INT_FLAG_RF00</code>	receive FIFO0 overfull interrupt flag
<code>CAN_INT_FLAG_RFF0</code>	receive FIFO0 full interrupt flag
<code>CAN_INT_FLAG_RF01</code>	receive FIFO1 overfull interrupt flag
<code>CAN_INT_FLAG_RFF1</code>	receive FIFO1 full interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## 3.5. CRC

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

### 3.5.1. Descriptions of Peripheral registers

CRC registers are listed in the table shown as below:

**Table 3-94. CRC Registers**

Registers	Descriptions
<code>CRC_DATA</code>	CRC data register

Registers	Descriptions
CRC_FDATA	CRC free data register
CRC_CTL	CRC control register

### 3.5.2. Descriptions of Peripheral functions

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

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

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

#### crc\_deinit

The description of crc\_deinit is shown as below:

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

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

-	-
---	---

Example:

```
/* reset crc */
```

```
crc_deinit();
```

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

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

**Table 3-97. Function **crc\_data\_register\_reset****

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

Example:

```
/* reset crc data register */
```

```
crc_data_register_reset();
```

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

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

**Table 3-98. Function **crc\_data\_register\_read****

<b>Function name</b>	crc_data_register_read
<b>Function prototype</b>	uint32_t crc_data_register_read(void);
<b>Function descriptions</b>	read the value of the data register
<b>Precondition</b>	-

<b>The called functions</b>	
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	32-bit value of the data register (0-0xFFFFFFFF)

Example:

```
/* read crc data register */

uint32_t crc_value = 0;

crc_value = crc_data_register_read();
```

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

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

**Table 3-99. Function crc\_free\_data\_register\_read**

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

Example:

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

uint8_t crc_value = 0;
```

```
crc_value = crc_free_data_register_read();
```

### **crc\_free\_data\_register\_write**

The description of `crc_free_data_register_write` is shown as below:

**Table 3-100. Function `crc_free_data_register_write`**

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

Example:

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

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

The description of `crc_single_data_calculate` is shown as below:

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

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

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 3-102. 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>	-
Input parameter{in}	
<b>array</b>	pointer to an array of 32 bit data words
Input parameter{in}	
<b>size</b>	size of the array
Output parameter{out}	
-	-
Return value	
<b>uint32_t</b>	32-bit CRC calculate value (0-0xFFFFFFFF)

Example:

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

---

```
#define BUFFER_SIZE    6

uint32_t valcrc = 0;

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

rcu_periph_clock_enable(RCU_CRC);

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

## 3.6. CTC

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

### 3.6.1. Descriptions of Peripheral registers

CTC registers are listed in the table shown as below:

**Table 3-103. CTC Registers**

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

### 3.6.2. Descriptions of Peripheral functions

CTC registers are listed in the table shown as below:

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

Function name	Function description
ctc_deinit	reset CTC clock trim controller
ctc_counter_enable	enable CTC trim counter
ctc_counter_disable	disable CTC trim counter
ctc_irc48m_trim_value_config	configure the IRC48M trim value

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

## ctc\_deinit

The description of ctc\_deinit is shown as below:

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

Function name	ctc_deinit
Function prototype	void ctc_deinit (void)
Function descriptions	Reset CTC peripheral
Precondition	-

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

Example:

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

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

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

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

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

Example:

```
/* enable CTC trim counter*/
ctc_counter_enable();
```

### **ctc\_counter\_disable**

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

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

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

Example:

```
/* disable CTC trim counter */
ctc_counter_disable();
```

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

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

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

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

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

Example:

```
/* IRC48M trim value configuration */

ctc_irc48m_trim_value_config (0x01);
```

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

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

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

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

Example:

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

ctc_software_refsource_pulse_generate ();
```

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

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

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

<b>Function name</b>	ctc_hardware_trim_mode_config
<b>Function prototype</b>	void ctc_hardware_trim_mode_config(uint32_t hardmode);

<b>Function descriptions</b>	configure hardware automatically trim mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>hardmode</b>	hardware automatically trim mode enable or disable
<b>CTC_HARDWARE_TRIM_MODE_ENABLE</b>	hardware automatically trim mode enable
<b>CTC_HARDWARE_TRIM_MODE_DISABLE</b>	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 3-111. Function ctc\_refsource\_polarity\_config**

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

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

Example:

```
/* set reference source polarity */
ctc_refresource_polarity_config (CTC_REFRESOURCE_POLARITY_RISING);
```

### **ctc\_refresource\_signal\_select**

The description of `ctc_refresource_signal_select` is shown as below:

**Table 3-112. Function `ctc_refresource_signal_select`**

<b>Function name</b>	<code>ctc_refresource_signal_select</code>
<b>Function prototype</b>	<code>void ctc_refresource_signal_select(uint32_t refs);</code>
<b>Function descriptions</b>	select reference signal source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>refs</code>	reference signal source
<code>CTC_REFRESOURCE_G PIO</code>	GPIO is selected
<code>CTC_REFRESOURCE_LX TAL</code>	LXTAL is selected
<code>CTC_REFRESOURCE_U SBSOF</code>	USBFS_SOF or USBD_SOF is selected
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

ctc\_refsource\_signal\_select (CTC\_REFRESOURCE\_LXTAL);

## ctc\_refsource\_prescaler\_config

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

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

<b>Function name</b>	ctc_refsource_prescaler_config
<b>Function prototype</b>	void ctc_refsource_prescaler_config(uint32_t prescaler);
<b>Function descriptions</b>	configure reference signal source prescaler
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>prescaler</b>	Prescaler factor
<i>CTC_REFRESOURCE_P_SC_OFF</i>	reference signal not divided
<i>CTC_REFRESOURCE_P_SC_DIV2</i>	reference signal divided by 2
<i>CTC_REFRESOURCE_P_SC_DIV4</i>	reference signal divided by 4
<i>CTC_REFRESOURCE_P_SC_DIV8</i>	reference signal divided by 8
<i>CTC_REFRESOURCE_P_SC_DIV16</i>	reference signal divided by 16
<i>CTC_REFRESOURCE_P_SC_DIV32</i>	reference signal divided by 32
<i>CTC_REFRESOURCE_P_SC_DIV64</i>	reference signal divided by 64
<i>CTC_REFRESOURCE_P_SC_DIV128</i>	reference signal divided by 128
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

ctc_refresource_prescaler_config(CTC_REFRESOURCE_PSC_DIV2);
```

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

The description of `ctc_clock_limit_value_config` is shown as below:

**Table 3-114. Function `ctc_clock_limit_value_config`**

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

Example:

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

ctc_clock_limit_value_config (0x1F);
```

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

The description of `ctc_counter_reload_value_config` is shown as below:

**Table 3-115. Function `ctc_counter_reload_value_config`**

<b>Function name</b>	ctc_counter_reload_value_config
<b>Function prototype</b>	void ctc_counter_reload_value_config(uint16_t reload_value);
<b>Function descriptions</b>	configure CTC counter reload value
<b>Precondition</b>	-
<b>The called functions</b>	-

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 3-116. Function `ctc_counter_capture_value_read`**

<b>Function name</b>	ctc_counter_capture_value_read
<b>Function prototype</b>	uint16_t ctc_counter_capture_value_read(void);
<b>Function descriptions</b>	read CTC counter capture value
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	读取计数器捕获值(0x0000 - 0xFFFF)

Example:

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

### **ctc\_counter\_direction\_read**

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

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

<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(向下计数) / RESET(向上计数)

Example:

```
/* read ctc counter direction */

FlagStatus ctc_direction = SET;

ctc_direction = ctc_counter_direction_read();
```

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

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

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

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

Output parameter{out}	
	-
Return value	
<b>uint16_t</b>	读取计数器重载值的16位数据 (0x0000 - 0xFFFF)

Example:

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

uint16_t ctc_reload_value = 0;

ctc_reload_value = ctc_counter_reload_value_read();
```

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

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

**Table 3-119. Function `ctc_irc48m_trim_value_read`**

<b>Function name</b>	ctc_irc48m_trim_value_read
<b>Function prototype</b>	uint8_t ctc_irc48m_trim_value_read(void);
<b>Function descriptions</b>	read the IRC48M trim value
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
<b>uint8_t</b>	6位IRC48M校准值 (0-63)

Example:

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

uint8_t ctc_trim_value = 0;

ctc_trim_value = ctc_irc48m_trim_value_read();
```

### **ctc\_interrupt\_enable**

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

**Table 3-120. Functionctc\_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 3-121. Functionctc\_interrupt\_disable**

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

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

Example:

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

### **ctc\_interrupt\_flag\_get**

The description of `ctc_interrupt_flag_get` is shown as below:

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

<b>Function name</b>	ctc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus ctc_interrupt_flag_get(uint32_t int_flag);
<b>Function descriptions</b>	get CTC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	CTC interrupt flag
<i>CTC_INT_FLAG_CKO_K</i>	clock trim OK interrupt
<i>CTC_INT_FLAG_CKWARN</i>	clock trim warning interrupt
<i>CTC_INT_FLAG_ERR</i>	error interrupt
<i>CTC_INT_FLAG_EREF</i>	expect reference interrupt
<i>CTC_INT_FLAG_CKE</i>	clock trim error bit interrupt

RR	
<i>CTC_INT_FLAG_REF MISS</i>	reference sync pulse miss interrupt
<i>CTC_INT_FLAG_TRIM ERR</i>	trim value error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get CTC interrupt flag status */
FlagStatus state = ctc_interrupt_flag_get (CTC_INT_FLAG_CKOK);
```

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

The description of `ctc_interrupt_flag_clear` is shown as below:

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

<b>Function name</b>	ctc_interrupt_flag_clear
<b>Function prototype</b>	void ctc_interrupt_flag_clear(uint32_t int_flag);
<b>Function descriptions</b>	clear CTC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	CTC interrupt flag
<i>CTC_INT_FLAG_CKO K</i>	clock trim OK interrupt
<i>CTC_INT_FLAG_CKW ARN</i>	clock trim warning interrupt
<i>CTC_INT_FLAG_ERR</i>	error interrupt
<i>CTC_INT_FLAG_EREF</i>	expect reference interrupt
<i>CTC_INT_FLAG_CKE RR</i>	clock trim error bit interrupt

<i>CTC_INT_FLAG_REF MISS</i>	reference sync pulse miss interrupt
<i>CTC_INT_FLAG_TRIM ERR</i>	trim value error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **ctc\_flag\_get**

The description of `ctc_flag_get` is shown as below:

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

<b>Function name</b>	ctc_flag_get
<b>Function prototype</b>	FlagStatus ctc_flag_get(uint32_t flag);
<b>Function descriptions</b>	get CTC status flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	CTC status flag
<i>CTC_FLAG_CKOK</i>	clock trim OK interrupt flag
<i>CTC_FLAG_CKWARN</i>	clock trim warning interrupt flag
<i>CTC_FLAG_ERR</i>	error interrupt flag
<i>CTC_FLAG_EREF</i>	expect reference interrupt flag
<i>CTC_FLAG_CKERR</i>	clock trim error bit
<i>CTC_FLAG_REFMISS</i>	reference sync pulse miss flag
<i>CTC_FLAG_TRIMERR</i>	trim value error flag
<b>Output parameter{out}</b>	

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

Example:

```
/* get CTC flag status */
FlagStatus state = ctc_flag_get (CTC_FLAG_CKOK);
```

### ctc\_flag\_clear

The description of `ctc_flag_clear` is shown as below:

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

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

Example:

```
/* clear CTC flag status */  
ctc_flag_clear (CTC_FLAG_CKOK);
```

## 3.7. DAC

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

### 3.7.1. Descriptions of Peripheral registers

DAC registers are listed in the table shown as below:

**Table 3-126. DAC Registers**

Registers	Descriptions
DAC_CTL	DAC control register
DAC_SWT	DAC software trigger register
DAC0_R12DH	DAC0 12-bit right-aligned data holding register
DAC0_L12DH	DAC0 12-bit left-aligned data holding register
DAC0_R8DH	DAC0 8-bit right-aligned data holding register
DAC1_R12DH	DAC1 12-bit right-aligned data holding register
DAC1_L12DH	DAC1 12-bit left-aligned data holding register
DAC1_R8DH	DAC1 8-bit right-aligned data holding register
DACC_R12DH	DAC concurrent mode 12-bit right-aligned data holding register
DACC_L12DH	DAC concurrent mode 12-bit left-aligned data holding register
DACC_R8DH	DAC concurrent mode 8-bit right-aligned data holding register
DAC0_DO	DAC0 data output register
DAC1_DO	DAC1 data output register

### 3.7.2. Descriptions of Peripheral functions

DAC registers are listed in the table shown as below:

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

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

### dac\_deinit

The description of dac\_deinit is shown as below:

**Table 3-128. Function dac\_deinit**

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

Example:

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

### dac\_enable

The description of dac\_enable is shown as below:

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

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

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC0 */
dac_enable(DAC0);
```

### **dac\_disable**

The description of dac\_disable is shown as below:

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

Function name	dac_disable
Function prototype	void dac_disable(uint32_t dac_periph);
Function descriptions	disable DAC
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
DACx	peripheral selection(x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC0 */
dac_disable(DAC0);
```

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

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

**Table 3-131. Function dac\_dma\_enable**

<b>Function name</b>	dac_dma_enable
<b>Function prototype</b>	void dac_dma_enable(uint32_t dac_periph);
<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>	peripheral
<i>DACx</i>	peripheral selection (x =0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DAC0 DMA function */

dac_dma_enable(DAC0);
```

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

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

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

<b>Function name</b>	dac_dma_disable
<b>Function prototype</b>	void dac_dma_disable(uint32_t dac_periph);
<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>	peripheral
<i>DACx</i>	peripheral selection (x =0,1)
<b>Output parameter{out}</b>	

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

Example:

```
/* disable DAC0 DMA function */

dac_dma_disable(DAC0);
```

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

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

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

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

Example:

```
/* enable DAC0 output buffer */

dac_output_buffer_enable(DAC0);
```

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

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

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

<b>Function name</b>	dac_output_buffer_disable
----------------------	---------------------------

<b>Function prototype</b>	void dac_output_buffer_disable(uint32_t dac_periph);
<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>	peripheral
<b>DACx</b>	peripheral selection (x =0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DAC0 output buffer */

dac_output_buffer_disable(DAC0);
```

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

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

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

<b>Function name</b>	dac_output_value_get
<b>Function prototype</b>	uint16_t dac_output_value_get(uint32_t dac_periph);
<b>Function descriptions</b>	get DAC output value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	peripheral
<b>DACx</b>	peripheral selection (x =0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

uint16_t	DAC output data (0x0000 – 0x07FF)-
----------	------------------------------------

Example:

```
/* get DAC0 output value */

data = dac_output_value_get(DAC0);
```

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

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

**Table 3-136. Function **dac\_data\_set****

<b>Function name</b>	dac_data_set
<b>Function prototype</b>	void dac_data_set(uint32_t dac_periph, uint32_t dac_align, uint16_t data);
<b>Function descriptions</b>	set the DAC specified data holding register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	peripheral
<b>DACx</b>	peripheral selection(x =0,1)
<b>Input parameter{in}</b>	
<b>dac_align</b>	DAC align mode
<b>DAC_ALIGN_8B_R</b>	data right 8 bit alignment
<b>DAC_ALIGN_12B_R</b>	data right 12 bit alignment
<b>DAC_ALIGN_12B_L</b>	data left 12 bit alignment
<b>Input parameter{in}</b>	
<b>data</b>	The data sending to holding register
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the DAC0 specified data holding register value */
```

```
dac_data_set(DAC0, DAC_ALIGN_8B_R, 0xff);
```

## **dac\_trigger\_enable**

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

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

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

Example:

```
/* enable DAC0 trigger */
dac_trigger_enable(DAC0);
```

## **dac\_trigger\_disable**

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

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

<b>Function name</b>	dac_trigger_disable
<b>Function prototype</b>	void dac_trigger_disable(uint32_t dac_periph);
<b>Function descriptions</b>	disable DAC trigger
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>dac_periph</b>	peripheral
<i>DACx</i>	peripheral selection (x =0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DAC0 trigger */
dac_trigger_disable(DAC0);
```

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

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

**Table 3-139. Function dac\_trigger\_source\_config**

<b>Function name</b>	dac_trigger_source_config
<b>Function prototype</b>	void dac_trigger_source_config(uint32_t dac_periph,uint32_t triggersource);
<b>Function descriptions</b>	set DAC trigger source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	peripheral
<i>DACx</i>	peripheral selection (x =0,1)
<b>Input parameter{in}</b>	
<b>triggersource</b>	external triggers of DAC
<i>DAC_TRIGGER_T1_TRGO</i>	TIMER1 TRGO
<i>DAC_TRIGGER_T2_TRGO</i>	TIMER2 TRGO (for GD32F30X_CL)
<i>DAC_TRIGGER_T3_TRGO</i>	TIMER3 TRGO
<i>DAC_TRIGGER_T4_</i>	TIMER4 TRGO

<i>TRGO</i>	
<i>DAC_TRIGGER_T5_TRGO</i>	TIMER5 TRGO
<i>DAC_TRIGGER_T6_TRGO</i>	TIMER6 TRGO
<i>DAC_TRIGGER_T7_TRGO</i>	TIMER7 TRGO (for GD32F30X_HD and GD32F30X_XD)
<i>DAC_TRIGGER EXTI_9</i>	EXTI interrupt line 9 event
<i>DAC_TRIGGER_SOFTWARE</i>	software trigger
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set DAC0 trigger source*/
dac_trigger_source_config(DAC0, DAC_TRIGGER_T1_TRGO);
```

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

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

**Table 3-140. Function **dac\_software\_trigger\_enable****

<b>Function name</b>	dac_software_trigger_enable
<b>Function prototype</b>	void dac_software_trigger_enable(uint32_t dac_periph);
<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>	peripheral
<b>DACx</b>	peripheral selection (x =0,1)
<b>Output parameter{out}</b>	

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

Example:

```
/* enable DAC0 software trigger */
dac_software_trigger_enable(DAC0);
```

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

The description of `dac_software_trigger_disable` is shown as below:

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

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

Example:

```
/* disable DAC0 software trigger */
dac_software_trigger_disable(DAC0);
```

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

The description of `dac_wave_mode_config` is shown as below:

**Table 3-142. Function `dac_wave_mode_config`**

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

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

Example:

```
/* configure DAC0 wave mode */
dac_wave_mode_config(DAC0, DAC_WAVE_DISABLE);
```

### **dac\_wave\_bit\_width\_config**

The description of `dac_wave_bit_width_config` is shown as below:

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

<b>Function name</b>	dac_wave_bit_width_config
<b>Function prototype</b>	void dac_wave_bit_width_config(uint32_t dac_periph, uint32_t bit_width);
<b>Function descriptions</b>	configure DAC wave bit width
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>dac_periph</b>	peripheral
<i>DACx</i>	peripheral selection (x =0,1)
Input parameter{in}	
<b>bit_width</b>	noise wave bit width
<i>DAC_WAVE_BIT_WIDTH_x</i>	noise wave bit width (x = 1..12)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DAC0 wave bit width */

dac_wave_bit_width_config(DAC0, DAC_WAVE_BIT_WIDTH_1);
```

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

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

**Table 3-144. Function dac\_lfsr\_noise\_config**

<b>Function name</b>	dac_lfsr_noise_config
<b>Function prototype</b>	void dac_lfsr_noise_config(uint32_t dac_periph, uint32_t unmask_bits);
<b>Function descriptions</b>	configure DAC LFSR noise mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>dac_periph</b>	peripheral
<i>DACx</i>	peripheral selection (x =0,1)
Input parameter{in}	
<b>unmask_bits</b>	noise wave unmask bit width
<i>DAC_LFSR_BIT0</i>	unmask the LFSR bit 0

<i>DAC_LFSR_BITSx_0</i>	unmask the LFSR bits[x:0]
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DAC0 LFSR noise mode */
dac_lfsr_noise_config(DAC0, DAC_LFSR_BIT0);
```

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

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

**Table 3-145. Function *dac\_triangle\_noise\_config***

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

Example:

---

```
/* configure DAC0 triangle noise mode */

dac_triangle_noise_config(DAC0, DAC_TRIANGLE_AMPLITUDE_1);
```

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

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

**Table 3-146. Function dac\_concurrent\_enable**

<b>Function name</b>	dac_concurrent_enable
<b>Function prototype</b>	void dac_concurrent_enable (void);
<b>Function descriptions</b>	enable DAC concurrent 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 DAC concurrent mode */

dac_concurrent_enable();
```

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

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

**Table 3-147. Function dac\_concurrent\_disable**

<b>Function name</b>	dac_concurrent_disable
<b>Function prototype</b>	void dac_concurrent_disable (void);
<b>Function descriptions</b>	disable DAC concurrent 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 DAC concurrent mode */

dac_concurrent_disable();
```

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

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

**Table 3-148. Function dac\_concurrent\_software\_trigger\_enable**

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

Example:

```
/* enable DAC concurrent software trigger function */

dac_concurrent_software_trigger_enable();
```

### **dac\_concurrent\_software\_trigger\_disable**

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

**Table 3-149. Function dac\_concurrent\_software\_trigger\_disable**

<b>Function name</b>	dac_concurrent_software_trigger_disable
<b>Function prototype</b>	void dac_concurrent_software_trigger_disable(void);
<b>Function descriptions</b>	disable DAC concurrent software trigger function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DAC concurrent software trigger function */

dac_concurrent_software_trigger_disable();
```

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

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

**Table 3-150. Function dac\_concurrent\_output\_buffer\_enable**

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

-	-
---	---

Example:

```
/* enable DAC concurrent buffer function */
```

```
dac_concurrent_output_buffer_enable();
```

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

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

**Table 3-151. Function dac\_concurrent\_output\_buffer\_disable**

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

Example:

```
/* disable DAC concurrent buffer function */
```

```
dac_concurrent_output_buffer_disable();
```

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

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

**Table 3-152. Function dac\_concurrent\_data\_set**

<b>Function name</b>	dac_concurrent_data_set
<b>Function prototype</b>	void dac_concurrent_data_set(uint32_t dac_align, uint16_t data0, uint16_t data1);
<b>Function descriptions</b>	set DAC concurrent mode data holding register value

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_align</b>	DAC align mode
<i>DAC_ALIGN_8B_R</i>	data right 8 bit alignment
<i>DAC_ALIGN_12B_R</i>	data right 12 bit alignment
<i>DAC_ALIGN_12B_L</i>	data left 12 bit alignment
<b>Input parameter{in}</b>	
<b>data0</b>	The data sending to holding register of DAC0
<b>Input parameter{in}</b>	
<b>data1</b>	The data sending to holding register of DAC1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set DAC concurrent mode data holding register value */
dac_concurrent_data_set(DAC_ALIGN_8B_R, 0xff, 0xff);
```

## 3.8. DBG

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

### 3.8.1. Descriptions of Peripheral registers

DBG registers are listed in the table shown as below:

**Table 3-153. DBG Registers**

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

### 3.8.2. Descriptions of Peripheral functions

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

**Table 3-154. DBG firmware function**

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

#### Enum dbg\_periph\_enum

**Table 3-155. Enum dbg\_periph\_enum**

Member name	Function description
DBG_FWDGT_HOLD	debug FWDGT kept when core is halted
DBG_WWDGT_HOLD	debug WWDGT kept when core is halted
DBG_TIMER0_HOLD	hold TIMER0 counter when core is halted
DBG_TIMER1_HOLD	hold TIMER1 counter when core is halted
DBG_TIMER2_HOLD	hold TIMER2 counter when core is halted
DBG_TIMER3_HOLD	hold TIMER3 counter when core is halted
DBG_CAN0_HOLD	debug CAN0 kept when core is halted
DBG_I2C0_HOLD	hold I2C0 smbus when core is halted
DBG_I2C1_HOLD	hold I2C1 smbus when core is halted
DBG_TIMER4_HOLD	hold TIMER4 counter when core is halted
DBG_TIMER5_HOLD	hold TIMER5 counter when core is halted
DBG_TIMER6_HOLD	hold TIMER6 counter when core is halted

DBG_TIMER7_HOLD	hold TIMER7 counter when core is halted
DBG_CAN1_HOLD	debug CAN1 kept when core is halted
DBG_TIMER11_HOLD	hold TIMER11 counter when core is halted
DBG_TIMER12_HOLD	hold TIMER12 counter when core is halted
DBG_TIMER13_HOLD	hold TIMER13 counter when core is halted
DBG_TIMER8_HOLD	hold TIMER8 counter when core is halted
DBG_TIMER9_HOLD	hold TIMER9 counter when core is halted
DBG_TIMER10_HOLD	hold TIMER10 counter when core is halted

## dbg\_deinit

The description of dbg\_deinit is shown as below:

**Table 3-156. Function dbg\_deinit**

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

Example:

```
/* deinitialize the DBG */

dbg_deinit();
```

## dbg\_id\_get

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

**Table 3-157. Function dbg\_id\_get**

<b>Function name</b>	dbg_id_get
<b>Function prototype</b>	uint32_t dbg_id_get(void);
<b>Function descriptions</b>	Read DBG_ID code register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
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 3-158. Function dbg\_low\_power\_enable**

<b>Function name</b>	dbg_low_power_enable
<b>Function prototype</b>	void dbg_low_power_enable(uint32_t dbg_low_power);
<b>Function descriptions</b>	Enable low power behavior when the mcu is in debug mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dbg_low_power</b>	low power mode
<b>DBG_LOW_POWER_SLEEP</b>	keep debugger connection during sleep mode
<b>DBG_LOW_POWER_D</b>	keep debugger connection during deepsleep mode

<i>EEPSLEEP</i>	
<i>DBG_LOW_POWER_S TANDBY</i>	keep debugger connection during standby mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

dbg_low_power_enable(DBG_LOW_POWER_SLEEP);
```

### **dbg\_low\_power\_disable**

The description of `dbg_low_power_disable` is shown as below:

**Table 3-159. Function `dbg_low_power_disable`**

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

Example:

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

```
dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

### **dbg\_periph\_enable**

The description of `dbg_periph_enable` is shown as below:

**Table 3-160. Function `dbg_periph_enable`**

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

Example:

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

```
dbg_periph_enable(DBG_TIMER0_HOLD);
```

### **dbg\_periph\_disable**

The description of `dbg_periph_disable` is shown as below:

**Table 3-161. Function dbg\_periph\_disable**

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

Example:

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

dbg_periph_disable(DBG_TIMER0_HOLD);
```

### **dbg\_trace\_pin\_enable**

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

**Table 3-162. 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>	-

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

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 3-163. Function dbg\_trace\_pin\_disable**

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

Example:

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

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

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

Table 3-164. Function `dbg_trace_pin_mode_set`

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

Example:

```
/* trace pin mode selection */

dbg_trace_pin_mode_set(TRACE_MODE_ASYNC);
```

### 3.9. DMA

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

### 3.9.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

**Table 3-165. 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

### 3.9.2. Descriptions of Peripheral functions

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

**Table 3-166. DMA firmware function**

Function name	Function description
dma_deinit	deinitialize DMA a channel registers
dma_struct_para_init	initialize the parameters of DMA struct with the default values
dma_init	initialize DMA channel
dma_circulation_enable	enable DMA circulation mode
dma_circulation_disable	disable DMA circulation mode
dma_memory_to_memory_enable	enable memory to memory mode
dma_memory_to_memory_disable	disable memory to memory mode
dma_channel_enable	enable DMA channel
dma_channel_disable	disable DMA channel
dma_periph_address_config	set DMA peripheral base address
dma_memory_address_config	set DMA memory base address

Function name	Function description
dma_transfer_number_config	set the number of remaining data to be transferred by the DMA
dma_transfer_number_get	get the number of remaining data to be transferred by the DMA
dma_priority_config	configure priority level of DMA channel
dma_memory_width_config	configure transfer data size of memory
dma_periph_width_config	configure transfer data size of peripheral
dma_memory_increase_enable	enable next address increasement algorithm of memory
dma_memory_increase_disable	disable next address increasement algorithm of memory
dma_periph_increase_enable	enable next address increasement algorithm of peripheral
dma_periph_increase_disable	disable next address increasement algorithm of peripheral
dma_transfer_direction_config	configure the direction of data transfer on the channel
dma_flag_get	check DMA flag is set or not
dma_flag_clear	clear the flag of a DMA channel
dma_interrupt_flag_get	check DMA flag and interrupt enable bit is set or not
dma_interrupt_flag_clear	clear the interrupt flag of a DMA channel
dma_interrupt_enable	enable DMA interrupt
dma_interrupt_disable	disable DMA interrupt

## Structure dma\_parameter\_struct

**Table 3-167. 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 3-168. Function dma\_deinit**

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

Example:

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

### dma\_struct\_para\_init

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

**Table 3-169. Function dma\_struct\_para\_init**

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

Example:

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

## dma\_init

The description of dma\_init is shown as below:

**Table 3-170. 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>	
dma_periph	DMA peripheral
DMAx(x=0, 1)	DMA peripheral selection
<b>Input parameter{in}</b>	
channelx	DMA channel
DMA_CHx(DMA0:x=..)	DMA channel selection

6; DMA1: x=0..4)	
<b>Input parameter{in}</b>	
init_struct	Structure for initialization, the structure members can refer to <a href="#">Table 3-167.</a> <a href="#"><u>Structure dma_parameter_struct</u></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_8BIT;
dma_init_struct.number = TRANSFER_NUM;
dma_init_struct.periph_addr = (uint32_t)BANK0_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_8BIT;
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_init(DMA0, DMA_CH0, &dma_init_struct);

```

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

The description of `dma_circulation_enable` is shown as below:

**Table 3-171. Function `dma_circulation_enable`**

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

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

Example:

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

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

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

**Table 3-172. 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

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

The description of `dma_memory_to_memory_enable` is shown as below:

**Table 3-173. 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>	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<b>DMAx(x=0,1)</b>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<b>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</b>	DMA channel selection
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 3-174. Function `dma_memory_to_memory_disable`**

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

Example:

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

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

The description of `dma_channel_enable` is shown as below:

**Table 3-175. Function `dma_channel_enable`**

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

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

Example:

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

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

The description of `dma_channel_disable` is shown as below:

**Table 3-176. 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
<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 3-177. Function `dma_periph_address_config`**

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

-	-
Return value	
-	-

Example:

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

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

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

The description of `dma_memory_address_config` is shown as below:

**Table 3-178. Function `dma_memory_address_config`**

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

Example:

```

uint8_t g_destbuf[TRANSFER_NUM];

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

```

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

The description of `dma_transfer_number_config` is shown as below:

**Table 3-179. Function `dma_transfer_number_config`**

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

Example:

```

#define TRANSFER_NUM          0x400

dma_transfer_number_config(DMA0, DMA_CH0, TRANSFER_NUM);

```

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

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

**Table 3-180. 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
DMAx(x=0, 1)	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	0-0xffff

Example:

```
uint32_t number = 0;  
  
number = dma_transfer_number_get(DMA0, DMA_CH0);
```

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

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

**Table 3-181. Function dma\_priority\_config**

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

<b>Function descriptions</b>	configure priority level of DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0..1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>priority</b>	priority Level of this channel
<i>DMA_PRIORITY_LOW</i>	low priority
<i>DMA_PRIORITY_MEDIUM</i>	medium priority
<i>DMA_PRIORITY_HIGH</i>	high priority
<i>DMA_PRIORITY_ULTRA_HIGH</i>	ultra high priority
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

The description of `dma_memory_width_config` is shown as below:

**Table 3-182. Function `dma_memory_width_config`**

<b>Function name</b>	dma_memory_width_config
<b>Function prototype</b>	void dma_memory_width_config (uint32_t dma_periph, dma_channel_enum

	channelx, uint32_t mwidth);
<b>Function descriptions</b>	configure transfer data size of memory
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>mwidth</b>	transfer data width of memory
<i>DMA_MEMORY_WI_DTH_8BIT</i>	transfer data width of memory is 8-bit
<i>DMA_MEMORY_WI_DTH_16BIT</i>	transfer data width of memory is 16-bit
<i>DMA_MEMORY_WI_DTH_32BIT</i>	transfer data width of memory is 32-bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
dma_memory_width_config(DMA0, DMA_CH0, DMA_MEMORY_WIDTH_8BIT);
```

### **dma\_periph\_width\_config**

The description of `dma_periph_width_config` is shown as below:

**Table 3-183. 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
<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 3-184. 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
<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 3-185. 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

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

Example:

```
dma_memory_increase_disable(DMA0, DMA_CH0);
```

### **dma\_periph\_increase\_enable**

The description of `dma_periph_increase_enable` is shown as below:

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

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

-	-
---	---

Example:

```
dma_periph_increase_enable(DMA0, DMA_CH0);
```

### **dma\_periph\_increase\_disable**

The description of dma\_periph\_increase\_disable is shown as below:

**Table 3-187. Function dma\_periph\_increase\_disable**

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

Example:

```
dma_periph_increase_disable(DMA0, DMA_CH0);
```

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

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

**Table 3-188. Function dma\_transfer\_direction\_config**

<b>Function name</b>	dma_transfer_direction_config
<b>Function prototype</b>	void dma_transfer_direction_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t direction);
<b>Function descriptions</b>	configure the direction of data transfer on the channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>direction</b>	specify the direction of data transfer
<i>DMA_PERIPHERAL_TO_MEMORY</i>	read from peripheral and write to memory
<i>DMA_MEMORY_TO_PERIPHERAL</i>	read from memory and write to peripheral
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **dma\_flag\_get**

The description of `dma_flag_get` is shown as below:

**Table 3-189. Function dma\_flag\_get**

<b>Function name</b>	dma_flag_get
----------------------	--------------

<b>Function prototype</b>	FlagStatus dma_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
<b>Function descriptions</b>	check DMA flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0..1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>DMA_FLAG_G</i>	global interrupt flag of channel
<i>DMA_FLAG_FTF</i>	full transfer finish flag of channel
<i>DMA_FLAGHTF</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 3-190. Function `dma_flag_clear`**

<b>Function name</b>	dma_flag_clear
----------------------	----------------

<b>Function prototype</b>	void dma_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
<b>Function descriptions</b>	clear the flag of a DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0..1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>DMA_FLAG_G</i>	global interrupt flag of channel
<i>DMA_FLAG_FTF</i>	full transfer finish flag of channel
<i>DMA_FLAG_HTF</i>	half transfer finish flag of channel
<i>DMA_FLAG_ERR</i>	error flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
dma_flag_clear(DMA0, DMA_CH0, DMA_FLAG_FTF);
```

### **dma\_interrupt\_flag\_get**

The description of `dma_interrupt_flag_get` is shown as below:

**Table 3-191. 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>	
dma_periph	DMA peripheral
DMAx(x=0, 1)	DMA peripheral selection
<b>Input parameter{in}</b>	
channelx	DMA channel
DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)	DMA channel selection
<b>Input parameter{in}</b>	
flag	specify get which flag
DMA_INT_FLAG_FTF	full transfer finish interrupt flag of channel
DMA_INT_FLAG_HTF	half transfer finish interrupt flag of channel
DMA_INT_FLAG_ER	error interrupt flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
FlagStatus	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 3-192. Function `dma_interrupt_flag_clear`

<b>Function name</b>	dma_interrupt_flag_clear
<b>Function prototype</b>	<code>void dma_interrupt_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);</code>
<b>Function descriptions</b>	clear the interrupt flag of a DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>dma_periph</code>	DMA peripheral
<code>DMAx(x=0, 1)</code>	DMA peripheral selection
<b>Input parameter{in}</b>	
<code>channelx</code>	DMA channel
<code>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</code>	DMA channel selection
<b>Input parameter{in}</b>	
<code>flag</code>	specify get which flag
<code>DMA_INT_FLAG_G</code>	global interrupt flag of channel
<code>DMA_INT_FLAG_FT F</code>	full transfer finish interrupt flag of channel
<code>DMA_INT_FLAG_HTF</code>	half transfer finish interrupt flag of channel
<code>DMA_INT_FLAG_ER R</code>	error interrupt flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
if(dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_G);
}
```

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

The description of `dma_interrupt_enable` is shown as below:

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

<b>Function name</b>	dma_interrupt_enable
<b>Function prototype</b>	<code>void dma_interrupt_enable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);</code>
<b>Function descriptions</b>	enable DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>source</b>	DMA interrupt source
<i>DMA_INT_FTF</i>	full transfer finish interrupt of channel
<i>DMA_INT_HTF</i>	half transfer finish interrupt of channel
<i>DMA_INT_ERR</i>	error interrupt of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* DMA0 channel0 interrupt configuration */
dma_interrupt_enable(DMA0, DMA_CH0, DMA_INT_FTF);
```

### **dma\_interrupt\_disable**

The description of `dma_interrupt_disable` is shown as below:

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

<b>Function name</b>	dma_interrupt_disable
<b>Function prototype</b>	<code>void dma_interrupt_disable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);</code>
<b>Function descriptions</b>	disable DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>source</b>	DMA interrupt source
<i>DMA_INT_FTF</i>	full transfer finish interrupt of channel
<i>DMA_INT_HTF</i>	half transfer finish interrupt of channel
<i>DMA_INT_ERR</i>	error interrupt of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* DMA0 channel0 interrupt configuration */

dma_interrupt_disable(DMA0, DMA_CH0, DMA_INT_FTF);
```

## 3.10. ENET

There is a media access controller (MAC) designed in Ethernet module to support 10/100Mbps interface speed. For more efficient data transfer between Ethernet and memory, a DMA controller is designed in this module. The support interface protocol for Ethernet is media independent interface (MII) and reduced media independent interface (RMII). The ENET registers are listed in chapter [3.10.1](#), the ENET firmware functions are introduced in chapter [3.10.2](#).

### 3.10.1. Descriptions of Peripheral registers

ENET registers are listed in the table shown as below:

**Table 3-195. ENET Registers**

Registers	Descriptions
ENET_MAC_CFG	MAC configuration register
ENET_MAC_FRMF	MAC frame filter register
ENET_MAC_HLH	MAC hash list high register
ENET_MAC_HLL	MAC hash list low register
ENET_MAC_PHY_CTL	MAC PHY control register
ENET_MAC_PHY_DATA	MAC MII 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_ADDR_0H	MAC address 0 high register
ENET_MAC_ADDR_0L	MAC address 0 low register

Registers	Descriptions
ENET_MAC_ADDR 1H	MAC address 1 high register
ENET_MAC_ADDR 1L	MAC address 1 low register
ENET_MAC_ADDT 2H	MAC address 2 high register
ENET_MAC_ADDR 2L	MAC address 2 low register
ENET_MAC_ADDR 3H	MAC address 3 high register
ENET_MAC_ADDR 3L	MAC address 3 low register
ENET_MAC_FCTH	MAC flow control threshold register
ENET_MSC_CTL	MSC control register
ENET_MSC_RINTF	MSC receive interrupt flag register
ENET_MSC_TINTF	MSC transmit interrupt flag register
ENET_MSC_RINT MSK	MSC receive interrupt mask register
ENET_MSC_TINTM SK	MSC transmit interrupt mask register
ENET_MSC_SCCN T	MSC transmitted good frames after a single collision counter register
ENET_MSC_MSCC NT	MSC transmitted good frames after more than a single collision counter register
ENET_MSC_TGFC NT	MSC transmitted good frames counter register
ENET_MSC_RFCE CNT	MSC received frames with CRC error counter register
ENET_MSC_RFAE CNT	MSC received frames with alignment error counter register
ENET_MSC_RGUF CNT	MSC received good unicast frames counter register

Registers	Descriptions
ENET_PTP_TSCTL	PTP time stamp control register
ENET_PTP_SSINC	PTP subsecond increment register
ENET_PTP_TSH	PTP time stamp high register
ENET_PTP_TSL	PTP time stamp low register
ENET_PTP_TSUH	PTP time stamp update high register
ENET_PTP_TSUL	PTP time stamp update low register
ENET_PTP_TSADD END	PTP time stamp addend register
ENET_PTP_ETH	PTP expected time high register
ENET_PTP_ETL	PTP expected time low register
ENET_PTP_TSFLAG	PTP time stamp flag register
ENET_PTP_PPSCTL	PTP PPS control register
ENET_DMA_BCTL	DMA bus control register
ENET_DMA_TPEN	DMA transmit poll enable register
ENET_DMA_RPEN	DMA receive poll enable register
ENET_DMA_RDTA DDR	DMA receive descriptor table address register
ENET_DMA_TDТА 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_MFBOCNT	DMA missed frame and buffer overflow counter register
ENET_DMA_RSWD	DMA receive state watchdog counter register
ENET_DMA_CTDA DDR	DMA current transmit descriptor address register
ENET_DMA_CRDA	DMA current receive descriptor address register

Registers	Descriptions
DDR	
ENET_DMA_CTBA DDR	DMA current transmit buffer address register
ENET_DMA_CRBA DDR	DMA current receive buffer address register

### 3.10.2. Descriptions of Peripheral functions

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

**Table 3-196. 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

Function name	Function description
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
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

Function name	Function description
	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 be 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

Function name	Function description
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 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 ptpt flag status
internal function	
enet_initpara_reset	reset the ENET initpara struct, call it before using enet_initpara_config()

## Structure enet\_initpara\_struct

Table 3-197. 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 3-198. 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 3-199. Structure enet\_ptp\_systime\_struct**

member name	Function description
second	second of system time
nanosecond	nanosecond of system time
sign	sign of system time

### enet\_deinit

The description of enet\_deinit is shown as below:

**Table 3-200. Function enet\_deinit**

<b>Function name</b>	enet_deinit
<b>Function prototype</b>	void enet_deinit(void);
<b>Function descriptions</b>	deinitialize the ENET, and reset structure parameters for ENET initialization
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable() /rcu_periph_reset_disable()/enet_initpara_reset()
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* deinitialize the ENET */
enet_deinit( );
```

### enet\_initpara\_config

The description of enet\_initpara\_config is shown as below:

**Table 3-201. Function enet\_initpara\_config**

<b>Function name</b>	enet_initpara_config
<b>Function prototype</b>	void enet_initpara_config(enet_option_enum option, uint32_t para)

<b>Function descriptions</b>	configure the parameters which are usually less cared for initialization, this function must be called before enet_init(), otherwise configuration will be no effect
<b>Precondition</b>	enet_initpara_reset(void)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>option</b>	different function option, which is related to several parameters only one parameter can be selected which is shown as below
<i>FORWARD_OPTION</i>	choose to configure the frame forward related parameters
<i>DMABUS_OPTION</i>	choose to configure the DMA bus mode related parameters
<i>DMA_MAXBURST_OPTION</i>	choose to configure the DMA max burst related parameters
<i>DMA_ARBITRATION_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>HASL_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
<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 <i>FORWARD_OPTION</i>	

value1	<i>ENET_AUTO_PADCRC_DROP_ENABLE / ENET_AUTO_PADCRC_DROP_DISABLE</i>
value2	<i>ENET_FORWARD_ERRFRAMES_ENABLE / ENET_FORWARD_ERRFRAMES_DISABLE</i>
value3	<i>ENET_FORWARD_UNDERSZ_GOODFRAMES_ENABLE / ENET_FORWARD_UNDERSZ_GOODFRAMES_DISABLE</i>
value4	<i>ENET_FORWARD_UNDERSZ_GOODFRAMES_ENABLE / ENET_FORWARD_UNDERSZ_GOODFRAMES_DISABLE</i>
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</i>

	<i>ENET_RX_THRESHOLD_96BYTES / ENET_RX_THRESHOLD_128BYTES</i>
value4	<i>ENET_TX_THRESHOLD_64BYTES / ENET_TX_THRESHOLD_128BYTES / ENET_TX_THRESHOLD_192BYTES / ENET_TX_THRESHOLD_256BYTES / ENET_TX_THRESHOLD_40BYTES / ENET_TX_THRESHOLD_32BYTES / ENET_TX_THRESHOLD_24BYTES / ENET_TX_THRESHOLD_16BYTES</i>
When value of parameter <b>option</b> is <i>DMA_OPTION</i>	
value1	<i>ENET_FLUSH_RXFRAME_ENABLE / ENET_FLUSH_RXFRAME_DISABLE</i>
value2	<i>ENET_SECONDFRAME_OPT_ENABLE / ENET_SECONDFRAME_OPT_DISABLE</i>
value3	<i>ENET_ENHANCED_DESCRIPTOR / ENET_NORMAL_DESCRIPTOR</i>
When value of parameter <b>option</b> is <i>VLAN_OPTION</i>	
value1	<i>ENET_VLANTAGCOMPARISON_12BIT / ENET_VLANTAGCOMPARISON_16BIT</i>
value2	<i>MAC_VLT_VLTI(regval)</i>
When value of parameter <b>option</b> is <i>FLOWCTL_OPTION</i>	
value1	<i>MAC_FCTL_PTM(regval)</i>
value2	<i>ENET_ZERO_QUANTA_PAUSE_ENABLE / ENET_ZERO_QUANTA_PAUSE_DISABLE</i>
value3	<i>ENET_PAUSETIME_MINUS4 / ENET_PAUSETIME_MINUS28 / ENET_PAUSETIME_MINUS144 / ENET_PAUSETIME_MINUS256</i>
value4	<i>ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDTECT / ENET_UNIQUE_PAUSEDTECT</i>
value5	<i>ENET_RX_FLOWCONTROL_ENABLE / ENET_RX_FLOWCONTROL_DISABLE</i>
value6	<i>ENET_TX_FLOWCONTROL_ENABLE / ENET_TX_FLOWCONTROL_DISABLE</i>
When value of parameter <b>option</b> is <i>HASHH_OPTION</i>	
value1	<i>0x0~0xFFFF FFFFU</i>
When value of parameter <b>option</b> is <i>HASHL_OPTION</i>	
value1	<i>0x0~0xFFFF FFFFU</i>

When value of parameter <b>option</b> is <i>FILTER_OPTION</i>	
value1	<i>ENET_SRC_FILTER_NORMAL_ENABLE / ENET_SRC_FILTER_INVERSE_ENABLE / ENET_SRC_FILTER_DISABLE</i>
value2	<i>ENET_DEST_FILTER_INVERSE_ENABLE / ENET_DEST_FILTER_INVERSE_DISABLE</i>
value3	<i>ENET_MULTICAST_FILTER_HASH_OR_PERFECT / ENET_MULTICAST_FILTER_HASH / ENET_MULTICAST_FILTER_PERFECT / ENET_MULTICAST_FILTER_NONE</i>
value4	<i>ENET_UNICAST_FILTER EITHER / ENET_UNICAST_FILTER_HASH / ENET_UNICAST_FILTER_PERFECT</i>
value5	<i>ENET_PCFRM_PREVENT_ALL / ENET_PCFRM_PREVENT_PAUSEFRAME / ENET_PCFRM_FORWARD_ALL / ENET_PCFRM_FORWARD_FILTERED</i>
When value of parameter <b>option</b> is <i>HALFDUPLEX_OPTION</i>	
value1	<i>ENET_CARRIERSENSE_ENABLE / ENET_CARRIERSENSE_DISABLE</i>
value2	<i>ENET_RECEIVEOWN_ENABLE / ENET_RECEIVEOWN_DISABLE</i>
value3	<i>ENET_RETRYTRANSMISSION_ENABLE / ENET_RETRYTRANSMISSION_DISABLE</i>
value4	<i>ENET_BACKOFFLIMIT_10 / ENET_BACKOFFLIMIT_8 / ENET_BACKOFFLIMIT_4 / ENET_BACKOFFLIMIT_1</i>
value5	<i>ENET_DEFERRALCHECK_ENABLE / ENET_DEFERRALCHECK_DISABLE</i>
When value of parameter <b>option</b> is <i>TIMER_OPTION</i>	
value1	<i>ENET_WATCHDOG_ENABLE / ENET_WATCHDOG_DISABLE</i>
value2	<i>ENET_JABBER_ENABLE / ENET_JABBER_DISABLE</i>
When value of parameter <b>option</b> is <i>INTERFRAMEGAP_OPTION</i>	
value1	<i>ENET_INTERFRAMEGAP_96BIT / ENET_INTERFRAMEGAP_88BIT / ENET_INTERFRAMEGAP_80BIT / ENET_INTERFRAMEGAP_72BIT / ENET_INTERFRAMEGAP_64BIT / ENET_INTERFRAMEGAP_56BIT / ENET_INTERFRAMEGAP_48BIT / ENET_INTERFRAMEGAP_40BIT</i>
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* config DMA option of the ENET */

enet_initpara_reset();

enet_initpara_config(DMA_OPTION, ENET_FLUSH_RXFRAME_ENABLE | ENET_SECO
NDFRAME_OPT_ENABLE | ENET_NORMAL_DESCRIPTOR);
```

### enet\_init

The description of enet\_init is shown as below:

**Table 3-202. Function enet\_init**

<b>Function name</b>	enet_init
<b>Function prototype</b>	ErrStatus enet_init(enet_mediemode_enum mediemode, enetc_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>mediemode</b>	PHY mode and mac loopback configurations, only one parameter can be selected
<i>ENET_AUTO_NEGOTIATION</i>	PHY auto negotiation
<i>ENET_100M_FULLDUPLEX</i>	100Mbit/s, full-duplex
<i>ENET_100M_HALFDUPLEX</i>	100Mbit/s, half-duplex
<i>ENET_10M_FULLDUPLEX</i>	10Mbit/s, full-duplex
<i>ENET_10M_HALFDUPLEX</i>	10Mbit/s, half-duplex
<i>ENET_LOOPBACKMODE</i>	MAC in loopback mode at the MII

Input parameter{in}	
<b>checksum</b>	IP frame checksum offload function, only one parameter can be selected
<i>ENET_NO_AUTOCHECKSUM</i>	disable IP frame checksum function
<i>ENET_AUTOCHECKSUM</i>	enable IP frame checksum function
<i>UM_DROP_FAILFRAMES</i>	
<i>AMES</i>	enable IP frame checksum function, and the received frame with only payload error but no other errors will not be dropped
Input parameter{in}	
<b>receipt</b>	frame filter function, only one parameter can be selected
<i>ENET_PROMISCUOUS_MODE</i>	promiscuous mode enabled
<i>ENET_RECEIVEALL</i>	all received frame are forwarded to application
<i>ENET_BROADCAST_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
Output parameter{out}	
-	-
Return value	
<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 3-203. Function enet\_software\_reset**

<b>Function name</b>	enet_software_reset
<b>Function prototype</b>	ErrStatus enet_software_reset(void);
<b>Function descriptions</b>	reset all core internal registers located in CLK_TX and CLK_RX
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* reset all core internal registers located in CLK_TX and CLK_RX */

ErrStatus reval_state = ERROR;

reval_state = enet_software_reset();
```

### **enet\_rxframe\_size\_get**

The description of enet\_rxframe\_size\_get is shown as below:

**Table 3-204. Function enet\_rxframe\_size\_get**

<b>Function name</b>	enet_rxframe_size_get
<b>Function prototype</b>	uint32_t enet_rxframe_size_get(void);
<b>Function descriptions</b>	check receive frame valid and return frame size
<b>Precondition</b>	-
<b>The called functions</b>	enet_rxframe_drop()
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

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 3-205. Function enet\_descriptors\_chain\_init**

<b>Function name</b>	enet_descriptors_chain_init
<b>Function prototype</b>	void enet_descriptors_chain_init(enet_dmadirection_enum direction);
<b>Function descriptions</b>	initialize the DMA Tx/Rx descriptors's parameters in chain mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>direction</b>	the descriptors which users want to init only one parameter can be selected which is shown as below
<i>ENET_DMA_TX</i>	DMA Tx descriptors
<i>ENET_DMA_RX</i>	DMA Rx descriptors
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the DMA Tx/Rx descriptors's parameters in chain mode */

enet_descriptors_chain_init(ENET_DMA_TX);
```

### **enet\_descriptors\_ring\_init**

The description of enet\_descriptors\_ring\_init is shown as below:

**Table 3-206. Function enet\_descriptors\_ring\_init**

<b>Function name</b>	enet_descriptors_ring_init
<b>Function prototype</b>	void enet_descriptors_ring_init(enet_dmadirection_enum direction);
<b>Function descriptions</b>	initialize the DMA Tx/Rx descriptors's parameters in ring mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>direction</b>	the descriptors which users want to init only one parameter can be selected which is shown as below
<i>ENET_DMA_TX</i>	DMA Tx descriptors
<i>ENET_DMA_RX</i>	DMA Rx descriptors
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the DMA Tx/Rx descriptors's parameters in ring mode */
enet_descriptors_ring_init(ENET_DMA_TX);
```

### **enet\_frame\_receive**

The description of enet\_frame\_receive is shown as below:

**Table 3-207. 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)

Output parameter{out}	
<b>buffer</b>	pointer to the received frame data if the input is NULL, user should copy data in application by himself
Return value	
<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 3-208. 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>	-
Input parameter{in}	
<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
Input parameter{in}	
<b>length</b>	the length of frame data to be transmitted, (0 -- 1524)
Output parameter{out}	
-	-
Return value	
<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 3-209. Function enet\_transmit\_checksum\_config**

<b>Function name</b>	enet_transmit_checksum_config
<b>Function prototype</b>	void enet_transmit_checksum_config(enet_descriptors_struct *desc, uint32_t checksum);
<b>Function descriptions</b>	configure the transmit IP frame checksum offload calculation and insertion
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>desc</b>	the descriptor pointer which users want to configure, the structure members can refer to <a href="#">Structure enet_descriptors_struct</a>
<b>Input parameter{in}</b>	
<b>checksum</b>	IP frame checksum configuration only one parameter can be selected which is shown as below
<i>ENET_CHECKSUM_DISABLE</i>	checksum insertion disabled
<i>ENET_CHECKSUM_IP_V4HEADER</i>	only IP header checksum calculation and insertion are enabled
<i>ENET_CHECKSUM_TCUPDPICMP_SEGMENT</i>	TCP/UDP/ICMP checksum insertion calculated but pseudo-header
<i>ENET_CHECKSUM_TCUPDPICMP_FULL</i>	TCP/UDP/ICMP checksum insertion fully calculated
<b>Output parameter{out}</b>	
<b>Return value</b>	

Example:

```
/* configure the transmit IP frame checksum offload calculation and insertion */

enet_descriptors_struct rx_desc;

enet_transmit_checksum_config(rx_desc, ENET_CHECKSUM_TCPUDPICMP_FULL);
```

### **enet\_enable**

The description of enet\_enable is shown as below:

**Table 3-210. 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 3-211. 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 3-212. Function enet\_mac\_address\_set**

<b>Function name</b>	enet_mac_address_set
<b>Function prototype</b>	void enet_mac_address_set(enet_macaddress_enum mac_addr, uint8_t paddr[]);
<b>Function descriptions</b>	configure MAC address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mac_addr</b>	select which MAC address will be set only one parameter can be selected which is shown as below
<i>ENET_MAC_ADDRESS_S0</i>	set MAC address 0 filter
<i>ENET_MAC_ADDRESS_S1</i>	set MAC address 1 filter
<i>ENET_MAC_ADDRESS_S2</i>	set MAC address 2 filter
<i>ENET_MAC_ADDRESS_S3</i>	set MAC address 3 filter

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

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 3-213. Function enet\_mac\_address\_get**

<b>Function name</b>	enet_mac_address_get
<b>Function prototype</b>	void enet_mac_address_get(enet_macaddress_enum mac_addr, uint8_t paddr[]);
<b>Function descriptions</b>	get MAC address
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
mac_addr	select which MAC address will be set only one parameter can be selected which is shown as below

<i>ENET_MAC_ADDRESS0</i>	set MAC address 0 filter
<i>ENET_MAC_ADDRESS1</i>	set MAC address 1 filter
<i>ENET_MAC_ADDRESS2</i>	set MAC address 2 filter
<i>ENET_MAC_ADDRESS3</i>	set MAC address 3 filter
<b>Output parameter{out}</b>	
<b>paddr</b>	the buffer pointer which stores the MAC address little-ending store, such as MAC address is aa:bb:cc:dd:ee:22, the buffer is {22, ee, dd, cc, bb, aa}
<b>Return value</b>	
-	-

Example:

```
/* get mac address */
enet_mac_address_get (ENET_MAC_ADDRESS0, netif->hwaddr);
```

### enet\_flag\_get

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

**Table 3-214. Function enet\_flag\_get**

<b>Function name</b>	enet_flag_get
<b>Function prototype</b>	FlagStatus enet_flag_get(enet_flag_enum enet_flag);
<b>Function descriptions</b>	get the ENET MAC/MSC/PTP/DMA status flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>enet_flag</b>	ENET status flag, refer to enet_flag_enum only one parameter can be selected which is shown as below
<i>ENET_MAC_FLAG_MP_KR</i>	magic packet received flag

<i>ENET_MAC_FLAG_WUFR</i>	wakeup frame received flag
<i>ENET_MAC_FLAG_FLOWCONTROL</i>	flow control status flag
<i>ENET_MAC_FLAG_WUM</i>	WUM status flag
<i>ENET_MAC_FLAG_MSCC</i>	MSC status flag
<i>ENET_MAC_FLAG_MSCCR</i>	MSC receive status flag
<i>ENET_MAC_FLAG_MSCCT</i>	MSC transmit status flag
<i>ENET_MAC_FLAG_TMS</i>	time stamp trigger status flag
<i>ENET_PTP_FLAG_TS</i>	timestamp second counter overflow flag
<i>ENET_PTP_FLAG_TTM</i>	target time match flag
<i>ENET_MSC_FLAG_RFCE</i>	received frames CRC error flag
<i>ENET_MSC_FLAG_RFAE</i>	received frames alignment error flag
<i>ENET_MSC_FLAG_RGUF</i>	received good unicast frames flag
<i>ENET_MSC_FLAG_TGFSC</i>	transmitted good frames single collision flag
<i>ENET_MSC_FLAG_TGFMSC</i>	transmitted good frames more single collision flag
<i>ENET_MSC_FLAG_TGF</i>	transmitted good frames flag
<i>ENET_DMA_FLAG_TS</i>	transmit status flag
<i>ENET_DMA_FLAG_TPS</i>	transmit process stopped status flag
<i>ENET_DMA_FLAG_TB</i>	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_RW</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>	
-	-

Return value	
FlagStatus	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 3-215. Function enet\_flag\_clear**

<b>Function name</b>	enet_flag_clear
<b>Function prototype</b>	void enet_flag_clear(enet_flag_clear_enum enet_flag);
<b>Function descriptions</b>	clear the ENET DMA status flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>enet_flag</b>	ENET DMA flag clear, refer to enet_flag_clear_enum only one parameter can be selected which is shown as below
<i>ENET_DMA_FLAG_TS_CLR</i>	transmit status flag clear
<i>ENET_DMA_FLAG_TP_S_CLR</i>	transmit process stopped status flag clear
<i>ENET_DMA_FLAG_TB_U_CLR</i>	transmit buffer unavailable status flag clear
<i>ENET_DMA_FLAG_TJ_T_CLR</i>	transmit jabber timeout status flag clear
<i>ENET_DMA_FLAG_RO_CLR</i>	receive overflow status flag clear
<i>ENET_DMA_FLAG_TU_CLR</i>	transmit underflow status flag clear
<i>ENET_DMA_FLAG_RS_CLR</i>	receive status flag clear
<i>ENET_DMA_FLAG_RB</i>	receive buffer unavailable status flag clear

<i>U_CLR</i>	
<i>ENET_DMA_FLAG_RP_S_CLR</i>	receive process stopped status flag clear
<i>ENET_DMA_FLAG_R_WT_CLR</i>	receive watchdog timeout status flag clear
<i>ENET_DMA_FLAG_ET_CLR</i>	early transmit status flag clear
<i>ENET_DMA_FLAG_FB_E_CLR</i>	fatal bus error status flag clear
<i>ENET_DMA_FLAG_ER_CLR</i>	early receive status flag clear
<i>ENET_DMA_FLAG_AI_CLR</i>	abnormal interrupt summary flag clear
<i>ENET_DMA_FLAG_NI_CLR</i>	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 3-216. Function enet\_interrupt\_enable**

<b>Function name</b>	enet_interrupt_enable
<b>Function prototype</b>	void enet_interrupt_enable(enet_int_enum enet_int);
<b>Function descriptions</b>	enable ENET MAC/MSC/DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>enet_int</b>	ENET interrupt only one parameter can be selected which is shown as below
<i>ENET_MAC_INT_WUM_IM</i>	WUM interrupt mask
<i>ENET_MAC_INT_TMS_TIM</i>	timestamp trigger interrupt mask
<i>ENET_MSC_INT_RFC_EIM</i>	received frame CRC error interrupt mask
<i>ENET_MSC_INT_RFA_EIM</i>	received frames alignment error interrupt mask
<i>ENET_MSC_INT_RGU_FIM</i>	received good unicast frames interrupt mask
<i>ENET_MSC_INT_TGF_SCIM</i>	transmitted good frames single collision interrupt mask
<i>ENET_MSC_INT_TGF_MSCIM</i>	transmitted good frames more single collision interrupt mask
<i>ENET_MSC_INT_TGFI_M</i>	transmitted good frames interrupt mask
<i>ENET_DMA_INT_TIE</i>	transmit interrupt enable
<i>ENET_DMA_INT_TPSI_E</i>	transmit process stopped interrupt enable
<i>ENET_DMA_INT_TBUI_E</i>	transmit buffer unavailable interrupt enable
<i>ENET_DMA_INT_TJTI_E</i>	transmit jabber timeout interrupt enable
<i>ENET_DMA_INT_ROIE</i>	receive overflow interrupt enable
<i>ENET_DMA_INT_TUIE</i>	transmit underflow interrupt enable
<i>ENET_DMA_INT_RIE</i>	receive interrupt enable
<i>ENET_DMA_INT_RBUI_E</i>	receive buffer unavailable interrupt enable
<i>ENET_DMA_INT_RPSI_E</i>	receive process stopped interrupt enable
<i>ENET_DMA_INT_RWTIE</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:

```
/* 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 3-217. 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 only one parameter can be selected which is shown as below
<i>ENET_MAC_INT_WUM_IM</i>	WUM interrupt mask
<i>ENET_MAC_INT_TMS_TIM</i>	timestamp trigger interrupt mask
<i>ENET_MSC_INT_RFC</i>	received frame CRC error interrupt mask

<i>EIM</i>	
<i>ENET_MSC_INT_RFA_EIM</i>	received frames alignment error interrupt mask
<i>ENET_MSC_INT_RGU_FIM</i>	received good unicast frames interrupt mask
<i>ENET_MSC_INT_TGF_SCIM</i>	transmitted good frames single collision interrupt mask
<i>ENET_MSC_INT_TGF_MSCIM</i>	transmitted good frames more single collision interrupt mask
<i>ENET_MSC_INT_TGFI_M</i>	transmitted good frames interrupt mask
<i>ENET_DMA_INT_TIE</i>	transmit interrupt enable
<i>ENET_DMA_INT_TPSI_E</i>	transmit process stopped interrupt enable
<i>ENET_DMA_INT_TBUI_E</i>	transmit buffer unavailable interrupt enable
<i>ENET_DMA_INT_TJTI_E</i>	transmit jabber timeout interrupt enable
<i>ENET_DMA_INT_ROIE</i>	receive overflow interrupt enable
<i>ENET_DMA_INT_TUIE</i>	transmit underflow interrupt enable
<i>ENET_DMA_INT_RIE</i>	receive interrupt enable
<i>ENET_DMA_INT_RBUI_E</i>	receive buffer unavailable interrupt enable
<i>ENET_DMA_INT_RPSI_E</i>	receive process stopped interrupt enable
<i>ENET_DMA_INT_RWT_IE</i>	receive watchdog timeout interrupt enable
<i>ENET_DMA_INT_ETIE</i>	early transmit interrupt enable
<i>ENET_DMA_INT_FBEI_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

Output parameter{out}	
-	-
Return value	
-	-

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 3-218. Function enet\_interrupt\_flag\_get**

Function name	enet_interrupt_flag_get
Function prototype	FlagStatus enet_interrupt_flag_get(enet_int_flag_enum int_flag);
Function descriptions	get ENET MAC/MSC/DMA interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
<b>int_flag</b>	ENET interrupt flag only one parameter can be selected which is shown as below
<i>ENET_MAC_INT_FLA G_WUM</i>	WUM status flag
<i>ENET_MAC_INT_FLA G_MSC</i>	MSC status flag
<i>ENET_MAC_INT_FLA G_MSFR</i>	MSC receive status flag
<i>ENET_MAC_INT_FLA G_MSCT</i>	MSC transmit status flag
<i>ENET_MAC_INT_FLA G_TMST</i>	time stamp trigger status flag
<i>ENET_MSC_INT_FLA G_RFCE</i>	received frames CRC error flag

<i>ENET_MSC_INT_FLA</i> <i>G_RFAE</i>	received frames alignment error flag
<i>ENET_MSC_INT_FLA</i> <i>G_RGUF</i>	received good unicast frames flag
<i>ENET_MSC_INT_FLA</i> <i>G_TGFSC</i>	transmitted good frames single collision flag
<i>ENET_MSC_INT_FLA</i> <i>G_TGFMSC</i>	transmitted good frames more single collision flag
<i>ENET_MSC_INT_FLA</i> <i>G_TGF</i>	transmitted good frames flag
<i>ENET_DMA_INT_FLA</i> <i>G_TS</i>	transmit status flag
<i>ENET_DMA_INT_FLA</i> <i>G_TPS</i>	transmit process stopped status flag
<i>ENET_DMA_INT_FLA</i> <i>G_TBU</i>	transmit buffer unavailable status flag
<i>ENET_DMA_INT_FLA</i> <i>G_TJT</i>	transmit jabber timeout status flag
<i>ENET_DMA_INT_FLA</i> <i>G_RO</i>	receive overflow status flag
<i>ENET_DMA_INT_FLA</i> <i>G_TU</i>	transmit underflow status flag
<i>ENET_DMA_INT_FLA</i> <i>G_RS</i>	receive status flag
<i>ENET_DMA_INT_FLA</i> <i>G_RBU</i>	receive buffer unavailable status flag
<i>ENET_DMA_INT_FLA</i> <i>G_RPS</i>	receive process stopped status flag
<i>ENET_DMA_INT_FLA</i> <i>G_RWT</i>	receive watchdog timeout status flag
<i>ENET_DMA_INT_FLA</i> <i>G_ET</i>	early transmit status flag
<i>ENET_DMA_INT_FLA</i> <i>G_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 3-219. Function enet\_interrupt\_flag\_clear**

<b>Function name</b>	enet_interrupt_flag_clear
<b>Function prototype</b>	void enet_interrupt_flag_clear(enet_int_flag_clear_enum int_flag_clear);
<b>Function descriptions</b>	clear ENET DMA interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag_clear</b>	clear ENET interrupt flag only one parameter can be selected which is shown as below

<i>ENET_DMA_INT_FLA</i>	
<i>G_TS_CLR</i>	transmit status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_TPS_CLR</i>	transmit process stopped status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_TBU_CLR</i>	transmit buffer unavailable status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_TJT_CLR</i>	transmit jabber timeout status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_RO_CLR</i>	receive overflow status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_TU_CLR</i>	transmit underflow status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_RS_CLR</i>	receive status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_RBU_CLR</i>	receive buffer unavailable status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_RPS_CLR</i>	receive process stopped status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_RWT_CLR</i>	receive watchdog timeout status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_ET_CLR</i>	early transmit status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_FBE_CLR</i>	fatal bus error status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_ER_CLR</i>	early receive status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_AI_CLR</i>	abnormal interrupt summary flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_NI_CLR</i>	normal interrupt summary flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-220. 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 3-221. 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()

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

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 3-222. 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>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

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 3-223. 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 3-224. Function enet\_registers\_get**

<b>Function name</b>	enet_registers_get
<b>Function prototype</b>	void enet_registers_get(enet_registers_type_enum type, uint32_t *preg, uint32_t num);
<b>Function descriptions</b>	put registers value into the application buffer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>type</b>	register type which will be get only one parameter can be selected which is shown as below
<b>ALL_MAC_REG</b>	get the registers within the offset scope range ENET_MAC_CFG to ENET_MAC_FCTH

<i>ALL_MSC_REG</i>	get the registers within the offset scope range ENET_MSC_CTL to ENET_MSC_RGUFCNT
<i>ALL_PTP_REG</i>	get the registers within the offset scope range ENET_PTP_TSCTL to ENET_PTP_PPSCTL
<i>ALL_DMA_REG</i>	get the registers within the offset scope range ENET_DMA_BCTL to ENET_DMA_CRBADDR
<b>Input parameter{in}</b>	
<b>num</b>	the number of registers that the user want to get, (0 -- 54)
<b>Output parameter{out}</b>	
<b>preg</b>	the application buffer pointer for storing the register value
<b>Return value</b>	
-	-

Example:

```
/* get all mac registers value */

uint32_t register_buffer[5];

enet_registers_get(ALL_MAC_REG, 5, register_buffer);
```

### **enet\_debug\_status\_get**

The description of `enet_debug_status_get` is shown as below:

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

<b>Function name</b>	enet_debug_status_get
<b>Function prototype</b>	uint32_t enet_debug_status_get(uint32_t mac_debug);
<b>Function descriptions</b>	get the enet debug status from the debug register
<b>Precondition</b>	-
<b>The called functions</b>	--
<b>Input parameter{in}</b>	
<b>mac_debug</b>	enet debug status only one parameter can be selected which is shown as below
<i>ENET_MAC_RECEIVE_R_NOT_IDLE</i>	MAC receiver is not in idle state
<i>ENET_RX_ASYNCNHR</i>	Rx asynchronous FIFO status

<code>ONOUS_FIFO_STATE</code>	
<code>ENET_RXFIFO_WRITING</code>	RxFIFO is doing write operation
<code>ENET_RXFIFO_READ_STATUS</code>	RxFIFO read operation status
<code>ENET_RXFIFO_STATE</code>	RxFIFO state
<code>ENET_MAC_TRANSMITTER_NOT_IDLE</code>	MAC transmitter is not in idle state
<code>ENET_MAC_TRANSMITTER_STATUS</code>	status of MAC transmitter
<code>ENET_PAUSE_CONDITION_STATUS</code>	pause condition status
<code>ENET_TXFIFO_READ_STATUS</code>	TxFIFO read operation status
<code>ENET_TXFIFO_WRITING</code>	TxFIFO is doing write operation
<code>ENET_TXFIFO_NOT_EMPTY</code>	TxFIFO is not empty
<code>ENET_TXFIFO_FULL</code>	TxFIFO is full
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<code>uint32_t</code>	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 3-226. Function `enet_address_filter_enable`**

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

<b>Function prototype</b>	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 only one parameter can be selected which is shown as below
<i>ENET_MAC_ADDRESS1</i>	enable MAC address 1 filter
<i>ENET_MAC_ADDRESS2</i>	enable MAC address 2 filter
<i>ENET_MAC_ADDRESS3</i>	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 3-227. 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 only one parameter can be selected which is shown as below
<i>ENET_MAC_ADDRESS1</i>	enable MAC address 1 filter
<i>ENET_MAC_ADDRESS2</i>	enable MAC address 2 filter
<i>ENET_MAC_ADDRESS3</i>	enable MAC address 3 filter
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-228. 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 only one parameter can be selected which is shown as below
<i>ENET_MAC_ADDRESS1</i>	enable MAC address 1 filter
<i>ENET_MAC_ADDRESS2</i>	enable MAC address 2 filter

<i>ENET_MAC_ADDRESS3</i>	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
<i>ENET_ADDRESS_MASK_BYTE0</i>	mask ENET_MAC_ADDR1L[7:0] bits
<i>ENET_ADDRESS_MASK_BYTE1</i>	mask ENET_MAC_ADDR1L[15:8] bits
<i>ENET_ADDRESS_MASK_BYTE2</i>	mask ENET_MAC_ADDR1L[23:16] bits
<i>ENET_ADDRESS_MASK_BYTE3</i>	mask ENET_MAC_ADDR1L [31:24] bits
<i>ENET_ADDRESS_MASK_BYTE4</i>	mask ENET_MAC_ADDR1H [7:0] bits
<i>ENET_ADDRESS_MASK_BYTE5</i>	mask ENET_MAC_ADDR1H [15:8] bits
<b>Input parameter{in}</b>	
<b>filter_type</b>	select which MAC address filter type will be selected only one parameter can be selected which is shown as below
<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 3-229. 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 3-230. Function enet\_phy\_write\_read**

<b>Function name</b>	enet_phy_write_read
<b>Function prototype</b>	ErrStatus enet_phy_write_read(enet_phydirection_enum direction, uint16_t phy_address, uint16_t phy_reg, uint16_t *pvalue);
<b>Function descriptions</b>	write to / read from a PHY register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>direction</b>	only one parameter can be selected which is shown as below
<i>ENET_PHY_WRITE</i>	write data to phy register

<b>ENET_PHY_READ</b>	read data from phy register
<b>Input parameter{in}</b>	
<b>phy_address</b>	0x0 - 0x1F
<b>Input parameter{in}</b>	
<b>phy_reg</b>	0x0 - 0x1F
<b>Input parameter{in}</b>	
<b>pvalue</b>	the value will be written to the PHY register in ENET_PHY_WRITE direction
<b>Output parameter{out}</b>	
<b>pvalue</b>	the value will be read from the PHY register in ENET_PHY_READ direction
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* write 0 to PHY BCR register */

uint16_t temp_phy = 0U;

phy_state = enet_phy_write_read(ENET_PHY_WRITE, PHY_ADDRESS, PHY_REG_BCR,
&temp_phy);
```

### **enet\_phyloopback\_enable**

The description of enet\_phyloopback\_enable is shown as below:

**Table 3-231. 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 3-232. Function enet\_phyloopback\_disable**

<b>Function name</b>	enet_phyloopback_disable
<b>Function prototype</b>	ErrStatus enet_phyloopback_disable(void);
<b>Function descriptions</b>	disable the loopback function of PHY chip
<b>Precondition</b>	-
<b>The called functions</b>	enet_phy_write_read
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* disable the loopback function of PHY chip */

ErrStatus phy_state = ERROR;

phy_state = enet_phyloopback_disable();
```

### **enet\_forward\_feature\_enable**

The description of enet\_forward\_feature\_enable is shown as below:

**Table 3-233. Function enet\_forward\_feature\_enable**

<b>Function name</b>	enet_forward_feature_enable
----------------------	-----------------------------

<b>Function prototype</b>	void enet_forward_feature_enable(uint32_t feature);
<b>Function descriptions</b>	enable ENET forward feature
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>feature</b>	the feature of ENET forward mode one or more parameters can be selected which are shown as below
<i>ENET_AUTO_PADCR_C_DROP</i>	the function of the MAC strips the Pad/FCS field on received frames
<i>ENET_TYPEFRAME_C_RC_DROP</i>	the function that FCS field(last 4 bytes) of frame will be dropped before forwarding
<i>ENET_FORWARD_ER_RFRAMES</i>	the function that all frame received with error except runt error are forwarded to memory
<i>ENET_FORWARD_UNDERSZ_GOODFRAME_S</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 3-234. 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_PADCR</i> <i>C_DROP</i>	the function of the MAC strips the Pad/FCS field on received frames
<i>ENET_TYPEFRAME_C</i> <i>RC_DROP</i>	the function that FCS field(last 4 bytes) of frame will be dropped before forwarding
<i>ENET_FORWARD_ER</i> <i>RFRAMES</i>	the function that all frame received with error except runt error are forwarded to memory
<i>ENET_FORWARD_UN</i> <i>DERSZ_GOODFRAME</i> <i>S</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\_fliter\_feature\_enable**

The description of enet\_fliter\_feature\_enable is shown as below:

**Table 3-235. 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
<i>ENET_SRC_FILTER</i>	filter source address function
<i>ENET_SRC_FILTER_INVERSE</i>	inverse source address filtering result function
<i>ENET_DEST_FILTER_INVERSE</i>	inverse DA filtering result function
<i>ENET_MULTICAST_FILTER_PASS</i>	pass all multicast frames function
<i>ENET_MULTICAST_FILTER_HASH_MODE</i>	HASH multicast filter function
<i>ENET_UNICAST_FILTER_HASH_MODE</i>	HASH unicast filter function
<i>ENET_FILTER_MODE_EITHER</i>	HASH or perfect filter function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable filter source address function */
enet_filter_feature_enable(ENET_SRC_FILTER);
```

### **enet\_filter\_feature\_disable**

The description of enet\_filter\_feature\_disable is shown as below:

**Table 3-236. Function enet\_filter\_feature\_disable**

<b>Function name</b>	enet_filter_feature_disable
<b>Function prototype</b>	void enet_filter_feature_disable(uint32_t feature);
<b>Function descriptions</b>	disable ENET filter feature
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>feature</b>	the feature of ENET filter mode one or more parameters can be selected which are shown as below
<i>ENET_SRC_FILTER</i>	filter source address function
<i>ENET_SRC_FILTER_INVERSE</i>	inverse source address filtering result function
<i>ENET_DEST_FILTER_INVERSE</i>	inverse DA filtering result function
<i>ENET_MULTICAST_FILTER_PASS</i>	pass all multicast frames function
<i>ENET_MULTICAST_FILTER_HASH_MODE</i>	HASH multicast filter function
<i>ENET_UNICAST_FILTER_HASH_MODE</i>	HASH unicast filter function
<i>ENET_FILTER_MODE_EITHER</i>	HASH or perfect filter function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable filter source address function */
enet_filter_feature_enable(ENET_SRC_FILTER);
```

### **enet\_pauseframe\_generate**

The description of enet\_pauseframe\_generate is shown as below:

**Table 3-237. 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 rval;
rval = enet_pauseframe_generate();
```

### **enet\_pauseframe\_detect\_config**

The description of `enet_pauseframe_detect_config` is shown as below:

**Table 3-238. 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_PAUSEDTECT</i>	besides the unique multicast address, MAC can also use the MAC0 address to detecting pause frame
<i>ENET_UNIQUE_PAUSEDTECT</i>	only the unique multicast address for pause frame which is specified in IEEE802.3 can be detected
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* config the pause frame detect type as ENET_UNIQUE_PAUSEDTECT */
enet_pauseframe_detect_config(ENET_UNIQUE_PAUSEDTECT);
```

### enet\_pauseframe\_config

The description of enet\_pauseframe\_config is shown as below:

**Table 3-239. 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
<i>ENET_PAUSETIME_MI_NUS4</i>	pause time minus 4 slot times
<i>ENET_PAUSETIME_MI_NUS28</i>	pause time minus 28 slot times
<i>ENET_PAUSETIME_MI_NUS144</i>	pause time minus 144 slot times
<i>ENET_PAUSETIME_MI_NUS256</i>	pause time minus 256 slot times
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

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 3-240. Function enet\_flowcontrol\_threshold\_config**

Function name		enet_flowcontrol_threshold_config
Function prototype		void enet_flowcontrol_threshold_config(uint32_t deactive, uint32_t active);
Function descriptions		configure the threshold of the flow control(deactive and active threshold)
Precondition		-
The called functions		-
Input parameter{in}		
<b>deactive</b>	the threshold of the deactive flow control, this value should always be less than active flow control value, only one parameter can be selected which is shown as below	
<i>ENET_DEACTIVE_TH RESHOLD_256BYTES</i>	threshold level is 256 bytes	
<i>ENET_DEACTIVE_TH RESHOLD_512BYTES</i>	threshold level is 512 bytes	
<i>ENET_DEACTIVE_TH RESHOLD_768BYTES</i>	threshold level is 768 bytes	
<i>ENET_DEACTIVE_TH RESHOLD_1024BYTES</i>	threshold level is 1024 bytes	
<i>ENET_DEACTIVE_TH RESHOLD_1280BYTES</i>	threshold level is 1280 bytes	
<i>ENET_DEACTIVE_TH RESHOLD_1536BYTES</i>	threshold level is 1536 bytes	

<i>ENET_DEACTIVE_THRESHOLD_1792BYTES</i>	threshold level is 1792 bytes
<b>Input parameter{in}</b>	
<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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the threshold of the flow control */
enet_flowcontrol_threshold_config(ENET_DEACTIVE_THRESHOLD_256BYTES, ENET_ACTIVE_THRESHOLD_256BYTES);
```

#### **enet\_flowcontrol\_feature\_enable**

The description of enet\_flowcontrol\_feature\_enable is shown as below:

**Table 3-241. Function enet\_flowcontrol\_feature\_enable**

Function name	enet_flowcontrol_feature_enable
---------------	---------------------------------

<b>Function prototype</b>	void enet_flowcontrol_feature_enable(uint32_t feature);
<b>Function descriptions</b>	enable ENET flow control feature
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>feature</b>	the feature of ENET flow control mode one or more parameters can be selected which are shown as below
<i>ENET_ZERO_QUANT_A_PAUSE</i>	the automatic zero-quanta generation function
<i>ENET_TX_FLOWCONTROL</i>	the flow control operation in the MAC
<i>ENET_RX_FLOWCONTROL</i>	decoding function for the received pause frame and process it
<i>ENET_BACK_PRESSURE</i>	back pressure operation in the MAC(only use in half-duplex mode)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the flow control operation in the MAC */
enet_flowcontrol_feature_enable(ENET_ZERO_QUANTA_PAUSE);
```

### **enet\_flowcontrol\_feature\_disable**

The description of enet\_flowcontrol\_feature\_disable is shown as below:

**Table 3-242. 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>	-

Input parameter{in}	
<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_QUANT_A_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)
Output parameter{out}	
-	-
Return value	
-	-

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 3-243. 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>	-
Input parameter{in}	
<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>	
<i>uint32_t</i>	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_QUEUEING / 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 eval;

eval = enet_dmaprocess_state_get(ENET_DMA_RX);

if(ENET_RX_STATE_SUSPENDED == eval){

    do...

}
```

### **enet\_dmaprocess\_resume**

The description of `enet_dmaprocess_resume` is shown as below:

**Table 3-244. 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 3-245. 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 3-246. 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 3-247. Function enet\_current\_desc\_address\_get**

<b>Function name</b>	enet_current_desc_address_get
<b>Function prototype</b>	uint32_t enet_current_desc_address_get(enet_desc_reg_enum addr_get);
<b>Function descriptions</b>	get the transmit/receive address of current descriptor, or current buffer, or descriptor table
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>addr_get</b>	choose the address which users want to get

	only one parameter can be selected which is shown as below
<i>ENET_RX_DESC_TAB</i> <i>LE</i>	the start address of the receive descriptor table
<i>ENET_RX_CURRENT_</i> <i>DESC</i>	the start descriptor address of the current receive descriptor read by the RxDMA controller
<i>ENET_RX_CURRENT_</i> <i>BUFFER</i>	the current receive buffer address being read by the RxDMA controller
<i>ENET_TX_DESC_TAB</i> <i>LE</i>	the start address of the transmit descriptor table
<i>ENET_TX_CURRENT_</i> <i>DESC</i>	the start descriptor address of the current transmit descriptor read by the TxDMA controller
<i>ENET_TX_CURRENT_</i> <i>BUFFER</i>	the current transmit buffer address being read by the TxDMA controller
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	0- 0xFFFFFFFF

Example:

```
/* get the start address of the receive descriptor table */
uint32_t reval;
reval = enet_current_desc_address_get(ENET_RX_DESC_TABLE);
```

### **enet\_desc\_information\_get**

The description of enet\_desc\_information\_get is shown as below:

**Table 3-248. Function enet\_desc\_information\_get**

<b>Function name</b>	enet_desc_information_get
<b>Function prototype</b>	uint32_t enet_desc_information_get(enet_descriptors_struct *desc, enetc_descstate_enum info_get);
<b>Function descriptions</b>	get the Tx or Rx descriptor information
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>desc</b>	the descriptor pointer which users want to get information, the structure members can refer to <a href="#">Structure_enet_descriptors_struct</a>
<b>Input parameter{in}</b>	
<b>info_get</b>	the descriptor information type which is selected only one parameter can be selected which is shown as below
<i>RXDESC_BUFFER_1_SIZE</i>	receive buffer 1 size
<i>RXDESC_BUFFER_2_SIZE</i>	receive buffer 2 size
<i>RXDESC_FRAME_LENGTH</i>	the byte length of the received frame that was transferred to the buffer
<i>TXDESC_COLLISION_COUNT</i>	the number of collisions occurred before the frame was transmitted
<i>RXDESC_BUFFER_1_ADDR</i>	the buffer1 address of the Rx frame
<i>TXDESC_BUFFER_1_ADDR</i>	the buffer1 address of the Tx frame
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	descriptor information if value is 0xFFFFFFFFU, means the false input parameter

Example:

```
/* get the reception buffer 1 size */

uint32_t reval;

reval = enet_desc_information_get(rx_desc, RXDESC_BUFFER_1_SIZE);
```

#### **enet\_missed\_frame\_counter\_get**

The description of `enet_missed_frame_counter_get` is shown as below:

**Table 3-249. 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 Rx FIFO
<b>Output parameter{out}</b>	
<b>rxdma_drop</b>	pointer to the number of frames missed by the Rx DMA 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 3-250. Function enet\_desc\_flag\_get**

<b>Function name</b>	enet_desc_flag_get
<b>Function prototype</b>	FlagStatus enet_desc_flag_get(enet_descriptors_struct *desc, uint32_t desc_flag);
<b>Function descriptions</b>	get the bit flag of ENET DMA descriptor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>desc</b>	the descriptor pointer which users want to get flag, the structure members can refer to <a href="#">Structure enet_descriptors_struct</a>
<b>Input parameter{in}</b>	
<b>desc_flag</b>	the bit flag of ENET DMA descriptor

(the value according to the parameter <b>desc</b> )	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 R</i>	IP frame header error
<i>ENET_RDES0_LDES</i>	last descriptor
<i>ENET_RDES0_FDES</i>	first descriptor
<i>ENET_RDES0_VTAG</i>	VLAN tag
<i>ENET_RDES0_OERR</i>	overflow error
<i>ENET_RDES0_LERR</i>	length error
<i>ENET_RDES0_SAFF</i>	SA filter fail
<i>ENET_RDES0_DERR</i>	descriptor error
<i>ENET_RDES0_ERRS</i>	error summary
<i>ENET_RDES0_DAFF</i>	destination address filter fail
<i>ENET_RDES0_DAV</i>	descriptor available
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the bit flag of ENET DMA descriptor */
FlagStatus reval;
reval = enet_desc_flag_get(p_txdesc, ENET_TDES0_TCHM);
```

### **enet\_desc\_flag\_set**

The description of `enet_desc_flag_set` is shown as below:

Table 3-251. Function enet\_desc\_flag\_set

<b>Function name</b>	enet_desc_flag_set
<b>Function prototype</b>	void enet_desc_flag_set(enet_descriptors_struct *desc, uint32_t desc_flag);
<b>Function descriptions</b>	set the bit flag of ENET DMA descriptor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>desc</b>	the descriptor pointer which users want to set flag, the structure members can refer to <a href="#">Structure enet_descriptors_struct</a>
<b>Input parameter{in}</b>	
<b>desc_flag</b> (the value according to the parameter <b>desc</b> )	the bit flag of ENET DMA descriptor only one parameter can be selected which is shown as below
When type of parameter <b>desc</b> is TX	
<i>ENET_TDES0_VFRM</i>	VLAN frame
<i>ENET_TDES0_FRMF</i>	frame flushed
<i>ENET_TDES0_TCHM</i>	the second address chained mode
<i>ENET_TDES0_TERM</i>	transmit end of ring mode
<i>ENET_TDES0_TTSEN</i>	transmit timestamp function enable
<i>ENET_TDES0_DPAD</i>	disable adding pad
<i>ENET_TDES0_DCRC</i>	disable CRC
<i>ENET_TDES0_FSG</i>	first segment
<i>ENET_TDES0_LSG</i>	last segment
<i>ENET_TDES0_INTC</i>	interrupt on completion
<i>ENET_TDES0_DAV</i>	DAV bit
When type of parameter <b>desc</b> is RX	
<i>ENET_RDES0_DAV</i>	descriptor available
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* set VLAN frame bit flag of ENET DMA descriptor */
enet_desc_flag_set(p_txdesc, ENET_TDES0_VFRM);
```

### **enet\_desc\_flag\_clear**

The description of enet\_desc\_flag\_clear is shown as below:

**Table 3-252. Function enet\_desc\_flag\_clear**

<b>Function name</b>	enet_desc_flag_clear
<b>Function prototype</b>	void enet_desc_flag_clear(enet_descriptors_struct *desc, uint32_t desc_flag);
<b>Function descriptions</b>	clear the bit flag of ENET DMA descriptor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>desc</b>	the descriptor pointer which users want to set flag, the structure members can refer to <a href="#">Structure enet descriptors struct</a>
<b>Input parameter{in}</b>	
<b>desc_flag</b> (the value according to the parameter <b>desc</b> )	the bit flag of ENET DMA descriptor only one parameter can be selected which is shown as below
When type of parameter <b>desc</b> is TX	
<i>ENET_TDES0_VFRM</i>	VLAN frame
<i>ENET_TDES0_FRMF</i>	frame flushed
<i>ENET_TDES0_TCHM</i>	the second address chained mode
<i>ENET_TDES0_TERM</i>	transmit end of ring mode
<i>ENET_TDES0_TTSEN</i>	transmit timestamp function enable
<i>ENET_TDES0_DPAD</i>	disable adding pad
<i>ENET_TDES0_DCRC</i>	disable CRC

<i>ENET_TDES0_FSG</i>	first segment
<i>ENET_TDES0_LSG</i>	last segment
<i>ENET_TDES0_INTC</i>	interrupt on completion
<i>ENET_TDES0_DAV</i>	DAV bit
When type of parameter <b>desc</b> is RX	
<i>ENET_RDES0_DAV</i>	descriptor available
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear VLAN frame bit flag of ENET DMA descriptor */
```

```
enet_desc_flag_clear(p_txdesc, ENET_TDES0_VFRM);
```

#### **enet\_rx\_desc\_immediate\_receive\_complete\_interrupt**

The description of **enet\_rx\_desc\_immediate\_receive\_complete\_interrupt** is shown as below:

**Table 3-253. Function enet\_rx\_desc\_immediate\_receive\_complete\_interrupt**

<b>Function name</b>	enet_rx_desc_immediate_receive_complete_interrupt
<b>Function prototype</b>	void enet_rx_desc_immediate_receive_complete_interrupt(enet_descriptors_struct *desc);
<b>Function descriptions</b>	when receiving completed, set RS bit in ENET_DMA_STAT register will immediately set
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>desc</b>	the descriptor pointer which users want to get flag, the structure members can refer to <a href="#">Structure enet_descriptors_struct</a>
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

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 3-254. 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 be set after a configurable delay time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>desc</b>	the descriptor pointer which users want to get flag, the structure members can refer to <a href="#">Structure enet_descriptors_struct</a>
<b>Input parameter{in}</b>	
<b>delay_time</b>	delay a time of 256*delay_time HCLK(0x00000000 - 0x000000FF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* when receiving completed, RS bit in ENET_DMA_STAT register will be set after 256*16
HCLK */
enet_rx_desc_delay_receive_complete_interrupt(p_rxdesc, 0x00000010);
```

### **enet\_rxframe\_drop**

The description of enet\_rxframe\_drop is shown as below:

**Table 3-255. 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 3-256. 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
<b>ENET_NO_FLUSH_RX</b>	RxDMA does not flushes frames function

<b>FRAME</b>	
<i>ENET_SECONDFRAM E_OPT</i>	TxDMA controller operate on second frame function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-257. Function enet\_dma\_feature\_disable**

<b>Function name</b>	enet_dma_feature_disable
<b>Function prototype</b>	void enet_dma_feature_disable(uint32_t feature);
<b>Function descriptions</b>	disable 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
<i>ENET_NO_FLUSH_RX FRAME</i>	RxDMA does not flushes frames function
<i>ENET_SECONDFRAM E_OPT</i>	TxDMA controller operate on second frame function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-258. Function enet\_rx\_desc\_enhanced\_status\_get**

<b>Function name</b>	enet_rx_desc_enhanced_status_get
<b>Function prototype</b>	uint32_t enet_rx_desc_enhanced_status_get(enet_descriptors_struct *desc, uint32_t desc_status);
<b>Function descriptions</b>	get the bit of extended status flag in 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 the extended status flag, the structure members can refer to <a href="#">Structure enet descriptors struct</a>
<b>Input parameter{in}</b>	
<b>desc_status</b>	desc_status: the extended status want to get only one parameter can be selected which is shown as below
<i>ENET_RDES4_IPPLDT</i>	IP frame payload type
<i>ENET_RDES4_IPHER_R</i>	IP frame header error
<i>ENET_RDES4_IPPLDE_RR</i>	IP frame payload error
<i>ENET_RDES4_IPCKS_B</i>	IP frame checksum bypassed
<i>ENET_RDES4_IPF4</i>	IP frame in version 4
<i>ENET_RDES4_IPF6</i>	IP frame in version 6
<i>ENET_RDES4_PTPMT</i>	PTP message type
<i>ENET_RDES4_PTPOE_F</i>	PTP on ethernet frame
<i>ENET_RDES4_PTPVF</i>	PTP version format
<b>Output parameter{out}</b>	

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

Example:

```
/* get the IP frame payload type in ENET DMA descriptor */

uint32_t status;

status = enet_rx_desc_enhanced_status_get(p_rxdesc, ENET_RDES4_IPPLDT);
```

### **enet\_desc\_select\_enhanced\_mode**

The description of enet\_desc\_select\_enhanced\_mode is shown as below:

**Table 3-259. Function enet\_desc\_select\_enhanced\_mode**

<b>Function name</b>	enet_desc_select_enhanced_mode
<b>Function prototype</b>	void enet_desc_select_enhanced_mode(void);
<b>Function descriptions</b>	configure descriptor to work in enhanced mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure descriptor to work in enhanced mode */

enet_desc_select_enhanced_mode();
```

### **enet\_ptp\_enhanced\_descriptors\_chain\_init**

The description of enet\_ptp\_enhanced\_descriptors\_chain\_init is shown as below:

**Table 3-260. 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
<i>ENET_DMA_TX</i>	DMA Tx descriptors
<i>ENET_DMA_RX</i>	DMA Rx descriptors
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the DMA Tx 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 3-261. 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 3-262. 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 3-263. 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 3-264. 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 3-265. 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>	-

Input parameter{in}	
<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
Input parameter{in}	
<b>desc_ptptab</b>	pointer to the first descriptor address of PTP Rx descriptor table, the structure members can refer to <a href="#"><u>Structure enet_descriptors_struct</u></a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the DMA Rx descriptors's parameters in normal chain mode with PTP function */
enet_descriptors_struct ptp_rxdesc_tab[ENET_RXBUF_NUM];
enet_ptp_normal_descriptors_chain_init(ENET_DMA_RX, ptp_rxdesc_tab);
```

### **enet\_ptp\_normal\_descriptors\_ring\_init**

The description of **enet\_ptp\_normal\_descriptors\_ring\_init** is shown as below:

**Table 3-266. 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>	-
Input parameter{in}	
<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>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="#"><u>Structure enet_descriptors_struct</u></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 3-267. 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 3-268. 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 3-269. 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 3-270. 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 3-271. 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
<i>ENET_WUM_POWER_DOWN</i>	power down mode
<i>ENET_WUM_MAGIC_PACKET_FRAME</i>	enable a wakeup event due to magic packet reception
<i>ENET_WUM_WAKE_U</i>	enable a wakeup event due to wakeup frame reception

<i>P_FRAME</i>	
<i>ENET_WUM_GLOBAL_UNICAST</i>	any received unicast frame passed filter is considered to be a wakeup frame
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* 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 3-272. Function enet\_wum\_feature\_disable**

<b>Function name</b>	enet_wum_feature_disable
<b>Function prototype</b>	void enet_wum_feature_disable(uint32_t feature)
<b>Function descriptions</b>	disable wakeup management features
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>feature</b>	one or more parameters can be selected which are shown as below
<i>ENET_WUM_MAGIC_PACKET_FRAME</i>	enable a wakeup event due to magic packet reception
<i>ENET_WUM_WAKE_UP_FRAME</i>	enable a wakeup event due to wakeup frame reception
<i>ENET_WUM_GLOBAL_UNICAST</i>	any received unicast frame passed filter is considered to be a wakeup frame
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable power down mode */

enet_wum_feature_disable(ENET_WUM_POWER_DOWN);
```

### **enet\_msc\_counters\_reset**

The description of enet\_msc\_counters\_reset is shown as below:

**Table 3-273. Function enet\_msc\_counters\_reset**

<b>Function name</b>	enet_msc_counters_reset
<b>Function prototype</b>	void enet_msc_counters_reset(void);
<b>Function descriptions</b>	reset the MAC statistics counters
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<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 3-274. 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>	-

Input parameter{in}	
<b>feature</b>	one or more parameters can be selected which are shown as below
<i>ENET_MSC_COUNTE R_STOP_ROLLOVER</i>	counter stop rollover
<i>ENET_MSC_RESET_O N_READ</i>	reset on read
<i>ENET_MSC_COUNTE RS_FREEZE</i>	MSC counter freeze
Output parameter{out}	
-	-
Return value	
-	-

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 3-275. 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>	-
Input parameter{in}	
<b>feature</b>	one or more parameters can be selected which are shown as below
<i>ENET_MSC_COUNTE R_STOP_ROLLOVER</i>	counter stop rollover
<i>ENET_MSC_RESET_O N_READ</i>	reset on read
<i>ENET_MSC_COUNTE</i>	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 3-276. 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 <code>enet_msc_preset_enum</code> 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 3-277. Function enet\_msc\_counters\_get**

<b>Function name</b>	enet_msc_counters_get
<b>Function prototype</b>	uint32_t enet_msc_counters_get(enet_msc_counter_enum counter);
<b>Function descriptions</b>	get MAC statistics counter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>counter</b>	MSC counters which is selected only one parameter can be selected which is shown as below
<i>ENET_MSC_TX_SCCN T</i>	MSC transmitted good frames after a single collision counter
<i>ENET_MSC_TX_MSC CNT</i>	MSC transmitted good frames after more than a single collision counter
<i>ENET_MSC_TX_TGFC NT</i>	MSC transmitted good frames counter
<i>ENET_MSC_RX_RFCE CNT</i>	MSC received frames with CRC error counter
<i>ENET_MSC_RX_RFAE CNT</i>	MSC received frames with alignment error counter
<i>ENET_MSC_RX_RGU FCNT</i>	MSC received good unicast frames counter
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	the MSC counter value

Example:

```

/* get MSC transmitted good frames after a single collision counter value*/

```

```

uin32_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 3-278. 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 */

uin32_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 3-279. 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>	-

Input parameter{in}	
<b>nanosecond</b>	Nanosecond value
Output parameter{out}	
-	-
Return value	
<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 3-280. 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_INT</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>NAPSHOT</i>	
<i>ENET_IPV4_FRAME_SNAPSHOT</i>	take snapshot for IPv4 frame
<i>ENET_PTP_FRAME_USE_MACADDRESS_FILTER</i>	use MAC address1-3 to filter the PTP frame
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable PTP function for all received frames */
enet_ptp_feature_enable(ENET_ALL_RX_TIMESTAMP);
```

### **enet\_ptp\_feature\_disable**

The description of enet\_ptp\_feature\_disable is shown as below:

**Table 3-281. Function enet\_ptp\_feature\_disable**

<b>Function name</b>	enet_ptp_feature_disable
<b>Function prototype</b>	void enet_ptp_feature_disable(uint32_t feature);
<b>Function descriptions</b>	disable the PTP features
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>feature</b>	the feature of ENET PTP mode one or more parameters can be selected which are shown as below
<i>ENET_RXTX_TIMESTAMP</i>	timestamp function for transmit and receive frames
<i>ENET_PTP_TIMESTAMP_INT</i>	timestamp interrupt trigger
<i>ENET_ALL_RX_TIMESTAMP</i>	all received frames are taken snapshot

<i>ENET_NONTYPE_FRAME_SNAPSHOT</i>	take snapshot when received non type frame
<i>ENET_IPV6_FRAME_SNAPSHOT</i>	take snapshot for IPv6 frame
<i>ENET_IPV4_FRAME_SNAPSHOT</i>	take snapshot for IPv4 frame
<i>ENET_PTP_FRAME_USE_MACADDRESS_FILTER</i>	use MAC address1-3 to filter the PTP frame
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable PTP function for all received frames */
enet_ptp_feature_disable(ENET_ALL_RX_TIMESTAMP);
```

#### **enet\_ptp\_timestamp\_function\_config**

The description of `enet_ptp_timestamp_function_config` is shown as below:

**Table 3-282. Function `enet_ptp_timestamp_function_config`**

<b>Function name</b>	<code>enet_ptp_timestamp_function_config</code>
<b>Function prototype</b>	<code>ErrStatus enet_ptp_timestamp_function_config(enet_ptp_function_enum func);</code>
<b>Function descriptions</b>	configure the PTP timestamp function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>func</b>	only one parameter can be selected which is shown as below
<i>ENET_CKNT_ORDINARY</i>	type of ordinary clock node type for timestamp
<i>ENET_CKNT_BOUNDARY</i>	type of boundary clock node type for timestamp

<i>ENET_CKNT_END_TO_END</i>	type of end-to-end transparent clock node type for timestamp
<i>ENET_CKNT_PEER_TO_PEER</i>	type of peer-to-peer transparent clock node type for timestamp
<i>ENET_PTP_ADDEND_UPDATE</i>	addend register update
<i>ENET_PTP_SYSTIME_UPDATE</i>	timestamp update
<i>ENET_PTP_SYSTIME_INIT</i>	timestamp initialize
<i>ENET_PTP_FINE_MODE_E</i>	the system timestamp uses the fine method for updating
<i>ENET_PTP_COARSE_MODE</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_PT_P_VERSION_2</i>	version 2
<i>ENET_SNOOPING_PT_P_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_SNAPS_HOT</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 rval = ERROR;

rval = 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 3-283. Function enet\_ptp\_subsecond\_increment\_config**

<b>Function name</b>	enet_ptp_subsecond_increment_config
<b>Function prototype</b>	void enet_ptp_subsecond_increment_config(uint32_t subsecond);
<b>Function descriptions</b>	configure system time subsecond increment value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>subsecond</b>	the value will be added to the subsecond value of system time, this value must be between 0 and 0xFF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure 0x1F as system time subsecond increment value */

enet_ptp_subsecond_increment_config(0x1F);
```

### **enet\_ptp\_timestamp\_addend\_config**

The description of **enet\_ptp\_timestamp\_addend\_config** is shown as below:

**Table 3-284. Function enet\_ptp\_timestamp\_addend\_config**

<b>Function name</b>	enet_ptp_timestamp_addend_config
<b>Function prototype</b>	void enet_ptp_timestamp_addend_config(uint32_t add);
<b>Function descriptions</b>	adjusting the clock frequency only in fine update mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>add</b>	the value will be added to the accumulator register to achieve time synchronization (0 – 0xFFFF FFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

```
/* added 0x1FFF to the accumulator register */
```

```
enet_ptp_timestamp_addend_config(0x1FFF);
```

### **enet\_ptp\_timestamp\_update\_config**

The description of enet\_ptp\_timestamp\_update\_config is shown as below:

**Table 3-285. Function enet\_ptp\_timestamp\_update\_config**

<b>Function name</b>	enet_ptp_timestamp_update_config
<b>Function prototype</b>	void enet_ptp_timestamp_update_config(uint32_t sign, uint32_t second, uint32_t subsecond);
<b>Function descriptions</b>	initialize or add/subtract to second of the system time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sign</b>	timestamp update positive or negative sign, only one parameter can be selected which is shown as below
<b>ENET_PTP_ADD_TO_TIME</b>	update value is added to system time
<b>ENET_PTP_SUBSTRA</b>	timestamp update value is subtracted from system time

<i>CT_FROM_TIME</i>	
<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 3-286. Function enet\_ptp\_expected\_time\_config**

<b>Function name</b>	enet_ptp_expected_time_config
<b>Function prototype</b>	void enet_ptp_expected_time_config(uint32_t second, uint32_t nanosecond);
<b>Function descriptions</b>	configure the expected target time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>second</b>	the expected target second time (0 – 0xFFFF FFFF)
<b>Input parameter{in}</b>	
<b>nanosecond</b>	the expected target nanosecond time (signed) (0 – 0xFFFF FFFF)
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* configure the expected target time */
enet_ptp_expected_time_config(2000, 0);
```

### **enet\_ptp\_system\_time\_get**

The description of enet\_ptp\_system\_time\_get is shown as below:

**Table 3-287. 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="#">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 3-288. enet\_ptp\_pps\_output\_frequency\_config**

<b>Function name</b>	enet_ptp_pps_output_frequency_config
<b>Function prototype</b>	void enet_ptp_pps_output_frequency_config(uint32_t freq);
<b>Function descriptions</b>	configure the PPS output frequency
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>freq</b>	
<i>ENET_PPSOFC_1HZ</i>	PPS output 1Hz frequency
<i>ENET_PPSOFC_2HZ</i>	PPS output 2Hz frequency
<i>ENET_PPSOFC_4HZ</i>	PPS output 4Hz frequency
<i>ENET_PPSOFC_8HZ</i>	PPS output 8Hz frequency
<i>ENET_PPSOFC_16HZ</i>	PPS output 16Hz frequency
<i>ENET_PPSOFC_32HZ</i>	PPS output 32Hz frequency
<i>ENET_PPSOFC_64HZ</i>	PPS output 64Hz frequency
<i>ENET_PPSOFC_128HZ</i>	PPS output 128Hz frequency
<i>ENET_PPSOFC_256HZ</i>	PPS output 256Hz frequency
<i>ENET_PPSOFC_512HZ</i>	PPS output 512Hz frequency
<i>ENET_PPSOFC_1024HZ</i>	PPS output 1024Hz frequency
<i>ENET_PPSOFC_2048HZ</i>	PPS output 2048Hz frequency
<i>ENET_PPSOFC_4096HZ</i>	PPS output 4096Hz frequency
<i>ENET_PPSOFC_8192HZ</i>	PPS output 8192Hz frequency
<i>ENET_PPSOFC_16384HZ</i>	PPS output 16384Hz frequency

<i>ENET_PPSOFC_32768HZ</i>	PPS output 32768Hz frequency
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PPS output frequency as 1Hz */
enet_ptp_pps_output_frequency_config(ENET_PPSOFC_1HZ);
```

### **enet\_ptp\_start**

The enet\_ptp\_start is shown as below:

**Table 3-289. enet\_ptp\_start**

<b>Function name</b>	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> <i>E</i>	fine correction method
<i>ENET_PTP_COARSE</i> <i>MODE</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 3-290. 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 3-291. 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="#">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 3-292. 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="#">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 3-293. 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
<i>ENET_PTP_ADDEND_UPDATE</i>	addend register update
<i>ENET_PTP_SYSTIME_UPDATE</i>	timestamp update
<i>ENET_PTP_SYSTIME_INIT</i>	timestamp initialize
<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 3-294. Function enet\_initpara\_reset**

<b>Function name</b>	enet_initpara_reset
<b>Function prototype</b>	void enet_initpara_reset(void);
<b>Function descriptions</b>	reset the ENET initpara struct, call it before using enet_initpara_config()
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset the ENET initpara struct */
enet_initpara_reset();
```

## 3.11. EXMC

The external memory controller EXMC, is used as a translator for MCU to access a variety of external memory. The EXMC registers are listed in chapter [3.11.1](#), the EXMC firmware functions are introduced in chapter [3.11.2](#).

### 3.11.1. Descriptions of Peripheral registers

EXMC registers are listed in the table shown as below:

**Table 3-295. EXMC Registers**

Registers	Descriptions
EXMC_SNCTL	SRAM/NOR Flash control registers

Registers	Descriptions
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

### 3.11.2. Descriptions of Peripheral functions

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

**Table 3-296. EXMC firmware function**

Function name	Function description
exmc_norsram_deinit	deinitialize EXMC NOR/SRAM regionx
exmc_norsram_struct_para_init	initialize exmc_norsram_parameter_struct with the default values
exmc_norsram_init	initialize EXMC NOR/SRAM regionx
exmc_norsram_enable	enable EXMC NOR/PSRAM regionx
exmc_norsram_disable	disable EXMC NOR/PSRAM regionx
exmc_nand_deinit	deinitialize EXMC NAND bankx
exmc_nand_struct_para_init	initialize exmc_norsram_parameter_struct with the default values
exmc_nand_init	initialize EXMC NAND bankx
exmc_nand_enable	enable EXMC NAND bankx
exmc_nand_disable	disable EXMC NAND bankx
exmc_pccard_deinit	deinitialize EXMC PC card bank
exmc_pccard_struct_para_init	initialize exmc_pccard_parameter_struct with the default values
exmc_pccard_init	initialize EXMC PC card bank

Function name	Function description
exmc_pccard_enable	enable EXMC PC card bank
exmc_pccard_disable	disable EXMC PC card bank
exmc_norsram_page_size_config	configure CRAM page size
exmc_nand_ecc_config	enable or disable the EXMC NAND ECC function
exmc_ecc_get	get the EXMC ECC value
exmc_interrupt_enable	enable EXMC interrupt
exmc_interrupt_disable	disable EXMC interrupt
exmc_flag_get	get EXMC flag status
exmc_flag_clear	clear EXMC flag status
exmc_interrupt_flag_get	get EXMC interrupt flag
exmc_interrupt_flag_clear	clear EXMC interrupt flag

## Structure exmc\_norsram\_timing\_parameter\_struct

Table 3-297. Structure exmc\_norsram\_timing\_parameter\_struct

Member name	Function description
asyn_access_mode	asynchronous access mode
syn_data_latency	configure the data latency
syn_clk_division	configure the clock divide ratio
bus_latency	configure the bus latency
asyn_data_setuptime	configure the data setup time, asynchronous access mode valid
asyn_address_holdtime	configure the address hold time, asynchronous access mode valid
asyn_address_setuptime	configure the data setup time, asynchronous access mode valid

## Structure exmc\_norsram\_parameter\_struct

Table 3-298. Structure exmc\_norsram\_parameter\_struct

Member name	Function description
norsram_region	select the region of EXMC NOR/SRAM

Member name	Function description
write_mode	the write mode, synchronous mode or asynchronous mode
extended_mode	enable or disable the extended mode
asyn_wait	enable or disable the asynchronous wait function
nwait_signal	enable or disable the NWAIT signal while in synchronous burst mode
memory_write	enable or disable the write operation
nwait_config	NWAIT signal configuration
wrap_burst_mode	enable or disable the wrap burst mode
nwait_polarity	specifies the polarity of NWAIT signal from memory
burst_mode	enable or disable the burst mode
databus_width	specifies the databus width of external memory
memory_type	specifies the type of external memory
address_data_mux	specifies whether the data bus and address bus are multiplexed
read_write_timing	timing parameters for read and write if the extended mode is not used or the timing parameters for read if the extended mode is used
write_timing	timing parameters for write when the extended mode is used

## Structure exmc\_nand\_pccard\_timing\_parameter\_struct

**Table 3-299. Structure exmc\_nand\_pccard\_timing\_parameter\_struct**

Member name	Function description
databus_hiztime	configure the dadtabus 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 3-300. 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

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
ecc_logic	enable or disable the ECC calculation logic
databus_width	the NAND flash databus width
wait_feature	enable or disable the wait feature
common_space_timing	the timing parameters for NAND flash common space
attribute_space_timing	the timing parameters for NAND flash attribute space

### Structure exmc\_pccard\_parameter\_struct

Table 3-301. Structure exmc\_pccard\_parameter\_struct

Member name	Function description
atr_latency	configure the latency of ALE low to RB low
ctr_latency	configure the latency of CLE low to RB low
wait_feature	enable or disable the wait feature
common_space_timing	the timing parameters for PC card common space
attribute_space_timing	the timing parameters for PC card attribute space
io_space_timing	the timing parameters for PC card IO space

### exmc\_norsram\_deinit

The description of exmc\_norsram\_deinit is shown as below:

Table 3-302. Function exmc\_norsram\_deinit

Function name	exmc_norsram_deinit
Function prototype	void exmc_norsram_deinit(uint32_t exmc_norsram_region);
Function descriptions	deinitialize EXMC NOR/SRAM region
Precondition	-
The called functions	-

Input parameter{in}	
<b>exmc_norsram_region</b> <b>n</b>	EXMC NOR/SRAM region
<b>EXMC_BANK0_NORS RAM_REGIONx</b>	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 3-303. 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>	-
Input parameter{in}	
<b>exmc_norsram_init_st ruct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-298.</a> <a href="#">Structure exmc_norsram_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the struct nor_init_struct */
```

```
exmc_norsram_parameter_struct nor_init_struct;
exmc_norsram_struct_para_init (&nor_init_struct);
```

### **exmc\_norsram\_init**

The description of exmc\_norsram\_init is shown as below:

**Table 3-304. Function exmc\_norsram\_init**

<b>Function name</b>	exmc_norsram_init
<b>Function prototype</b>	void exmc_norsram_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);
<b>Function descriptions</b>	initialize EXMC NOR/SRAM region
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
exmc_norsram_init_st ruct	Structure for initialization, the structure members can refer to <a href="#">Table 3-298.</a> <a href="#">Structure exmc_norsram parameter struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize EXMC NOR/SRAM bank */

exmc_norsram_parameter_struct lcd_init_struct;
exmc_norsram_timing_parameter_struct lcd_timing_init_struct;

/* configure timing parameter */

lcd_timing_init_struct.asyn_access_mode = EXMC_ACCESS_MODE_A;
lcd_timing_init_struct.syn_data_latency = EXMC_DATALAT_2_CLK;
lcd_timing_init_struct.syn_clk_division = EXMC_SYN_CLOCK_RATIO_DISABLE;
lcd_timing_init_struct.bus_latency = 1;
lcd_timing_init_struct.asyn_data_setuptime = 5;
lcd_timing_init_struct.asyn_address_holdtime = 2;
```

```

lcd_timing_init_struct.asyn_address_setuptime = 2;

/* configure EXMC bus parameters */

lcd_init_struct.norsram_region = EXMC_BANK0_NORSRAM_REGION1;

lcd_init_struct.write_mode = EXMC_ASYN_WRITE;

lcd_init_struct.extended_mode = DISABLE;

lcd_init_struct.asyn_wait = DISABLE;

lcd_init_struct.nwait_signal = DISABLE;

lcd_init_struct.memory_write = ENABLE;

lcd_init_struct.nwait_config = EXMC_NWAIT_CONFIG_BEFORE;

lcd_init_struct.wrap_burst_mode = DISABLE;

lcd_init_struct.nwait_polarity = EXMC_NWAIT_POLARITY_LOW;

lcd_init_struct.burst_mode = DISABLE;

lcd_init_struct.databus_width = EXMC_NOR_DATABUS_WIDTH_16B;

lcd_init_struct.memory_type = EXMC_MEMORY_TYPE_SRAM;

lcd_init_struct.address_data_mux = DISABLE;

lcd_init_struct.read_write_timing = &lcd_timing_init_struct;

lcd_init_struct.write_timing = &lcd_timing_init_struct;

exmc_norsram_init(&lcd_init_struct);

```

### **exmc\_norsram\_enable**

The description of exmc\_norsram\_enable is shown as below:

**Table 3-305. Function exmc\_norsram\_enable**

<b>Function name</b>	exmc_norsram_enable
<b>Function prototype</b>	void exmc_norsram_enable(uint32_t exmc_norsram_region);
<b>Function descriptions</b>	enable EXMC NOR/PSRAM region
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_norsram_region</b>	EXMC NOR/SRAM region

<i>EXMC_BANK0_NORS RAM_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 3-306. 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>	
<i>exmc_norsram_region</i>	EXMC NOR/SRAM region
<i>EXMC_BANK0_NORS RAM_REGIONx</i>	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\_nand\_deinit**

The description of exmc\_nand\_deinit is shown as below:

**Table 3-307. 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 3-308. 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 3-300</a> .

<b>t</b>	<a href="#"><b>Structure exmc_nand_parameter_struct</b></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 3-309. 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> <b>t</b>	Structure for initialization, the structure members can refer to <a href="#"><b>Table 3-300.</b></a> <a href="#"><b>Structure exmc_nand_parameter_struct</b></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 3-310. 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
<b>EXMC_BANKx_NAND</b>	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 3-311. 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
<b>EXMC_BANKx_NAND</b>	x=1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable EXMC NAND bank1 */

exmc_nand_disable(EXMC_BANK1_NAND);
```

### **exmc\_pccard\_deinit**

The description of exmc\_pccard\_deinit is shown as below:

**Table 3-312. Function exmc\_pccard\_deinit**

<b>Function name</b>	exmc_pccard_deinit
<b>Function prototype</b>	void exmc_pccard_deinit(void);
<b>Function descriptions</b>	deinitialize EXMC PC card bank
<b>Precondition</b>	-

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

Example:

```
/* deinitialize EXMC PC card bank */
exmc_pccard_deinit();
```

#### **exmc\_pccard\_struct\_para\_init**

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

**Table 3-313. Function exmc\_pccard\_struct\_para\_init**

<b>Function name</b>	exmc_pccard_struct_para_init
<b>Function prototype</b>	void exmc_pccard_struct_para_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);
<b>Function descriptions</b>	initialize the struct exmc_pccard_parameter_struct
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_pccard_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-301.</a> <a href="#">Structure exmc_pccard_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the struct pccard_init_struct */
exmc_pccard_parameter_struct pccard_init_struct;
```

```
exmc_pccard_struct_para_init (&pccard_init_struct);
```

### **exmc\_pccard\_init**

The description of exmc\_pccard\_init is shown as below:

**Table 3-314. 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 3-301.</a> <a href="#">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 3-315. 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 3-316. 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>	-

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

Example:

```
/* disable EXMC PC card bank */
exmc_pccard_disable();
```

### **exmc\_norsram\_page\_size\_config**

The description of exmc\_norsram\_page\_size\_config is shown as below:

**Table 3-317. Function exmc\_norsram\_page\_size\_config**

Function name	exmc_norsram_page_size_config
Function prototype	void exmc_norsram_page_size_config(uint32_t page_size);
Function descriptions	configure CRAM page size
Precondition	-
The called functions	-
Input parameter{in}	
<b>page_size</b>	CRAM page size
<i>EXMC_CRAM_AUTO_SPLIT</i>	the clock is generated only during synchronous access
<i>EXMC_CRAM_PAGE_SIZE_128_BYTES</i>	page size is 128 bytes
<i>EXMC_CRAM_PAGE_SIZE_256_BYTES</i>	page size is 256 bytes
<i>EXMC_CRAM_PAGE_SIZE_512_BYTES</i>	page size is 512 bytes
<i>EXMC_CRAM_PAGE_SIZE_1024_BYTES</i>	page size is 1024 bytes
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* configure CRAM page size */
exmc_norsram_page_size_config (EXMC_CRAM_PAGE_SIZE_128_BYTES);
```

### **exmc\_nand\_ecc\_config**

The description of exmc\_nand\_ecc\_config is shown as below:

**Table 3-318. Function exmc\_nand\_ecc\_config**

<b>Function name</b>	exmc_nand_ecc_config
<b>Function prototype</b>	void exmc_nand_ecc_config(uint32_t exmc_nand_bank, ControlStatus newvalue);
<b>Function descriptions</b>	enable or disable the EXMC NAND ECC function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_nand_bank</b>	specifie the NAND bank
<b>EXMC_BANKx_NAND</b>	x=1,2
<b>Input parameter{in}</b>	
<b>newvalue</b>	ENABLE or DISABLE
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the EXMC NAND ECC function */
exmc_nand_ecc_config(EXMC_BANK1_NAND, ENABLE);
```

### **exmc\_ecc\_get**

The description of exmc\_ecc\_get is shown as below:

**Table 3-319. 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>	
<b>uint32_t</b>	the error correction code(ECC) value

Example:

```
/* get the EXMC ECC value */

uint32_t ecc_value;

ecc_value = exmc_ecc_get(EXMC_BANK1_NAND);
```

### **exmc\_interrupt\_enable**

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

**Table 3-320. Function exmc\_interrupt\_enable**

<b>Function name</b>	exmc_interrupt_enable
<b>Function prototype</b>	void exmc_interrupt_enable(uint32_t exmc_bank,uint32_t interrupt);
<b>Function descriptions</b>	enable EXMC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>exmc_bank</b>	specifies the NAND bank , PC card bank or SDRAM device
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA RD</i>	the PC Card bank
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify get which interrupt flag
<i>EXMC_NAND_PCCARD_INT_FLAG_RISE</i>	rising edge interrupt and flag
<i>EXMC_NAND_PCCARD_INT_FLAG_LEVEL</i>	high-level interrupt and flag
<i>EXMC_NAND_PCCARD_INT_FLAG_FALL</i>	falling edge interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable EXMC rising edge interrupt*/
exmc_interrupt_enable(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

### **exmc\_interrupt\_disable**

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

**Table 3-321. Function exmc\_interrupt\_disable**

<b>Function name</b>	exmc_interrupt_disable
<b>Function prototype</b>	void exmc_interrupt_disable(uint32_t exmc_bank,uint32_t interrupt);
<b>Function descriptions</b>	disable EXMC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>exmc_bank</b>	specifies the NAND bank , PC card bank or SDRAM device
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA RD</i>	the PC Card bank
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify get which interrupt flag
<i>EXMC_NAND_PCCARD_INT_FLAG_RISE</i>	rising edge interrupt and flag
<i>EXMC_NAND_PCCARD_INT_FLAG_LEVEL</i>	high-level interrupt and flag
<i>EXMC_NAND_PCCARD_INT_FLAG_FALL</i>	falling edge interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable EXMC rising edge interrupt */
exmc_interrupt_disable(EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

### **exmc\_flag\_get**

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

**Table 3-322. Function exmc\_flag\_get**

<b>Function name</b>	exmc_flag_get
<b>Function prototype</b>	FlagStatus exmc_flag_get(uint32_t exmc_bank,uint32_t flag);
<b>Function descriptions</b>	get EXMC flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>exmc_bank</b>	specifies the NAND bank , PC card bank or SDRAM device
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA RD</i>	the PC Card bank
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>EXMC_NAND_PCCAR D_FLAG_RISE</i>	interrupt rising edge status
<i>EXMC_NAND_PCCAR D_FLAG_LEVEL</i>	interrupt high-level status
<i>EXMC_NAND_PCCAR D_FLAG_FALL</i>	interrupt falling edge status
<i>EXMC_NAND_PCCAR 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 3-323. Function exmc\_flag\_clear**

<b>Function name</b>	exmc_flag_clear
<b>Function prototype</b>	FlagStatus exmc_flag_clear (uint32_t exmc_bank,uint32_t flag);
<b>Function descriptions</b>	clear EXMC flag status
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_bank</b>	specifies the NAND bank , PC card bank or SDRAM device
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA RD</i>	the PC Card bank
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>EXMC_NAND_PCCAR D_FLAG_RISE</i>	interrupt rising edge status
<i>EXMC_NAND_PCCAR D_FLAG_LEVEL</i>	interrupt high-level status
<i>EXMC_NAND_PCCAR D_FLAG_FALL</i>	interrupt falling edge status
<i>EXMC_NAND_PCCAR 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 3-324. Function exmc\_interrupt\_flag\_get**

<b>Function name</b>	exmc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus exmc_interrupt_flag_get(uint32_t exmc_bank,uint32_t interrupt);
<b>Function descriptions</b>	get EXMC interrupt flag

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_bank</b>	specifies the NAND bank , PC card bank or SDRAM device
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA RD</i>	the PC Card bank
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify get which interrupt flag
<i>EXMC_NAND_PCCAR D_INT_FLAG_RISE</i>	rising edge interrupt and flag
<i>EXMC_NAND_PCCAR D_INT_FLAG_LEVEL</i>	high-level interrupt and flag
<i>EXMC_NAND_PCCAR D_INT_FLAG_FALL</i>	falling edge interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* check rising edge interrupt flag is set or not*/
if(RESET != exmc_interrupt_flag_get((EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_INT_FLAG_RISE));
```

### **exmc\_interrupt\_flag\_clear**

The description of `exmc_interrupt_flag_clear` is shown as below:

**Table 3-325. Function `exmc_interrupt_flag_clear`**

<b>Function name</b>	<code>exmc_interrupt_flag_clear</code>
<b>Function prototype</b>	<code>void exmc_interrupt_flag_clear(uint32_t exmc_bank,uint32_t interrupt);</code>
<b>Function descriptions</b>	clear EXMC interrupt flag

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_bank</b>	specifies the NAND bank , PC card bank or SDRAM device
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA RD</i>	the PC Card bank
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify get which interrupt flag
<i>EXMC_NAND_PCCAR D_INT_FLAG_RISE</i>	rising edge interrupt and flag
<i>EXMC_NAND_PCCAR D_INT_FLAG_LEVEL</i>	high-level interrupt and flag
<i>EXMC_NAND_PCCAR D_INT_FLAG_FALL</i>	falling edge interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear rising edge interrupt and flag */

exmc_interrupt_flag_clear(EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

## 3.12. EXTI

EXTI is the interrupt / event controller in the MCU. It contains up to 20 independent edge detectors and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter [3.12.1](#), the EXTI firmware functions are introduced in chapter [3.12.2](#).

### 3.12.1. Descriptions of Peripheral registers

EXTI registers are listed in the table shown as below:

**Table 3-326. EXTI Registers**

Registers	Descriptions
EXTI_INTEN	Interrupt enable register
EXTI_EVENT	Event enable register
EXTI_RTEN	Rising edge trigger enable register
EXTI_FTEN	Falling edge trigger enable register
EXTI_SWIEV	Software interrupt event register
EXTI_PD	Pending register

### 3.12.2. Descriptions of Peripheral functions

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

**Table 3-327. EXTI firmware function**

Function name	Function description
exti_deinit	reset EXTI
exti_init	initialize EXTI line x
exti_interrupt_enable	enable EXTI line x interrupt
exti_interrupt_disable	disable EXTI line x interrupt
exti_event_enable	enable EXTI line x event
exti_event_disable	disable EXTI line x event
exti_software_interrupt_enable	enable EXTI line x software interrupt
exti_software_interrupt_disable	disable EXTI line x software interrupt
exti_flag_get	get EXTI line x flag
exti_flag_clear	clear EXTI line x flag
exti_interrupt_flag_get	get EXTI line x interrupt flag
exti_interrupt_flag_clear	clear EXTI line x interrupt flag

### **exti\_deinit**

The description of exti\_deinit is shown as below:

**Table 3-328. Function exti\_deinit**

<b>Function name</b>	exti_deinit
<b>Function prototype</b>	void exti_deinit(void);
<b>Function descriptions</b>	reset EXTI. reset the value of all EXTI registers into initial values
<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 EXTI */
exti_deinit();
```

### **exti\_init**

The description of exti\_init is shown as below:

**Table 3-329. Function exti\_init**

<b>Function name</b>	exti_init
<b>Function prototype</b>	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
<b>Function descriptions</b>	initialize EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x
<b>EXTI_x</b>	x=0,1,2..19

Input parameter{in}	
<b>mode</b>	EXTI mode
<i>EXTI_INTERRUPT</i>	interrupt mode
<i>EXTI_EVENT</i>	event mode
Input parameter{in}	
<b>trig_type</b>	trigger type
<i>EXTI_TRIG_RISING</i>	rising edge trigger
<i>EXTI_TRIG_FALLING</i>	falling edge trigger
<i>EXTI_TRIG_BOTH</i>	rising edge and falling edge trigger
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 3-330. Function exti\_interrupt\_enable**

<b>Function name</b>	exti_interrupt_enable
<b>Function prototype</b>	void exti_interrupt_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable EXTI line x interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>linex</b>	EXTI line x
<i>EXTI_x</i>	x=0,1,2..19
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 3-331. Function exti\_interrupt\_disable**

<b>Function name</b>	exti_interrupt_disable
<b>Function prototype</b>	void exti_interrupt_disable(exti_line_enum linex);
<b>Function descriptions</b>	disable EXTI line x interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x
<i>EXTI_x</i>	x=0,1,2..19
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the interrupts from EXTI line 0 */
exti_interrupt_disable(EXTI_0);
```

### exti\_event\_enable

The description of exti\_event\_enable is shown as below:

**Table 3-332. Function exti\_event\_enable**

<b>Function name</b>	exti_event_enable
----------------------	-------------------

<b>Function prototype</b>	void exti_event_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable EXTI line x event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
linex	EXTI line x
EXTI_x	x=0,1,2..19
<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 3-333. 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 EXTI line x event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
linex	EXTI line x
EXTI_x	x=0,1,2..19
<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 3-334. 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 EXTI line x software interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>linex</i>	EXTI line x
<i>EXTI_x</i>	x=0,1,2..19
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable EXTI line 0 software interrupt */
```

```
exti_software_interrupt_enable(EXTI_0);
```

### **exti\_software\_interrupt\_disable**

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

**Table 3-335. Function exti\_software\_interrupt\_disable**

<b>Function name</b>	exti_software_interrupt_disable
<b>Function prototype</b>	void exti_software_interrupt_disable(exti_line_enum linex);
<b>Function descriptions</b>	disable EXTI line x software interrupt

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x
<i>EXTI_x</i>	x=0,1,2..19
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **exti\_flag\_get**

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

**Table 3-336. Function exti\_flag\_get**

<b>Function name</b>	exti_flag_get
<b>Function prototype</b>	FlagStatus exti_flag_get(exti_line_enum linex);
<b>Function descriptions</b>	get EXTI line x flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x
<i>EXTI_x</i>	x=0,1,2..19
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```

/* get EXTI line 0 flag status */

FlagStatus state = exti_flag_get(EXTI_0);

```

### **exti\_flag\_clear**

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

**Table 3-337. 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 flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
linex	EXTI line x
EXTI_x	x=0,1,2..19
<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 3-338. 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 flag
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
linex	EXTI line x
EXTI_x	x=0,1,2..19
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

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

### **exti\_interrupt\_flag\_clear**

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

**Table 3-339. Function exti\_interrupt\_flag\_clear**

Function name	exti_interrupt_flag_clear
Function prototype	void exti_interrupt_flag_clear(exti_line_enum linex);
Function descriptions	clear EXTI line x interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x
EXTI_x	x=0,1,2..19
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

## 3.13. FMC

There is flash controller and option byte for GD32F30x series. The FMC registers are listed in chapter [3.13.1](#) the FMC firmware functions are introduced in chapter [3.13.2](#).

### 3.13.1. Descriptions of Peripheral registers

FMC registers are listed in the table shown as below:

**Table 3-340. FMC Registers**

Registers	Descriptions
FMC_WS	FMC wait state register
FMC_KEY0	FMC unlock key register 0
FMC_OBKEY	FMC option bytes unlock key register
FMC_STAT0	FMC status register 0
FMC_CTL0	FMC control register 0
FMC_ADDR0	FMC address register 0
FMC_OBSTAT	FMC option bytes status register
FMC_WP	FMC erase/program protection register
FMC_KEY1	FMC unlock key register 1
FMC_STAT1	FMC status register 1
FMC_CTL1	FMC control register 1
FMC_ADDR1	FMC address register 1
FMC_WSEN	FMC wait state enable register
FMC_PID	FMC product ID register

### 3.13.2. Descriptions of Peripheral functions

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

**Table 3-341. FMC firmware function**

Function name	Function description
fmc_wscnt_set	set the FMC wait state counter
fmc_unlock	unlock the main FMC operation

Function name	Function description
fmc_bank0_unlock	unlock the FMC bank0 operation
fmc_bank1_unlock	unlock the FMC bank1 operation
fmc_lock	lock the main FMC operation
fmc_bank0_lock	lock the bank0 FMC operation
fmc_bank1_lock	lock the bank1 FMC operation
fmc_page_erase	FMC erase page
fmc_mass_erase	FMC erase whole chip
fmc_bank0_erase	FMC erase whole bank0
fmc_bank1_erase	FMC erase whole bank1
fmc_word_program	FMC program a word at the corresponding address
fmc_halfword_program	FMC program a half word at the corresponding address
fmc_word_reprogram	FMC reprogram a word at the corresponding address without erasing
ob_unlock	unlock the option byte operation
ob_lock	lock the option byte operation
ob_erase	erase the option byte
ob_write_protection_enable	enable write protect
ob_security_protection_config	configure the option byte security protection
ob_user_write	write the FMC option byte
ob_data_program	program option bytes data
ob_user_get	get the FMC option byte user
ob_data_get	get OB_DATA in register FMC_OBSTAT
ob_write_protection_get	get the FMC option byte write protection
ob_spc_get	get option byte security protection code value
fmc_interrupt_enable	enable FMC interrupt
fmc_interrupt_disable	disable FMC interrupt
fmc_flag_get	check flag is set or not
fmc_flag_clear	clear the FMC flag

Function name	Function description
fmc_interrupt_flag_get	get FMC interrupt flag state
fmc_interrupt_flag_clear	clear FMC interrupt flag state
fmc_bank0_state_get	return the FMC bank0 state
fmc_bank1_state_get	return the FMC bank1 state
fmc_bank0_ready_wait	check FMC bank0 ready or not
fmc_bank1_ready_wait	check FMC bank1 ready or not

## **fmc\_state\_enum**

**Table 3-342. fmc\_state\_enum**

enum name	enum description
FMC_READY	the operation has been completed
FMC_BUSY	the operation is in progress
FMC_PGERR	program error
FMC_WPERR	erase/program protection error
FMC_TOERR	timeout error

## **fmc\_int\_enum**

**Table 3-343. fmc\_int\_enum**

enum name	enum description
FMC_INT_BANK0-END	enable FMC end of program interrupt
FMC_INT_BANK0-ERR	enable FMC error interrupt
FMC_INT_BANK1-END	enable FMC bank1 end of program interrupt
FMC_INT_BANK1-ERR	enable FMC bank1 error interrupt

### fmc\_flag\_enum

**Table 3-344. fmc\_flag\_enum**

enum name	enum description
FMC_FLAG_BANK0_BUSY	FMC bank0 busy flag
FMC_FLAG_BANK0_PGERR	FMC bank0 operation error flag bit
FMC_FLAG_BANK0_WPERR	FMC bank0 erase/program protection error flag bit
FMC_FLAG_BANK0_END	FMC bank0 end of operation flag bit
FMC_FLAG_OBER_R	FMC option bytes read error flag
FMC_FLAG_BANK1_BUSY	FMC bank1 busy flag
FMC_FLAG_BANK1_PGERR	FMC bank1 operation error flag bit
FMC_FLAG_BANK1_WPERR	FMC bank1 erase/program protection error flag bit
FMC_FLAG_BANK1_END	FMC bank1 end of operation flag bit

### fmc\_interrupt\_flag\_enum

**Table 3-345. fmc\_interrupt\_flag\_enum**

enum name	enum description
FMC_INT_FLAG_BANK0_PGERR	FMC bank0 operation error interrupt flag bit
FMC_INT_FLAG_BANK0_WPERR	FMC bank0 erase/program protection error interrupt flag bit
FMC_INT_FLAG_BANK0_END	FMC bank0 end of operation interrupt flag bit
FMC_INT_FLAG_BANK1_PGERR	FMC bank1 operation error interrupt flag bit
FMC_INT_FLAG_BANK1_WPERR	FMC bank1 erase/program protection error interrupt flag bit

enum name	enum description
ANK0_WPERR	
FMC_INT_FLAG_B ANK0_END	FMC bank1 end of operation interrupt flag bit

### fmc\_wscnt\_set

The description of fmc\_wscnt\_set is shown as below:

**Table 3-346. Function fmc\_wscnt\_set**

Function name	fmc_wscnt_set
Function prototype	void fmc_wscnt_set(uint32_t wscnt);
Function descriptions	set the wait state counter value
Precondition	-
The called functions	-
<b>Input parameter{in}</b>	
wscnt	wait state counter value
WS_WSCNT_0	FMC 0 wait
WS_WSCNT_1	FMC 1 wait
WS_WSCNT_2	FMC 2 wait
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-
Function name	fmc_wscnt_set
Function prototype	void fmc_wscnt_set(uint32_t wscnt);
Function descriptions	set the wait state counter value
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
wscnt	wait state counter value
WS_WSCNT_0	FMC 0 wait

<i>WS_WSCNT_1</i>	FMC 1 wait
<i>WS_WSCNT_2</i>	FMC 2 wait
<i>WS_WSCNT_3</i>	FMC 3 wait
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the wait state counter value */
```

```
fmc_wscnt_set (WS_WSCNT_1);
```

<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 3-347. 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 3-348. 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>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable IBUS cache */
```

```
fmc_ibus_enable( );
```

### **fmc\_ibus\_disable**

The description of fmc\_ibus\_disable is shown as below:

**Table 3-349. Function fmc\_ibus\_disable**

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

Example:

```
/* disable IBUS cache */
```

```
fmc_ibus_disable( );
```

### **fmc\_dbus\_enable**

The description of fmc\_dbus\_enable is shown as below:

**Table 3-350. Function fmc\_dbus\_enable**

<b>Function name</b>	fmc_dbus_enable
<b>Function prototype</b>	void fmc_dbus_enable(void);
<b>Function descriptions</b>	enable DBUS cache
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

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

Example:

```
/* enable DBUS cache */

fmc_dbus_enable();
```

### fmc\_dbus\_disable

The description of fmc\_dbus\_disable is shown as below:

**Table 3-351. Function fmc\_dbus\_disable**

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

Example:

```
/* disable DBUS cache */

fmc_dbus_disable();
```

### fmc\_ibus\_reset

The description of fmc\_ibus\_reset is shown as below:

**Table 3-352. Function fmc\_ibus\_reset**

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

Example:

```
/* reset IBUS cache */

fmc_ibus_reset( );
```

### **fmc\_dbus\_reset**

The description of fmc\_dbus\_reset is shown as below:

**Table 3-353. Function fmc\_dbus\_reset**

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

-	-
---	---

Example:

```
/* reset DBUS cache */
```

```
fmc_dbus_reset( );
```

### **fmc\_program\_width\_set**

The description of fmc\_program\_width\_set is shown as below:

**Table 3-354. Function fmc\_program\_width\_set**

<b>Function name</b>	fmc_program_width_set
<b>Function prototype</b>	void fmc_program_width_set(uint32_t pgw);
<b>Function descriptions</b>	set program width to flash memory
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pgw</b>	program width
<i>FMC_PROG_W_32B</i>	32-bit program width to flash memory
<i>FMC_PROG_W_64B</i>	64-bit program width to flash memory
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set program width to flash memory */
```

```
fmc_program_width_set(FMC_PROG_W_32B);
```

### **fmc\_unlock**

The description of fmc\_unlock is shown as below:

**Table 3-355. Function fmc\_unlock**

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

<b>Function descriptions</b>	unlock the main FMC operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* unlock the main FMC operation */
fmc_unlock ( );
```

### **fmc\_bank0\_unlock**

The description of fmc\_bank0\_unlock is shown as below:

**Table 3-356. Function fmc\_bank0\_unlock**

<b>Function name</b>	fmc_bank0_unlock
<b>Function prototype</b>	void fmc_bank0_unlock(void);
<b>Function descriptions</b>	unlock the FMC bank0 operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* unlock the main FMC BANK0 operation */
```

```
fmc_bank0_unlock( );
```

### **fmc\_bank1\_unlock**

The description of fmc\_bank1\_unlock is shown as below:

**Table 3-357. Function fmc\_bank1\_unlock**

<b>Function name</b>	fmc_bank1_unlock
<b>Function prototype</b>	void fmc_bank1_unlock(void);
<b>Function descriptions</b>	unlock the FMC bank1 operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* unlock the main FMC BANK1 operation */

fmc_bank1_unlock( );
```

### **fmc\_lock**

The description of fmc\_lock is shown as below:

**Table 3-358. Function fmc\_lock**

<b>Function name</b>	fmc_lock
<b>Function prototype</b>	void fmc_lock(void);
<b>Function descriptions</b>	lock the main FMC operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
fmc_lock( );
```

### **fmc\_bank0\_lock**

The description of fmc\_bank0\_lock is shown as below:

**Table 3-359. Function fmc\_bank0\_lock**

<b>Function name</b>	fmc_bank0_lock
<b>Function prototype</b>	void fmc_bank0_lock (void);
<b>Function descriptions</b>	lock the FMC bank0 operation
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock the main FMC BANK0 operation */
```

```
fmc_bank0_lock( );
```

### **fmc\_bank1\_lock**

The description of fmc\_bank1\_lock is shown as below:

**Table 3-360. Function fmc\_bank1\_lock**

<b>Function name</b>	fmc_bank1_lock
----------------------	----------------

<b>Function prototype</b>	void fmc_bank1_lock (void);
<b>Function descriptions</b>	lock the FMC bank1 operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* lock the main FMC BANK1 operation */

fmc_bank1_lock( );
```

### fmc\_page\_erase

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

**Table 3-361. Function fmc\_page\_erase**

<b>Function name</b>	fmc_page_erase
<b>Function prototype</b>	fmc_state_enum fmc_page_erase(uint32_t page_address);
<b>Function descriptions</b>	erase page
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_bank0_ready_wait/ fmc_bank1_ready_wait
<b>Input parameter{in}</b>	
page_address	the page address to be erased
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, refer to <a href="#">fmc_state_enum</a>
<b>Function name</b>	fmc_page_erase

<b>Function prototype</b>	fmc_state_enum fmc_page_erase(uint32_t page_address);
<b>Function descriptions</b>	erase page
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
page_address	the page address to be erased
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	<a href="#"><u>state of FMC</u></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 3-362. Function fmc\_mass\_erase**

<b>Function name</b>	fmc_mass_erase
<b>Function prototype</b>	fmc_state_enum fmc_mass_erase(void );
<b>Function descriptions</b>	erase whole chip
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_bank0_ready_wait/ fmc_bank1_ready_wait
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">fmc_state_enum</a>
-----------------------	---

Example:

```
/* erase whole chip */

fmc_unlock();

fmc_state_enum state = fmc_mass_erase ( );
```

### **fmc\_bank0\_erase**

The description of fmc\_bank0\_erase is shown as below:

**Table 3-363. Function fmc\_bank0\_erase**

<b>Function name</b>	fmc_bank0_erase
<b>Function prototype</b>	fmc_state_enum fmc_bank0_erase (void );
<b>Function descriptions</b>	erase bank0
<b>Precondition</b>	fmc_unlock/fmc_bank0_unlock
<b>The called functions</b>	fmc_bank0_ready_wait
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">fmc_state_enum</a>

Example:

```
/* erase bank0 */

fmc_unlock();

fmc_state_enum state = fmc_bank0_erase ( );
```

### **fmc\_bank1\_erase**

The description of fmc\_bank1\_erase is shown as below:

**Table 3-364. Function fmc\_bank1\_erase**

<b>Function name</b>	fmc_bank1_erase
<b>Function prototype</b>	fmc_state_enum fmc_bank1_erase (void );

<b>Function descriptions</b>	erase bank1
<b>Precondition</b>	fmc_unlock/fmc_bank1_unlock
<b>The called functions</b>	fmc_bank1_ready_wait
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#"><i>fmc_state_enum</i></a>

Example:

```
/* erase bank1 */

fmc_unlock();

fmc_state_enum state = fmc_bank1_erase ( );
```

### fmc\_word\_program

The description of fmc\_word\_program is shown as below:

**Table 3-365. Function fmc\_word\_program**

<b>Function name</b>	fmc_word_program
<b>Function prototype</b>	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
<b>Function descriptions</b>	program a word at the corresponding address
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_bank0_ready_wait/ fmc_bank1_ready_wait
<b>Input parameter{in}</b>	
<b>address</b>	the address to program
<b>Input parameter{in}</b>	
<b>data</b>	the data to program
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

<b>fmc_state_enum</b>	state of FMC, refer to <a href="#"><u>fmc_state_enum</u></a>
<b>Function name</b>	fmc_word_program
<b>Function prototype</b>	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
<b>Function descriptions</b>	program a word at the corresponding address
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>address</b>	the address to program
<b>data</b>	the data to program
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	<a href="#"><u>state of FMC</u></a>

Example:

```
/* program a word at the corresponding address */

fmc_unlock();

fmc_page_erase(0x08004000);

fmc_state_enum state = fmc_word_program (0x08004000, 0xaabbccdd);
```

### **fmc\_halfword\_program**

The description of fmc\_halfword\_program is shown as below:

**Table 3-366. Function fmc\_halfword\_program**

<b>Function name</b>	fmc_halfword_program
<b>Function prototype</b>	fmc_state_enum fmc_halfword_program(uint32_t address, uint16_t data);
<b>Function descriptions</b>	program a halfword at the corresponding address
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_bank0_ready_wait/ fmc_bank1_ready_wait
<b>Input parameter{in}</b>	
<b>address</b>	the address to program

Input parameter{in}	
<b>data</b>	the data to program
Output parameter{out}	
-	-
Return value	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">fmc_state_enum</a>

Example:

```

/* program a half word at the corresponding address */

fmc_unlock();

fmc_page_erase(0x08004000);

fmc_state_enum state = fmc_halfword_program (0x08004000, 0xaabb);
  
```

### **fmc\_word\_reprogram**

The description of fmc\_word\_reprogram is shown as below:

**Table 3-367. Function fmc\_word\_reprogram**

<b>Function name</b>	fmc_word_reprogram
<b>Function prototype</b>	fmc_state_enum fmc_word_reprogram(uint32_t address, uint32_t data);
<b>Function descriptions</b>	program a word at the corresponding address without erasing
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_bank0_ready_wait/ fmc_bank1_ready_wait
Input parameter{in}	
<b>address</b>	the address to program
Input parameter{in}	
<b>data</b>	the data to program
Output parameter{out}	
-	-
Return value	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">fmc_state_enum</a>

Example:

---

```

/* program a word at the corresponding address without erasing*/
fmc_unlock();

fmc_state_enum state = fmc_word_reprogram (0x08004000, 0xaabbccdd);

```

### **ob\_unlock**

The description of ob\_unlock is shown as below:

**Table 3-368. Function ob\_unlock**

<b>Function name</b>	ob_unlock
<b>Function prototype</b>	void ob_unlock(void);
<b>Function descriptions</b>	unlock the option byte operation
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* unlock the option byte operation */

fmc_unlock( );

ob_unlock ( );

```

### **ob\_lock**

The description of ob\_lock is shown as below:

**Table 3-369. Function ob\_lock**

<b>Function name</b>	ob_lock
<b>Function prototype</b>	void ob_lock(void);
<b>Function descriptions</b>	lock the option byte operation
<b>Precondition</b>	fmc_unlock

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

Example:

```
/* lock the option byte operation */

fmc_unlock( );
ob_lock ( );
```

### ob\_erase

The description of ob\_erase is shown as below:

**Table 3-370. Function ob\_erase**

<b>Function name</b>	ob_erase
<b>Function prototype</b>	fmc_state_enum ob_erase(void);
<b>Function descriptions</b>	erase the FMC option byte
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_bank0_ready_wait
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">fmc_state_enum</a>
<b>Function name</b>	ob_erase
<b>Function prototype</b>	void ob_erase(void);
<b>Function descriptions</b>	erase the FMC option byte

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

Example:

```
/* erase the FMC option byte */

fmc_unlock( );
ob_unlock( );
fmc_state enum state = ob_erase ( );
```

### **ob\_write\_protection\_enable**

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

**Table 3-371. Function ob\_write\_protection\_enable**

<b>Function name</b>	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>	fmc_bank0_ready_wait
<b>Input parameter{in}</b>	
<b>ob_wp</b>	enable write protection
<b>OB_WP_X</b>	write protect specify sector x
<b>OB_WP_ALL</b>	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="#"><b>fmc_state_enum</b></a>
-----------------------	--

Example:

```
/* enable write protection */

fmc_unlock( );

ob_unlock( );

fmc_state_enum state =
    ob_write_protection_enable (OB_WP7);
```

### **ob\_security\_protection\_config**

The description of `ob_security_protection_config` is shown as below:

**Table 3-372. Function `ob_security_protection_config`**

<b>Function name</b>	ob_security_protection_config
<b>Function prototype</b>	<code>fmc_state_enum ob_security_protection_config (uint8_t ob_spc);</code>
<b>Function descriptions</b>	configure security protection
<b>Precondition</b>	<code>ob_unlock</code>
<b>The called functions</b>	<code>fmc_bank0_ready_wait</code>
<b>Input parameter{in}</b>	
<b>ob_spc</b>	specify security protection
<i>FMC_NSPC</i>	no security protection
<i>FMC_USPC</i>	under security protection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#"><b>fmc_state_enum</b></a>
<b>Function name</b>	ob_security_protection_config
<b>Function prototype</b>	<code>fmc_state_enum ob_security_protection_config (uint8_t ob_spc);</code>
<b>Function descriptions</b>	configure security protection
<b>Precondition</b>	<code>ob_unlock</code>
<b>The called functions</b>	<code>fmc_ready_wait</code>

Input parameter{in}	
<b>ob_spc</b>	specify security protection
<i>FMC_NSPC</i>	no security protection
<i>FMC_USPC</i>	under security protection
Output parameter{out}	
-	-
Return value	
<b>fmc_state_enum</b>	<a href="#"><u>state of FMC</u></a>

Example:

```
/* enable security protection */
ob_unlock();
fmc_state_enum state = ob_security_protection_config (FMC_USPC);
```

### **ob\_user\_write**

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

**Table 3-373. 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_stby, uint8_t ob_boot);
<b>Function descriptions</b>	program the FMC user option byte
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_bank0_ready_wait
Input parameter{in}	
<b>ob_fwdgt</b>	option byte watchdog value
<i>OB_FWDGT_SOFTWA RE</i>	software free watchdog
<i>OB_FWDGT_HARDWA RE</i>	hardware free watchdog
Input parameter{in}	
<b>ob_deepsleep</b>	option byte deepsleep reset value

<i>OB_DEEPSLEEP_NRS_T</i>	no reset when entering deepsleep mode
<i>OB_DEEPSLEEP_RST</i>	generate a reset instead of entering deepsleep mode
<b>Input parameter{in}</b>	
<b>ob_stdby</b>	option byte standby reset value
<i>OB_STDBY_NRST</i>	no reset when entering standby mode
<i>OB_STDBY_RST</i>	generate a reset instead of entering standby mode
<b>Input parameter{in}</b>	
<b>ob_boot</b>	specifies the option byte boot bank value
<i>OB_BOOT_B0</i>	boot from bank0
<i>OB_BOOT_B1</i>	boot from bank1 or bank0 if bank1 is void
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">fmc_state_enum</a>
<b>Function name</b>	<code>ob_user_write</code>
<b>Function prototype</b>	<code>fmc_state_enum ob_user_write(uint8_t ob_fwdgt, uint8_t ob_deepsleep,                                 uint8_t ob_stdby);</code>
<b>Function descriptions</b>	program the FMC user option byte
<b>Precondition</b>	<code>ob_unlock</code>
<b>The called functions</b>	<code>fmc_ready_wait</code>
<b>Input parameter{in}</b>	
<b>ob_fwdgt</b>	option byte watchdog value
<i>OB_FWDGT_SOFTWARE</i>	software free watchdog
<i>OB_FWDGT_HARDWARE</i>	hardware free watchdog
<b>Input parameter{in}</b>	
<b>ob_deepsleep</b>	option byte deepsleep reset value
<i>OB_DEEPSLEEP_NO_</i>	no reset when entering deepsleep mode

<i>RST</i>	
<i>OB_DEEPSLEEP_RST</i>	generate a reset instead of entering deepsleep mode
<b>Input parameter{in}</b>	
<b>ob_stby</b>	option byte standby reset value
<i>OB_STDBY_NO_RST</i>	no reset when entering standby mode
<i>OB_STDBY_RST</i>	generate a reset instead of entering standby mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	<a href="#"><u>state of FMC</u></a>

Example:

```
/* configure user option byte */

fmc_unlock( );

ob_unlock( );

fmc_state_enum state = ob_user_write(OB_FWDGT_HW,OB_DEEPSLEEP_RST,
OB_STDBY_RST);
```

### **ob\_data\_program**

The description of **ob\_data\_program** is shown as below:

**Table 3-374. Function ob\_data\_program**

<b>Function name</b>	ob_data_program
<b>Function prototype</b>	fmc_state_enum ob_data_program(uint32_t address, uint8_t data);
<b>Function descriptions</b>	program option bytes data
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_bank0_ready_wait
<b>Input parameter{in}</b>	
<b>address</b>	the option bytes address to be programmed
<b>Input parameter{in}</b>	
<b>data</b>	the byte to be programmed

Output parameter{out}	
	-
Return value	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">fmc_state_enum</a>

Example:

```
/* program option bytes data */

fmc_unlock( );

ob_unlock( );

fmc_state_enum state =
    ob_data_program (0x1ffff804, 0x56);
```

### ob\_user\_get

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

**Table 3-375. Function ob\_user\_get**

<b>Function name</b>	ob_user_get
<b>Function prototype</b>	uint8_t ob_user_get(void);
<b>Function descriptions</b>	get the FMC user option byte
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
<b>uint8_t</b>	the FMC user option byte values(0xF0 – 0xFF)

Example:

```
/* get the FMC user option byte */

uint8_t user = ob_user_get ( );
```

### **ob\_data\_get**

The description of ob\_data\_get is shown as below:

**Table 3-376. Function ob\_data\_program**

<b>Function name</b>	ob_data_get
<b>Function prototype</b>	uint16_t ob_data_get(void);
<b>Function descriptions</b>	get the FMC data option byte
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>Uint16_t</b>	the FMC data option byte values(0x0 – 0xFFFF)

Example:

```
/* get the FMC data option byte */

uint16_t data = ob_data_get();
```

### **ob\_write\_protection\_get**

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

**Table 3-377. Function ob\_write\_protection\_get**

<b>Function name</b>	ob_write_protection_get
<b>Function prototype</b>	uint32_t ob_write_protection_get(void);
<b>Function descriptions</b>	get the FMC option byte write protection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	

-	-
<b>Return value</b>	
<b>uint32_t</b>	the FMC write protection option byte value(0x0 – 0xFFFFFFFF)

Example:

```
/* get the FMC option byte write protection */
uint32_t wp = ob_write_protection_get();
```

### **ob\_spc\_get**

The description of ob\_spc\_get is shown as below:

**Table 3-378. Function ob\_spc\_get**

<b>Function name</b>	ob_spc_get
<b>Function prototype</b>	FlagStatus ob_spc_get(void);
<b>Function descriptions</b>	get the FMC option byte security protection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the FMC option byte security protection */
FlagStatus spc = ob_spc_get();
```

### **fmc\_interrupt\_enable**

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

**Table 3-379. Function fmc\_interrupt\_enable**

<b>Function name</b>	fmc_interrupt_enable
<b>Function prototype</b>	void fmc_interrupt_enable(uint32_t interrupt);

<b>Function descriptions</b>	enable FMC interrupt
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the FMC interrupt source
<i>FMC_INT_BANK0_EN</i> <i>D</i>	enable FMC bank0 end of program interrupt
<i>FMC_INT_BANK0_ER</i> <i>R</i>	enable FMC bank0 error interrupt
<i>FMC_INT_BANK1_EN</i> <i>D</i>	enable FMC bank1 end of program interrupt
<i>FMC_INT_BANK1_ER</i> <i>R</i>	enable FMC bank1 error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-
<b>Function name</b>	fmc_interrupt_enable
<b>Function prototype</b>	void fmc_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable FMC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the FMC interrupt source
<i>FMC_INT_END</i>	FMC end of program interrupt
<i>FMC_INT_ERR</i>	FMC error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable FMC BANK0 end of program interrupt */

fmc_unlock( );

fmc_interrupt_enable(FMC_INT_BANK0_END);
```

### **fmc\_interrupt\_disable**

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

**Table 3-380. Function fmc\_interrupt\_disable**

<b>Function name</b>	fmc_interrupt_disable
<b>Function prototype</b>	void fmc_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable FMC interrupt
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the FMC interrupt source
<i>FMC_INT_BANK0_EN D</i>	FMC bank0 end of program interrupt
<i>FMC_INT_BANK0_ER R</i>	FMC bank0 error interrupt
<i>FMC_INT_BANK1_EN D</i>	FMC bank1 end of program interrupt
<i>FMC_INT_BANK1_ER R</i>	FMC bank1 error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-
<b>Function name</b>	fmc_interrupt_disable
<b>Function prototype</b>	void fmc_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable FMC interrupt
<b>Precondition</b>	-

<b>The called functions</b>	
<b>Input parameter{in}</b>	
<b>interrupt</b>	the FMC interrupt source
<i>FMC_INT_END</i>	FMC end of program interrupt
<i>FMC_INT_ERR</i>	FMC error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable FMC BANK0 end of program interrupt */

fmc_unlock( );

fmc_interrupt_disable(FMC_INT_BANK0_END);
```

### fmc\_flag\_get

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

**Table 3-381. Function fmc\_flag\_get**

<b>Function name</b>	fmc_flag_get
<b>Function prototype</b>	FlagStatus fmc_flag_get(uint32_t flag);
<b>Function descriptions</b>	check flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	check FMC flag
<i>FMC_FLAG_BANK0_B USY</i>	FMC bank0 busy flag bit
<i>FMC_FLAG_BANK0_P GERR</i>	FMC bank0 operation error flag bit
<i>FMC_FLAG_BANK0_W PERR</i>	FMC bank0 erase/program protection error flag bit

<i>FMC_FLAG_BANK0_E ND</i>	FMC bank0 end of operation flag bit
<i>FMC_FLAG_OBERR</i>	option bytes read error flag bit
<i>FMC_FLAG_BANK1_B USY</i>	FMC bank1 busy flag bit
<i>FMC_FLAG_BANK1_P GERR</i>	FMC bank1 operation error flag bit
<i>FMC_FLAG_BANK1_W PERR</i>	FMC bank1 erase/program protection error flag bit
<i>FMC_FLAG_BANK1_E ND</i>	FMC bank1 end of operation flag bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET
<b>Function name</b>	fmc_flag_get
<b>Function prototype</b>	FlagStatus fmc_flag_get(uint32_t flag);
<b>Function descriptions</b>	check flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	check FMC flag
<i>FMC_FLAG_BUSY</i>	FMC busy flag bit
<i>FMC_FLAG_PGERR</i>	FMC operation error flag bit
<i>FMC_FLAG_PGAERR</i>	FMC program alignment error flag bit
<i>FMC_FLAG_WPERR</i>	FMC erase/program protection error flag bit
<i>FMC_FLAG_END</i>	FMC end of operation flag bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

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

Example:

```
/* get FMC bank0 end of operation flag bit */

FlagStatus flag = fmc_flag_get(FMC_FLAG_BANK0_END);
```

### fmc\_flag\_clear

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

**Table 3-382. 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>	clear FMC flag
<i>FMC_FLAG_BANK0_P GERR</i>	FMC bank0 operation error flag bit
<i>FMC_FLAG_BANK0_W PERR</i>	FMC bank0 erase/program protection error flag bit
<i>FMC_FLAG_BANK0_E ND</i>	FMC bank0 end of operation flag bit
<i>FMC_FLAG_BANK1_P GERR</i>	FMC bank1 operation error flag bit
<i>FMC_FLAG_BANK1_W PERR</i>	FMC bank1 erase/program protection error flag bit
<i>FMC_FLAG_BANK1_E ND</i>	FMC bank1 end of operation flag bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

<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>	clear FMC flag
<i>FMC_FLAG_PGERR</i>	FMC operation error flag bit
<i>FMC_FLAG_PGAERR</i>	FMC program alignment error flag bit
<i>FMC_FLAG_WPERR</i>	FMC erase/program protection error flag bit
<i>FMC_FLAG_END</i>	FMC end of operation flag bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear FMC bank0 operation error flag bit */
FlagStatus flag = fmc_flag_clear(FMC_FLAG_BANK0_PGERR);
```

### **fmc\_interrupt\_flag\_get**

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

**Table 3-383. 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 state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	FMC flag

<i>FMC_INT_FLAG_BAN K0_PGERR</i>	FMC bank0 operation error interrupt flag bit
<i>FMC_INT_FLAG_BAN K0_WPERR</i>	FMC bank0 erase/program protection error interrupt flag bit
<i>FMC_INT_FLAG_BAN K0_END</i>	FMC bank0 end of operation interrupt flag bit
<i>FMC_INT_FLAG_BAN K0_PGERR</i>	FMC bank1 operation error interrupt flag bit
<i>FMC_INT_FLAG_BAN K0_WPERR</i>	FMC bank1 erase/program protection error interrupt flag bit
<i>FMC_INT_FLAG_BAN K0_END</i>	FMC bank1 end of operation interrupt flag bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get FMC bank0 operation error interrupt flag bit */
FlagStatus flag = fmc_interrupt_flag_get (FMC_INT_FLAG_BANK0_PGERR);
```

### **fmc\_interrupt\_flag\_clear**

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

**Table 3-384. Function fmc\_interrupt\_flag\_clear**

<b>Function name</b>	fmc_interrupt_flag_clear
<b>Function prototype</b>	FlagStatus fmc_interrupt_flag_clear (fmc_interrupt_flag_enum flag);
<b>Function descriptions</b>	clear FMC interrupt flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	clear FMC flag
<i>FMC_INT_FLAG_BAN</i>	FMC bank0 operation error interrupt flag bit

<i>K0_PGERR</i>	
<i>FMC_INT_FLAG_BAN</i> <i>K0_WPERR</i>	FMC bank0 erase/program protection error interrupt flag bit
<i>FMC_INT_FLAG_BAN</i> <i>K0_END</i>	FMC bank0 end of operation interrupt flag bit
<i>FMC_INT_FLAG_BAN</i> <i>K0_PGERR</i>	FMC bank1 operation error interrupt flag bit
<i>FMC_INT_FLAG_BAN</i> <i>K0_WPERR</i>	FMC bank1 erase/program protection error interrupt flag bit
<i>FMC_INT_FLAG_BAN</i> <i>K0_END</i>	FMC bank1 end of operation interrupt flag bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* clear FMC bank0 operation error interrupt flag bit */
FlagStatus flag = fmc_interrupt_flag_clear (FMC_INT_FLAG_BANK0_PGERR);
```

#### **fmc\_bank0\_state\_get**

The description of fmc\_bank0\_

state\_get is shown as below:

**Table 3-385. Function fmc\_bank0\_state\_get**

<b>Function name</b>	fmc_bank0_state_get
<b>Function prototype</b>	fmc_state_enum fmc_bank0_state_get(void);
<b>Function descriptions</b>	get the FMC bank0 state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	

-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#"><b>fmc_state_enum</b></a>

Example:

```
/* get the FMC BANK0 state */

fmc_state_enum state = fmc_bank0_state_get();
```

### **fmc\_bank1\_state\_get**

The description of fmc\_bank1\_state\_get is shown as below:

**Table 3-386. Function fmc\_bank1\_state\_get**

<b>Function name</b>	fmc_bank1_state_get
<b>Function prototype</b>	fmc_state_enum fmc_bank1_state_get(void);
<b>Function descriptions</b>	get the FMC bank1 state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#"><b>fmc_state_enum</b></a>

Example:

```
/* get the FMC BANK1 state */

fmc_state_enum state = fmc_bank1_state_get();
```

### **fmc\_bank0\_ready\_wait**

<b>Function name</b>	fmc_state_get
<b>Function prototype</b>	fmc_state_enum fmc_state_get(void);
<b>Function descriptions</b>	get the FMC state
<b>Precondition</b>	-

<b>The called functions</b>		-
<b>Input parameter{in}</b>		
-	-	
<b>Output parameter{out}</b>		
-	-	
<b>Return value</b>		
<b>fmc_state_enum</b>	<a href="#"><u>state of FMC</u></a>	

Example:

```
/* get the FMC state */

fmc_state_enum state = fmc_state_get( );
```

### **fmc\_ready\_wait**

The description of **fmc\_bank0\_ready\_wait** is shown as below:

**Table 3-387. Function fmc\_bank0\_ready\_wait**

<b>Function name</b>	fmc_bank0_ready_wait
<b>Function prototype</b>	fmc_state_enum fmc_bank0_ready_wait(uint32_t timeout);
<b>Function descriptions</b>	check whether FMC bank0 is ready or not
<b>Precondition</b>	-
<b>The called functions</b>	fmc_bank0_state_get
<b>Input parameter{in}</b>	
<b>timeout</b>	count of loop
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#"><u>fmc_state_enum</u></a>
<b>Function name</b>	fmc_ready_wait
<b>Function prototype</b>	fmc_state_enum fmc_ready_wait(uint32_t timeout);
<b>Function descriptions</b>	check whether FMC is ready or not
<b>Precondition</b>	-

<b>The called functions</b>		fmc_state_get()
<b>Input parameter{in}</b>		
<b>timeout</b>	count of loop	
<b>Output parameter{out}</b>		
-	-	
<b>Return value</b>		
<b>fmc_state_enum</b>	<a href="#"><u>state of FMC</u></a>	

Example:

```
/* check whether FMC BANK0 is ready or not */

fmc_state_enum state = fmc_bank0_ready_wait (0x00001000 );
```

### **fmc\_bank1\_ready\_wait**

The description of fmc\_bank1\_ready\_wait is shown as below:

**Table 3-388. Function fmc\_bank1\_ready\_wait**

<b>Function name</b>	fmc_bank1_ready_wait	
<b>Function prototype</b>	fmc_state_enum fmc_bank1_ready_wait(uint32_t timeout);	
<b>Function descriptions</b>	check whether FMC bank1 is ready or not	
<b>Precondition</b>	-	
<b>The called functions</b>	fmc_bank1_state_get	
<b>Input parameter{in}</b>		
<b>timeout</b>	count of loop	
<b>Output parameter{out}</b>		
-	-	
<b>Return value</b>		
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#"><u>fmc state enum</u></a>	

Example:

```
/* check whether FMC BANK1 is ready or not */

fmc_state_enum state = fmc_bank1_ready_wait (0x00001000 );
```

## 3.14. FWDGT

The free watchdog timer (FWDGT) is a hardware timing circuitry that can be used to detect system failures due to software malfunctions. It's suitable for the situation that requires an independent environment and lower timing accuracy. The FWDGT registers are listed in chapter [3.14.1](#) the FWDGT firmware functions are introduced in chapter [3.14.2](#).

### 3.14.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

**Table 3-389. FWDGT Registers**

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

### 3.14.2. Descriptions of Peripheral functions

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

**Table 3-390. FWDGT firmware function**

Function name	Function description
fwdgt_write_enable	enable write access to FWDGT_PSC and FWDGT_RLD
fwdgt_write_disable	disable write access to FWDGT_PSC and FWDGT_RLD
fwdgt_enable	start the free watchdog timer counter
fwdgt_counter_reload	reload the counter of FWDGT
fwdgt_config	configure counter reload value, and prescaler divider value
fwdgt_flag_get	get flag state of FWDGT

#### **fwdgt\_write\_enable**

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

**Table 3-391. Function fwdgt\_write\_enable**

Function name	fwdgt_write_enable

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

Example:

```
/* 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 3-392. Function fwdgt\_write\_disable**

<b>Function name</b>	fwdgt_write_disable
<b>Function prototype</b>	void fwdgt_write_disable(void);
<b>Function descriptions</b>	disable write access to FWDGT_PSC and FWDGT_RLD
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

```
/* disable write access to FWDGT_PSC and FWDGT_RLD */
```

```
fwdgt_write_disable ( );
```

### **fwdgt\_enable**

The description of fwdgt\_enable is shown as below:

**Table 3-393. 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\_counter\_reload**

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

**Table 3-394. Function fwdgt\_counter\_reload**

<b>Function name</b>	fwdgt_counter_reload
<b>Function prototype</b>	void fwdgt_counter_reload(void);
<b>Function descriptions</b>	reload the counter of FWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

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

Example:

```
/* reload FWDGT counter */

fwdgt_counter_reload ( );
```

### fwdgt\_config

The description of fwdgt\_config is shown as below:

**Table 3-395. 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 - 0xFFFF)
<b>Input parameter{in}</b>	
<b>prescaler_div</b>	FWDGT prescaler value-
<i>FWDGT_PSC_DIV4</i>	FWDGT prescaler set to 4
<i>FWDGT_PSC_DIV8</i>	FWDGT prescaler set to 8
<i>FWDGT_PSC_DIV16</i>	FWDGT prescaler set to 16
<i>FWDGT_PSC_DIV32</i>	FWDGT prescaler set to 32
<i>FWDGT_PSC_DIV64</i>	FWDGT prescaler set to 64
<i>FWDGT_PSC_DIV128</i>	FWDGT prescaler set to 128
<i>FWDGT_PSC_DIV256</i>	FWDGT prescaler set to 256
<b>Output parameter{out}</b>	

-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* confiure FWDGT counter clock: 40KHz(IRC40K) / 64 = 0.625 KHz */
fwdgt_config(2*500, FWDGT_PSC_DIV64);
```

### fwdgt\_flag\_get

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

**Table 3-396. Function fwdgt\_flag\_get fwdgt\_write\_disable**

<b>Function name</b>	fwdgt_flag_get
<b>Function prototype</b>	FlagStatus fwdgt_flag_get(uint16_t flag);
<b>Function descriptions</b>	get flag state of FWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flag to get
<b>FWDGT_FLAG_PUD</b>	a write operation to FWDGT_PSC register is on going
<b>FWDGT_FLAG_RUD</b>	a write operation to FWDGT_RLD register is on going
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* test if a prescaler value update is on going */
FlagStatus status;
status = fwdgt_flag_get (FWDGT_FLAG_PUD);
if(status == RESET)
{

```

...

}else

{

...

}

## 3.15. GPIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter [3.15.1 Descriptions of Peripheral 4](#), the GPIO firmware functions are introduced in chapter [3.15.2 Descriptions of Peripheral 5](#).

### 3.15.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

**Table 3-397. GPIO Registers**

Registers	Descriptions
GPIOx_CTL0	Port control register 0
GPIOx_CTL1	Port control register 1
GPIOx_ISTAT	Port input status register
GPIOx_OCTL	Port output control register
GPIOx_BOP	Port bit operate register
GPIOx_BC	Port bit clear register
GPIOx_LOCK	Port configuration lock register
GPIOx_SPD	Port bit speed register
AFIO_EC	Event control register
AFIO_PCF0	AFIO port configuration register 0
AFIO_EXTISSL0	EXTI sources selection register 0
AFIO_EXTISSL1	EXTI sources selection register 1
AFIO_EXTISSL2	EXTI sources selection register 2
AFIO_EXTISSL3	EXTI sources selection register 3
AFIO_PCF1	AFIO port configuration register 1

Registers	Descriptions
AFIO_CPSCTL	IO compensation control register

### 3.15.2. Descriptions of Peripheral functions

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

**Table 3-398. 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_ethernet_phy_select	select ethernet MII or RMII PHY (for GD32F30X_CL devices)
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 3-399. Function gpio\_deinit**

<b>Function name</b>	gpio_deinit
<b>Function prototype</b>	void gpio_deinit(uint32_t gpio_periph);
<b>Function descriptions</b>	reset GPIO port
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
gpio_periph	GPIO port
GPIOx	GPIOx (x=A,B,C,D,E,F,G)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset GPIOA */
gpio_deinit(GPIOA);
```

### gpio\_afio\_deinit

The description of gpio\_afio\_deinit is shown as below:

**Table 3-400. Function gpio\_afio\_deinit**

<b>Function name</b>	gpio_afio_deinit
<b>Function prototype</b>	void gpio_afio_deinit(void);
<b>Function descriptions</b>	reset alternate function I/O(AFIO)
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset alternate function */
gpio_afio_deinit();
```

### gpio\_init

The description of gpio\_init is shown as below:

**Table 3-401. 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
<b>GPIOx</b>	GPIOx (x=A,B,C,D,E,F,G)
<b>Input parameter{in}</b>	
<b>gpio_mode</b>	gpio pin mode
<b>GPIO_MODE_AIN</b>	analog input mode
<b>GPIO_MODE_IN_FLOATING</b>	floating input mode
<b>GPIO_MODE_IPD</b>	pull-down input mode
<b>GPIO_MODE_IPU</b>	pull-up input mode
<b>GPIO_MODE_OUT_OD</b>	GPIO output with open-drain
<b>GPIO_MODE_OUT_PP</b>	GPIO output with push-pull

<i>GPIO_MODE_AF_OD</i>	AFIO output with open-drain
<i>GPIO_MODE_AF_PP</i>	AFIO output with push-pull
<b>Input parameter{in}</b>	
<b>speed</b>	gpio output max speed value
<i>GPIO_OSPEED_10MHZ</i>	output max speed 10MHz
<i>GPIO_OSPEED_2MHZ</i>	output max speed 2MHz
<i>GPIO_OSPEED_50MHZ</i>	output max speed 50MHz
<i>GPIO_OSPEED_MAX</i>	output max speed more than 50MHz
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	<i>GPIO_PIN_x(x=0..15)</i>
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-402. Function gpio\_bit\_set**

<b>Function name</b>	gpio_bit_set
<b>Function prototype</b>	void gpio_bit_set(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	set GPIO pin
<b>Precondition</b>	-

<b>The called functions</b>		-
<b>Input parameter{in}</b>		
<b>gpio_periph</b>		GPIO port
<b>GPIOx</b>		GPIOx (x=A,B,C,D,E,F,G)
<b>Input parameter{in}</b>		
<b>pin</b>		GPIO pin
<b>GPIO_PIN_x</b>		GPIO_PIN_x(x=0..15)
<b>GPIO_PIN_ALL</b>		All pins
<b>Output parameter{out}</b>		
-		-
<b>Return value</b>		
-		-

Example:

```
/* set PA0*/
gpio_bit_set(GPIOA, GPIO_PIN_0);
```

### gpio\_bit\_reset

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

**Table 3-403. 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
<b>GPIOx</b>	GPIOx (x=A,B,C,D,E,F,G)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin

<code>GPIO_PIN_x</code>	<code>GPIO_PIN_x (x=0..15)</code>
<code>GPIO_PIN_ALL</code>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset PA0*/
gpio_bit_set (GPIOA, GPIO_PIN_0);
```

### gpio\_bit\_write

The description of `gpio_bit_write` is shown as below:

**Table 3-404. Function `gpio_bit_write`**

<b>Function name</b>	<code>gpio_bit_write</code>
<b>Function prototype</b>	<code>void gpio_bit_write(uint32_t gpio_periph,uint32_t pin,bit_status bit_value);</code>
<b>Function descriptions</b>	write data to the specified GPIO pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>gpio_periph</code>	GPIO port
<code>GPIOx</code>	<code>GPIOx (x=A,B,C,D,E,F,G)</code>
<b>Input parameter{in}</b>	
<code>pin</code>	GPIO pin
<code>GPIO_PIN_x</code>	<code>GPIO_PIN_x(x=0..15)</code>
<code>GPIO_PIN_ALL</code>	All pins
<b>Input parameter{in}</b>	
<code>bit_value</code>	SET or RESET
<code>RESET</code>	clear the port pin
<code>SET</code>	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 3-405. 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 3-406. Function gpio\_input\_bit\_get**

<b>Function name</b>	gpio_input_bit_get
<b>Function prototype</b>	FlagStatus gpio_input_bit_get(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	get GPIO pin input status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
gpio_periph	GPIO port
GPIOx	GPIOx (x=A,B,C,D,E,F,G)
<b>Input parameter{in}</b>	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
FlagStatus	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 3-407. 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
<b>GPIOx</b>	GPIOx (x=A,B,C,D,E,F,G)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	0x0000-0xFFFF

Example:

```
/* get input value of Port A */
uint16_t port_state;
port_state = gpio_input_bit_get(GPIOA);
```

### gpio\_output\_bit\_get

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

**Table 3-408. Function gpio\_output\_bit\_get**

<b>Function name</b>	gpio_output_bit_get
<b>Function prototype</b>	FlagStatus gpio_output_bit_get(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	get GPIO pin output status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<b>GPIOx</b>	GPIOx (x=A,B,C,D,E,F,G)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<b>GPIO_PIN_x</b>	GPIO_PIN_x (x=0..15)

<code>GPIO_PIN_ALL</code>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<code>FlagStatus</code>	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 3-409. Function `gpio_output_port_get`**

<b>Function name</b>	<code>gpio_output_port_get</code>
<b>Function prototype</b>	<code>uint16_t gpio_output_port_get(uint32_t gpio_periph);</code>
<b>Function descriptions</b>	get GPIO port output status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>gpio_periph</code>	GPIO port
<code>GPIOx</code>	GPIOx (x=A,B,C,D,E,F,G)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<code>Uint16_t</code>	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 3-410. 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
<b>GPIO_SPI0_REMAP</b>	SPI0 remapping
<b>GPIO_I2C0_REMAP</b>	I2C0 remapping
<b>GPIO_USART0_REMAP</b>	USART0 remapping
<b>GPIO_USART1_REMAP</b>	USART1 remapping
<b>GPIO_USART2_PARTIAL_REMAP</b>	USART2 partial remapping
<b>GPIO_USART2_FULL_REMAP</b>	USART2 full remapping
<b>GPIO_TIMER0_PARTIAL_REMAP</b>	TIMER0 partial remapping
<b>GPIO_TIMER0_FULL_REMAP</b>	TIMER0 full remapping
<b>GPIO_TIMER1_PARTIAL_REMAP1</b>	TIMER1 partial remapping
<b>GPIO_TIMER1_PARTIAL_REMAP2</b>	TIMER1 partial remapping
<b>GPIO_TIMER1_FULL_REMAP</b>	TIMER1 full remapping
<b>GPIO_TIMER2_PARTIAL_REMAP</b>	TIMER2 partial remapping

<i>GPIO_TIMER2_FULL_REMAP</i>	TIMER2 full remapping
<i>GPIO_TIMER3_REMAP</i>	TIMER3 remapping
<i>GPIO_CAN_PARTIAL_REMAP</i>	CAN partial remapping (only for GD32F30X_HD devices and GD32F30X_XD devices)
<i>GPIO_CAN_FULL_REMAP</i>	CAN full remapping (only for GD32F30X_HD devices and GD32F30X_XD devices)
<i>GPIO_CAN0_PARTIAL_REMAP</i>	CAN0 partial remapping (only for GD32F30X_CL devices)
<i>GPIO_CAN0_FULL_REMAP</i>	CAN0 full remapping (only for GD32F30X_CL devices)
<i>GPIO_PD01_REMAP</i>	PD01 remapping
<i>GPIO_TIMER4CH3_IREMAPPING</i>	TIMER4 channel3 internal remapping (only for GD32F30X_CL devices and GD32F30X_HD devices)
<i>GPIO_ADC0_ETRGINS_REMAP</i>	ADC0 external trigger inserted conversion remapping (only for GD32F30X_HD devices and GD32F30X_XD devices)
<i>GPIO_ADC0_ETRGREG_REMAP</i>	ADC0 external trigger regular conversion remapping (only for GD32F30X_HD devices and GD32F30X_XD devices)
<i>GPIO_ADC1_ETRGINS_REMAP</i>	ADC1 external trigger inserted conversion remapping (only for GD32F30X_HD devices and GD32F30X_XD devices)
<i>GPIO_ADC1_ETRGREG_REMAP</i>	ADC1 external trigger regular conversion remapping (only for GD32F30X_HD devices and GD32F30X_XD devices)
<i>GPIO_ENET_REMAP</i>	ENET remapping (only for GD32F30X_CL devices)
<i>GPIO_CAN1_REMAP</i>	CAN1 remapping (only for GD32F30X_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 (only for GD32F30X_CL devices)
<i>GPIO_TIMER1ITR0_REMAP</i>	TIMER1 internal trigger 0 remapping (only for GD32F30X_CL devices)

<i>GPIO_PTP_PPS _REMAP</i>	ethernet PTP PPS remapping (only for GD32F30X_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>	
<i>newvalue</i>	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\_ethernet\_phy\_select**

The description of gpio\_ethernet\_phy\_select is shown as below:

**Table 3-411. 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 GD32F30X_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 3-412. 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
<i>GPIO_PORT_SOURCE</i>	output port source (x= A,B,C,D,E)

<code>_GPIOx</code>	
<b>Input parameter{in}</b>	
<code>output_pin</code>	gpio event output pin
<code>GPIO_PIN_SOURCE_x</code>	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 3-413. Function `gpio_event_output_config`**

<b>Function name</b>	<code>gpio_event_output_config</code>
<b>Function prototype</b>	<code>void gpio_event_output_config(uint8_t output_port, uint8_t output_pin);</code>
<b>Function descriptions</b>	configure GPIO pin event output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>output_port</code>	gpio event output port
<code>GPIO_EVENT_PORT_GPIOx</code>	event output port x (x= A,B,C,D,E)
<b>Input parameter{in}</b>	
<code>output_pin</code>	gpio event output pin
<code>GPIO_EVENT_PIN_x</code>	pin number (x=0..15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* Config PA0 as the output of event */

gpio_event_output_config(GPIO_EVENT_PORT_GPIOA, GPIO_EVENT_PIN_0);
```

### gpio\_event\_output\_enable

The description of gpio\_event\_output\_enable is shown as below:

**Table 3-414. Function gpio\_event\_output\_enable**

<b>Function name</b>	gpio_event_output_enable
<b>Function prototype</b>	void gpio_event_output_enable(void);
<b>Function descriptions</b>	enable GPIO pin event output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-415. Function gpio\_event\_output\_disable**

<b>Function name</b>	gpio_event_output_disable
<b>Function prototype</b>	void gpio_event_output_disable(void);
<b>Function descriptions</b>	disable GPIO pin event 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 GPIO pin event output */

gpio_event_output_disable(void);
```

### gpio\_pin\_lock

The description of gpio\_pin\_lock is shown as below:

**Table 3-416. 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>	
gpio_periph	GPIO port
GPIOx	GPIOx (x=A,B,C,D,E,F,G)
<b>Input parameter{in}</b>	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	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 3-417. 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
<b>GPIO_COMPENSATION_ENABLE</b>	I/O compensation cell is enabled
<b>GPIO_COMPENSATION_DISABLE</b>	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 3-418. Function gpio\_compensation\_flag\_get**

<b>Function name</b>	gpio_compensation_flag_get
<b>Function prototype</b>	FlagStatus gpio_compensation_flag_get(void);
<b>Function descriptions</b>	check the I/O compensation cell is ready or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* check the I/O compensation cell state */

FlagStatus cell_state;

cell_state = gpio_compensation_flag_get (void);
```

## 3.16. I2C

The I2C (inter-integrated circuit) module provides an I2C interface which is an industry standard two-line serial interface for MCU to communicate with external I2C interface. The I2C registers are listed in chapter [3.16.1](#), the I2C firmware functions are introduced in chapter [3.16.2](#).

### 3.16.1. Descriptions of Peripheral registers

I2C registers are listed in the table shown as below:

**Table 3-419. I2C Registers**

Registers	Descriptions
I2C_CTL0	Control register 0
I2C_CTL1	Control register 1
I2C_SADDR0	Slave address register 0

Registers	Descriptions
I2C_SADDR1	Slave address register 1
I2C_DATA	Transfer buffer register
I2C_STAT0	Transfer status register 0
I2C_STAT1	Transfer status register 1
I2C_CKCFG	Clock configure register
I2C_RT	Rise time register
I2C_FMPCFG	Fast mode plus configure register

### 3.16.2. Descriptions of Peripheral functions

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

**Table 3-420. I2C firmware function**

Function name	Function description
i2c_deinit	reset I2C
i2c_clock_config	configure I2C clock
i2c_mode_addr_config	configure I2C address
i2c_smbus_type_config	SMBus type selection
i2c_ack_config	whether or not to send an ACK
i2c_ackpos_config	configure I2C POAP position
i2c_master_addressing	master send slave address
i2c_dualaddr_enable	dual-address mode switch
i2c_enable	enable I2C
i2c_disable	disable I2C
i2c_start_on_bus	generate a START condition on I2C bus
i2c_stop_on_bus	generate a STOP condition on I2C bus
i2c_data_transmit	I2C transmit data function
i2c_data_receive	I2C receive data function
i2c_dma_enable	I2C DMA mode enable
i2c_dma_last_transfer_config	configure whether next DMA EOT is DMA last transfer or not

Function name	Function description
i2c_stretch_scl_low_config	whether to stretch SCL low when data is not ready in slave mode
i2c_slave_response_to_gcall_config	whether or not to response to a general call
i2c_software_reset_config	software reset I2C
i2c_pec_enable	I2C PEC calculation on or off
i2c_pec_transfer_enable	I2C whether to transfer PEC value
i2c_pec_value_get	packet error checking value
i2c_smbus_issue_alert	I2C issue alert through SMBA pin
i2c_smbus_arp_enable	I2C ARP protocol in SMBus switch
i2c_flag_get	check I2C flag is set or not
i2c_flag_clear	clear I2C flag
i2c_interrupt_enable	enable I2C interrupt
i2c_interrupt_disable	disable I2C interrupt
i2c_interrupt_flag_get	check I2C interrupt flag
i2c_interrupt_flag_clear	clear I2C interrupt flag

## i2c\_deinit

The description of i2c\_deinit is shown as below:

**Table 3-421. 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
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* reset I2C0 */
i2c_deinit(I2C0);
```

### i2c\_clock\_config

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

**Table 3-422. 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
<b>I2Cx</b>	(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
<b>I2C_DTCY_2</b>	T_low/T_high=2
<b>I2C_DTCY_16_9</b>	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 3-423. 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 3-424. Function i2c\_smbus\_type\_config**

<b>Function name</b>	i2c_smbus_type_config
<b>Function prototype</b>	void i2c_smbus_type_config(uint32_t i2c_periph, uint32_t type);
<b>Function descriptions</b>	SMBus type selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<b>I2Cx</b>	(x=0,1)
<b>Input parameter{in}</b>	
<b>type</b>	Device or host
<b>I2C_SMBUS_DEVICE</b>	device
<b>I2C_SMBUS_HOST</b>	host
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config I2C0 as SMBUS host type*/
i2c_smbus_type_config (I2C0, I2C_SMBUS_HOST);
```

### i2c\_ack\_config

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

**Table 3-425. Function i2c\_ack\_config**

<b>Function name</b>	i2c_ack_config
<b>Function prototype</b>	void i2c_ack_config(uint32_t i2c_periph, uint32_t ack);
<b>Function descriptions</b>	whether or not to send an ACK
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
ack	I2C peripheral
I2C_ACK_ENABLE	ACK will be sent
I2C_ACK_DISABLE	ACK will not be sent
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 will send ACK */
i2c_ack_config (I2C0, I2C_ACK_ENABLE);
```

### i2c\_ackpos\_config

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

**Table 3-426. 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
<b>I2Cx</b>	(x=0,1)
<b>Input parameter{in}</b>	
<b>pos</b>	ACK position
<b>I2C_ACKPOS_CURRENT</b>	whether to send ACK or not for the current
<b>I2C_ACKPOS_NEXT</b>	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 3-427. Function i2c\_master\_addressing**

<b>Function name</b>	i2c_master_addressing
<b>Function prototype</b>	void i2c_master_addressing(uint32_t i2c_periph, uint32_t addr, uint32_t trandirection);
<b>Function descriptions</b>	void i2c_master_addressing(uint32_t i2c_periph, uint32_t addr, uint32_t trandirection);
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral

<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>addr</b>	slave address
<b>Input parameter{in}</b>	
<b>trandirection</b>	transmitter or receiver
<i>I2C_TRANSMITTER</i>	transmitter
<i>I2C_RECEIVER</i>	receiver
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* send slave address to I2C bus and I2C0 act as receiver */

i2c_master_addressing(I2C0, 0x82, I2C_RECEIVER);
```

### i2c\_dualaddr\_enable

The description of i2c\_dualaddr\_enable is shown as below:

**Table 3-428. Function i2c\_dualaddr\_enable**

<b>Function name</b>	i2c_dualaddr_enable
<b>Function prototype</b>	void i2c_dualaddr_enable(uint32_t i2c_periph, uint32_t dualaddr);
<b>Function descriptions</b>	dual-address mode switch
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>dualaddr</b>	Enable or disable
<i>I2C_DUADEN_DISABLE</i>	disable dual-address mode

<i>E</i>	
<i>I2C_DUADDEN_ENABLE</i> <i>E</i>	enable dual-address mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 dual-address*/
i2c_dualaddr_enable (I2C0, I2C_DUADDEN_ENABLE);
```

### i2c\_enable

The description of i2c\_enable is shown as below:

**Table 3-429. Function i2c\_enable**

<b>Function name</b>	i2c_enable
<b>Function prototype</b>	void i2c_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### i2c\_disable

The description of i2c\_disable is shown as below:

**Table 3-430. 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>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### i2c\_start\_on\_bus

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

**Table 3-431. 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>	
i2c_periph	I2C peripheral

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

Example:

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

```
i2c_start_on_bus (I2C0);
```

### **i2c\_stop\_on\_bus**

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

**Table 3-432. 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>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
i2c_stop_on_bus (I2C0);
```

### **i2c\_data\_transmit**

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

**Table 3-433. Function i2c\_data\_transmit**

<b>Function name</b>	i2c_data_transmit
<b>Function prototype</b>	void i2c_data_transmit(uint32_t i2c_periph, uint8_t data);
<b>Function descriptions</b>	I2C transmit data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
data	transmit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 transmit data */
i2c_data_transmit (I2C0);
```

### i2c\_data\_receive

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

**Table 3-434. 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>	
i2c_periph	I2C peripheral

<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	0x00..0xFF

Example:

```
/* I2C0 receive data */

uint8_t i2c_receiver;

i2c_receiver = i2c_data_receive(I2C0);
```

### **i2c\_dma\_enable**

The description of i2c\_dma\_enable is shown as below:

**Table 3-435. Function i2c\_dma\_enable**

<b>Function name</b>	i2c_dma_enable
<b>Function prototype</b>	void i2c_dma_enable(uint32_t i2c_periph, uint32_t dmastate);
<b>Function descriptions</b>	enable I2C DMA mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>dmastate</b>	On or off
<i>I2C_DMA_ON</i>	DMA mode enable
<i>I2C_DMA_OFF</i>	DMA mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 DMA mode enable */

i2c_dma_enable(I2C0, I2C_DMA_ON);
```

### i2c\_dma\_last\_transfer\_enable

The description of i2c\_dma\_last\_transfer\_enable is shown as below:

**Table 3-436. Function i2c\_dma\_last\_transfer\_enable**

<b>Function name</b>	i2c_dma_last_transfer_enable
<b>Function prototype</b>	void i2c_dma_last_transfer_enable(uint32_t i2c_periph, uint32_t dmalast);
<b>Function descriptions</b>	flag indicating DMA last transfer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
dmalast	next DMA EOT is the last transfer or not
I2C_DMALST_ON	next DMA EOT is the last transfer
I2C_DMALST_OFF	next DMA EOT is not the last transfer
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* next DMA EOT is the last transfer */

i2c_dma_last_transfer_enable(I2C0, I2C_DMALST_ON);
```

### i2c\_stretch\_scl\_low\_config

The description of i2c\_stretch\_scl\_low\_config is shown as below:

**Table 3-437. Function i2c\_stretch\_scl\_low\_config**

<b>Function name</b>	i2c_stretch_scl_low_config
<b>Function prototype</b>	void i2c_stretch_scl_low_config(uint32_t i2c_periph, uint32_t stretchpara);
<b>Function descriptions</b>	whether to stretch SCL low when data is not ready in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
stretchpara	SCL stretching enable or disable
I2C_SCLSTRETCH_ENABLE	SCL stretching is enabled
I2C_SCLSTRETCH_DISABLE	SCL stretching is disabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* stretch SCL low when data is not ready in slave mode */
i2c_stretch_scl_low_config (I2C0, I2C_SCLSTRETCH_ENABLE);
```

### **i2c\_slave\_response\_to\_gcall\_config**

The description of i2c\_slave\_response\_to\_gcall\_config is shown as below:

**Table 3-438. Function i2c\_slave\_response\_to\_gcall\_config**

<b>Function name</b>	i2c_slave_response_to_gcall_config
<b>Function prototype</b>	void i2c_slave_response_to_gcall_config(uint32_t i2c_periph, uint32_t gcallpara);
<b>Function descriptions</b>	whether or not to response to a general call

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>gcallpara</b>	response to a general call or not
<i>I2C_GCEN_ENABLE</i>	slave will response to a general call
<i>I2C_GCEN_DISABLE</i>	slave will not response to a general call
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 will response to a general call */
i2c_slave_response_to_gcall_config (I2C0, I2C_GCEN_ENABLE);
```

### **i2c\_software\_reset\_config**

The description of i2c\_software\_reset\_config is shown as below:

**Table 3-439. Function i2c\_software\_reset\_config**

<b>Function name</b>	i2c_software_reset_config
<b>Function prototype</b>	void i2c_software_reset_config(uint32_t i2c_periph, uint32_t sreset);
<b>Function descriptions</b>	software reset I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	

<b>sreset</b>	under reset or not
<i>I2C_SRESET_SET</i>	I2C is under reset
<i>I2C_SRESET_RESET</i>	I2C is not under reset
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* software reset I2C0 */
i2c_software_reset_config (I2C0, I2C_SRESET_SET);
```

### i2c\_pec\_enable

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

**Table 3-440. Function i2c\_pec\_enable**

<b>Function name</b>	i2c_pec_enable
<b>Function prototype</b>	void i2c_pec_enable(uint32_t i2c_periph, uint32_t pecstate);
<b>Function descriptions</b>	I2C PEC calculation on or off
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>pecstate</b>	On or off
<i>I2C_PEC_ENABLE</i>	PEC calculation on
<i>I2C_PEC_DISABLE</i>	PEC calculation off
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* Enable I2C PEC calculation */

i2c_pec_enable (I2C0, I2C_PEC_ENABLE);
```

### i2c\_pec\_transfer\_enable

The description of i2c\_pec\_transfer\_enable is shown as below:

**Table 3-441. Function i2c\_pec\_transfer\_enable**

<b>Function name</b>	i2c_pec_transfer_enable
<b>Function prototype</b>	void i2c_pec_transfer_enable(uint32_t i2c_periph, uint32_t pecpara);
<b>Function descriptions</b>	I2C whether to transfer PEC value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
pecpara	Transfer PEC or not
I2C_PECTRANS_ENA BLE	transfer PEC
I2C_PECTRANS_DISA BLE	not transfer PEC
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 transfer PEC */

i2c_pec_transfer_enable (I2C0, I2C_PECTRANS_ENABLE);
```

### i2c\_pec\_value\_get

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

**Table 3-442. Function i2c\_pec\_value\_get**

<b>Function name</b>	i2c_pec_value_get
<b>Function prototype</b>	uint8_t i2c_pec_value_get(uint32_t i2c_periph);
<b>Function descriptions</b>	get packet error checking value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint8_t	0..255

Example:

```
/* I2C0 get packet error checking value */
uint8_t pec_value;
pec_value = i2c_pec_value_get (I2C0);
```

### i2c\_smbus\_issue\_alert

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

**Table 3-443. Function i2c\_smbus\_issue\_alert**

<b>Function name</b>	i2c_smbus_issue_alert
<b>Function prototype</b>	void i2c_smbus_issue_alert(uint32_t i2c_periph, uint32_t smbuspara);
<b>Function descriptions</b>	I2C issue alert through SMBA pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>smbuspara</b>	issue alert through SMBA pin or not
<i>I2C_SALTSEND_ENAB LE</i>	issue alert through SMBA pin
<i>I2C_SALTSEND_DISA BLE</i>	not issue alert through SMBA pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 issue alert through SMBA pin enable*/
i2c_smbus_issue_alert (I2C0, I2C_SALTSEND_ENABLE);
```

### **i2c\_smbus\_arp\_enable**

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

**Table 3-444. Function i2c\_smbus\_arp\_enable**

<b>Function name</b>	i2c_smbus_arp_enable
<b>Function prototype</b>	void i2c_smbus_arp_enable(uint32_t i2c_periph, uint32_t arpstate);
<b>Function descriptions</b>	enable or disable I2C ARP protocol in SMBus switch
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>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_enable (I2C0, I2C_ARP_ENABLE);
```

### i2c\_flag\_get

The description of i2c\_flag\_get is shown as below:

**Table 3-445. Function i2c\_flag\_get**

<b>Function name</b>	i2c_flag_get
<b>Function prototype</b>	FlagStatus i2c_flag_get(uint32_t i2c_periph,uint32_t flag );
<b>Function descriptions</b>	check I2C flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<i>flag</i>	specify get which flag
<i>I2C_FLAG_SBSEND</i>	start condition send out
<i>I2C_FLAG_ADDSEND</i>	address is sent in master mode or received and matches in slave mode
<i>I2C_FLAG_BTC</i>	byte transmission finishes
<i>I2C_FLAG_ADD10SEN D</i>	header of 10-bit address is sent in master mode
<i>I2C_FLAG_STPDET</i>	stop condition detected in slave mode
<i>I2C_FLAG_RBNE</i>	I2C_DATA is not Empty during receiving

<i>I2C_FLAG_TBE</i>	I2C_DATA is empty during transmitting
<i>I2C_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus
<i>I2C_FLAG_LOSTARB</i>	arbitration lost in master mode
<i>I2C_FLAG_AERR</i>	acknowledge error
<i>I2C_FLAG_OVERRR</i>	overrun or underrun situation occurs in slave mode
<i>I2C_FLAG_PECERR</i>	PEC error when receiving data
<i>I2C_FLAG_SMBTO</i>	timeout signal in SMBus mode
<i>I2C_FLAG_SMBALT</i>	SMBus alert status
<i>I2C_FLAG_MASTER</i>	a flag indicating whether I2C block is in master or slave mode
<i>I2C_FLAG_I2CBSY</i>	busy flag
<i>I2C_FLAG_TRS</i>	whether the I2C is a transmitter or a receiver
<i>I2C_FLAG_RXGC</i>	general call address (00h) received
<i>I2C_FLAG_DEFSMB</i>	default address of SMBus device
<i>I2C_FLAG_HSTSMB</i>	SMBus host header detected in slave mode
<i>I2C_FLAG_DUMOD</i>	dual flag in slave mode indicating which address is matched in dual-address mode
<i>I2C_FLAG_TFF</i>	txframe fall flag
<i>I2C_FLAG_TFR</i>	txframe rise flag
<i>I2C_FLAG_RFF</i>	rxframe fall flag
<i>I2C_FLAG_RFR</i>	rxframe rise flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* check whether start condition send out */

FlagStatus flag_state = RESET;

flag_state = i2c_flag_get (I2C0, I2C_FLAG_SBSEND);
```

### i2c\_flag\_clear

The description of i2c\_flag\_clear is shown as below:

**Table 3-446. Function i2c\_flag\_clear**

<b>Function name</b>	i2c_flag_clear
<b>Function prototype</b>	void i2c_flag_clear(uint32_t i2c_periph, uint32_t flag);
<b>Function descriptions</b>	clear I2C flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
flag	flag type
I2C_FLAG_SMBALT	SMBus Alert status
I2C_FLAG_SMBTO	timeout signal in SMBus mode
I2C_FLAG_PECERR	PEC error when receiving data
I2C_FLAG_OUERR	over-run or under-run situation occurs in slave mode
I2C_FLAG_AERR	acknowledge error
I2C_FLAG_LOSTARB	arbitration lost in master mode
I2C_FLAG_BERR	a bus error
I2C_FLAG_ADDSEND	cleared by reading I2C_STAT0 and reading I2C_STAT1
I2C_FLAG_TFF	txframe fall flag
I2C_FLAG_TFR	txframe rise flag
I2C_FLAG_RFF	rxframe fall flag
I2C_FLAG_RFR	rxframe rise flag
<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 3-447. Function i2c\_interrupt\_enable**

<b>Function name</b>	i2c_interrupt_enable
<b>Function prototype</b>	void i2c_interrupt_enable(uint32_t i2c_periph, uint32_t inttype);
<b>Function descriptions</b>	enable I2C interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<i>inttype</i>	interrupt type
<i>I2C_INT_ERR</i>	error interrupt enable
<i>I2C_INT_EV</i>	event interrupt enable
<i>I2C_INT_BUF</i>	<i>buffer interrupt enable</i>
<i>I2C_INT_TFF</i>	<i>txframe fall interrupt enable</i>
<i>I2C_INT_TFR</i>	<i>txframe rise interrupt enable</i>
<i>I2C_INT_RFF</i>	<i>rxframe fall interrupt enable</i>
<i>I2C_INT_RFR</i>	<i>rxframe rise interrupt enable</i>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 error interrupt */

i2c_interrupt_enable (I2C0, I2C_INT_EV);
```

### i2c\_interrupt\_disable

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

**Table 3-448. Function i2c\_interrupt\_disable**

<b>Function name</b>	i2c_interrupt_disable
<b>Function prototype</b>	void i2c_interrupt_disable(uint32_t i2c_periph, uint32_t inttype);
<b>Function descriptions</b>	disable I2C interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
inttype	interrupt type
I2C_INT_ERR	error interrupt disable
I2C_INT_EV	event interrupt disable
I2C_INT_BUF	buffer interrupt disable
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>
<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 3-449. Function i2c\_interrupt\_flag\_get**

<b>Function name</b>	i2c_interrupt_flag_get
<b>Function prototype</b>	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, uint32_t intflag);
<b>Function descriptions</b>	check I2C interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
intflag	interrupt flag
I2C_INT_FLAG_SBSE_ND	start condition sent out in master mode interrupt flag
I2C_INT_FLAG_ADDS_END	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_ADD10_SEND	header of 10-bit address is sent in master mode interrupt flag
I2C_INT_FLAG_STPD_ET	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_LOSTA_RB	arbitration lost in master mode interrupt flag
I2C_INT_FLAG_AERR	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_SMBTO O</i>	timeout signal in SMBus mode interrupt flag
<i>I2C_INT_FLAG_SMBA LT</i>	SMBus Alert status interrupt flag
<i>I2C_INT_FLAG_TFF</i>	txframe fall interrupt flag
<i>I2C_INT_FLAG_TFR</i>	txframe rise interrupt flag
<i>I2C_INT_FLAG_RFF</i>	rxframe fall interrupt flag
<i>I2C_INT_FLAG_RFR</i>	rxframe rise interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<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 3-450. 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, uint32_t intflag);
<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
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>intflag</b>	interrupt flag
<i>I2C_INT_FLAG_ADDSENDD</i>	address is sent in master mode or received and matches in slave mode interrupt flag
<i>I2C_INT_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag
<i>I2C_INT_FLAG_LOSTARB</i>	arbitration lost in master mode interrupt flag
<i>I2C_INT_FLAG_AERR</i>	acknowledge error interrupt flag
<i>I2C_INT_FLAG_OUERR</i>	over-run or under-run situation occurs in slave mode interrupt flag
<i>I2C_INT_FLAG_PECERR</i>	PEC error when receiving data interrupt flag
<i>I2C_INT_FLAG_SMBT0</i>	timeout signal in SMBus mode interrupt flag
<i>I2C_INT_FLAG_SMBA_LT</i>	SMBus Alert status interrupt flag
<i>I2C_INT_FLAG_TFF</i>	txframe fall interrupt flag
<i>I2C_INT_FLAG_TFR</i>	txframe rise interrupt flag
<i>I2C_INT_FLAG_RFF</i>	rxframe fall interrupt flag
<i>I2C_INT_FLAG_RFR</i>	rxframe rise interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the acknowledge error interrupt flag */
i2c_interrupt_flag_clear (I2C0, I2C_INT_FLAG_AERR);
```

## 3.17. MISC

MISC is a software package that provide the interfaces for NVIC and SysTick. The NVIC and SysTick registers are listed in chapter [3.17.1](#), the MISC firmware functions are introduced in chapter [3.17.2](#).

### 3.17.1. Descriptions of Peripheral registers

**Table 3-451. NVIC Registers**

Registers	Descriptions
ISER <sup>(1)</sup>	Interrupt Set Enable Register
ICER <sup>(1)</sup>	Interrupt Clear Enable Register
ISPR <sup>(1)</sup>	Interrupt Set Pending Register
ICPR <sup>(1)</sup>	Interrupt Clear Pending Register
IABR <sup>(1)</sup>	Interrupt Active bit Register
IP <sup>(1)</sup>	Interrupt Priority Register
STIR <sup>(1)</sup>	Software Trigger Interrupt Register
CPUID <sup>(2)</sup>	CPUID Base Register
ICSR <sup>(2)</sup>	Interrupt Control and State Register
VTOR <sup>(2)</sup>	Vector Table Offset Register
AIRCR <sup>(2)</sup>	Application Interrupt and Reset Control Register
SCR <sup>(2)</sup>	System Control Register
CCR <sup>(2)</sup>	Configuration Control Register
SHP <sup>(2)</sup>	System Handlers Priority Registers
SHCSR <sup>(2)</sup>	System Handler Control and State Register
CFSR <sup>(2)</sup>	Configurable Fault Status Register
HFSR <sup>(2)</sup>	HardFault Status Register
DFSR <sup>(2)</sup>	Debug Fault Status Register
MMFAR <sup>(2)</sup>	MemManage Fault Address Register
BFAR <sup>(2)</sup>	BusFault Address Register
AFSR <sup>(2)</sup>	Auxiliary Fault Status Register

Registers	Descriptions
PFR <sup>(2)</sup>	Processor Feature Register
DFR <sup>(2)</sup>	Debug Feature Register
ADR <sup>(2)</sup>	Auxiliary Feature Register
MMFR <sup>(2)</sup>	Memory Model Feature Register
ISAR <sup>(2)</sup>	Instruction Set Attributes Register
CPACR <sup>(2)</sup>	Coprocessor Access Control Register

1. refer to the structure NVIC\_Type, is defined in the core\_cm4.h file

2. refer to the structure SCB\_Type, is defined in the core\_cm4.h file

**Table 3-452. SysTick Registers**

Registers	Descriptions
CTRL <sup>(1)</sup>	SysTick Control and Status Register
LOAD <sup>(1)</sup>	SysTick Reload Value Register
VAL <sup>(1)</sup>	SysTick Current Value Register
CALIB <sup>(1)</sup>	SysTick Calibration Register

1. refer to the structure SysTick\_Type, is defined in the core\_cm4.h file

### 3.17.2. Descriptions of Peripheral functions

#### Enum IRQn\_Type

**Table 3-453. IRQn\_Type**

Member name	Function description
WWDGT_IRQn	window watchDog timer interrupt
LVD_IRQn	LVD through EXTI line detect interrupt
TAMPER_IRQn	tamper through EXTI line detect
RTC_IRQn	RTC through EXTI line interrupt
FMC_IRQn	FMC interrupt
RCU_CTC_IRQn	RCU and CTC interrupt
EXTI0_IRQn	EXTI line 0 interrupts
EXTI1_IRQn	EXTI line 1 interrupts
EXTI2_IRQn	EXTI line 2 interrupts
EXTI3_IRQn	EXTI line 3 interrupts
EXTI4_IRQn	EXTI line 4 interrupts
DMA0_Channel0_IRQn	DMA0 channel0 interrupt
DMA0_Channel1_IRQn	DMA0 channel1 interrupt

Member name	Function description
DMA0_Channel2_IRQHandler	DMA0 channel2 interrupt
DMA0_Channel3_IRQHandler	DMA0 channel3 interrupt
DMA0_Channel4_IRQHandler	DMA0 channel4 interrupt
DMA0_Channel5_IRQHandler	DMA0 channel5 interrupt
DMA0_Channel6_IRQHandler	DMA0 channel6 interrupt
ADC0_1_IRQHandler	ADC0 and ADC1 interrupt
USBD_HP_CAN0_TX_IRQHandler / CAN0_TX_IRQHandler	USBD high priority and CAN0 transmit interrupts / CAN0 transmit interrupts
USBD_LP_CAN0_RX0_IRQHandler / CAN0_RX0_IRQHandler	USBD low priority and CAN0 receive0 interrupts / CAN0 receive0 interrupts
CAN0_RX1_IRQHandler	CAN0 receive1 interrupts
CAN0_EWMC_IRQHandler	CAN0 EWMC interrupts
EXTI5_9_IRQHandler	EXTI[9:5] interrupts
TIMER0_BRK_TIMER8_IRQHandler	TIMER0 break and TIMER8 interrupts
TIMER0_UP_TIMER9_IRQHandler	TIMER0 update and TIMER9 interrupts
TIMER0_TRG_CMT_TIMER10_IRQHandler	TIMER0 trigger and commutation and TIMER10 interrupts
TIMER0_Channel_IRQHandler	TIMER0 channel capture compare interrupt
TIMER1_IRQHandler	TIMER1 interrupt
TIMER2_IRQHandler	TIMER2 interrupt
TIMER3_IRQHandler	TIMER3 interrupt
I2C0_EV_IRQHandler	I2C0 event interrupt
I2C0_ER_IRQHandler	I2C0 error interrupt
I2C1_EV_IRQHandler	I2C1 event interrupt
I2C1_ER_IRQHandler	I2C1 error interrupt
SPI0_IRQHandler	SPI0 interrupt
SPI1_IRQHandler	SPI1 interrupt
USART0_IRQHandler	USART0 interrupt
USART1_IRQHandler	USART1 interrupt
USART2_IRQHandler	USART2 interrupt
EXTI10_15_IRQHandler	EXTI[15:10] interrupts
RTC_ALARM_IRQHandler	RTC alarm interrupt
USBD_WKUP_IRQHandler / USBFS_WKUP_IRQHandler	USBD wakeup interrupt / USBFS wakeup interrupt
TIMER7_BRK_TIMER11_IRQHandler	TIMER7 break and TIMER11 interrupts
TIMER7_UP_TIMER12_IRQHandler	TIMER7 update and TIMER12 interrupts
TIMER7_TRG_CMT_TIMER13_IRQHandler	TIMER7 trigger and commutation and TIMER13 interrupts
TIMER7_Channel_IRQHandler	TIMER7 channel capture compare interrupt
EXMC_IRQHandler	EXMC global interrupt
SDIO_IRQHandler	SDIO global interrupt
TIMER4_IRQHandler	TIMER4 global interrupt
SPI2_IRQHandler	SPI2 global interrupt

Member name	Function description
UART3_IRQn	UART3 global interrupt
UART4_IRQn	UART4 global interrupt
TIMER5_IRQn	TIMER5 global interrupt
TIMER6_IRQn	TIMER6 global interrupt
DMA1_Channel0_IRQn	DMA1 channel0 global interrupt
DMA1_Channel1_IRQn	DMA1 channel1 global interrupt
DMA1_Channel2_IRQn	DMA1 channel2 global interrupt
DMA1_Channel3_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
CAN1_EWMC_IRQn	CAN1 EWMC interrupt
USBFS_IRQn	USBFS global interrupt

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

**Table 3-454. MISC firmware function**

Function name	Function description
nvic_priority_group_set	set the priority group
nvic_irq_enable	enable NVIC interrupt request
nvic_irq_disable	disable NVIC interrupt request
nvic_vector_table_set	set the NVIC vector table address
system_lowpower_set	set the state of the low power mode
system_lowpower_reset	reset the state of the low power mode
systick_clksource_set	set the systick clock source

### **nvic\_priority\_group\_set**

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

**Table 3-455. Function nvic\_priority\_group\_set**

Function name	nvic_priority_group_set
Function prototype	void nvic_priority_group_set(uint32_t nvic_prigroup);
Function descriptions	configure bits length of the priority group

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>nvic_prigroup</b>	priority group
<i>NVIC_PRIGROUP_PR E0_SUB4</i>	0 bits for pre-emption priority 4 bits for subpriority
<i>NVIC_PRIGROUP_PR E1_SUB3</i>	1 bits for pre-emption priority 3 bits for subpriority
<i>NVIC_PRIGROUP_PR E2_SUB2</i>	2 bits for pre-emption priority 2 bits for subpriority
<i>NVIC_PRIGROUP_PR E3_SUB1</i>	3 bits for pre-emption priority 1 bits for subpriority
<i>NVIC_PRIGROUP_PR E4_SUB0</i>	4 bits for pre-emption priority 0 bits for subpriority
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* priority group configuration , 0 bits for pre-emption priority 4 bits for subpriority */
nvic_priority_group_set(NVIC_PRIGROUP_PRE0_SUB4);
```

### **nvic\_irq\_enable**

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

**Table 3-456. Function nvic\_irq\_enable**

<b>Function name</b>	nvic_irq_enable
<b>Function prototype</b>	void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority);
<b>Function descriptions</b>	enable NVIC request, configure the priority of interrupt
<b>Precondition</b>	-
<b>The called functions</b>	nvic_priority_group_set

Input parameter{in}	
<b>nvic_irq</b>	NVIC interrupt, refer to enum <a href="#">Enum IRQn_Type</a>
Input parameter{in}	
<b>nvic_irq_pre_priority</b>	the pre-emption priority needed to set (0~4)
Input parameter{in}	
<b>nvic_irq_sub_priority</b>	the subpriority needed to set (0~4)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable window watchDog timer interrupt , pre-emption priority is 1, subpriority is 1 */
nvic_irq_enable(WWDGT_IRQn, 1, 1);
```

### **nvic\_irq\_disable**

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

**Table 3-457. Function nvic\_irq\_disable**

<b>Function name</b>	nvic_irq_disable
<b>Function prototype</b>	void nvic_irq_disable(uint8_t nvic_irq);
<b>Function descriptions</b>	disable NVIC request
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>nvic_irq</b>	NVIC interrupt, refer to enum <a href="#">Enum IRQn_Type</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable window watchDog timer interrupt */
```

```
nvic_irq_disable(WWDGT_IRQn);
```

### **nvic\_vector\_table\_set**

The description of nvic\_vector\_table\_set is shown as below:

**Table 3-458. 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_vict_tab, uint32_t offset);
<b>Function descriptions</b>	set the NVIC vector table address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>nvic_vict_tab</b>	the RAM or FLASH base address
<b>NVIC_VECTTAB_RAM</b>	RAM base address
<b>NVIC_VECTTAB_FLASH</b>	Flash base address
<b>Input parameter{in}</b>	
<b>offset</b>	Vector Table offset (vector table start address= base address+offset)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set vector table address = NVIC_VECTTAB_FLASH +0x200 */
```

```
nvic_vector_table_set (NVIC_VECTTAB_FLASH,0x200);
```

### **system\_lowpower\_set**

The description of system\_lowpower\_set is shown as below:

**Table 3-459. 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>	the state of the low power mode management
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lowpower_mode</b>	the low power mode state
<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_P</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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* the system always enter low power mode by exiting from ISR */
system_lowpower_set (SCB_LPM_SLEEP_EXIT_ISR);
```

### system\_lowpower\_reset

The description of system\_lowpower\_reset is shown as below:

**Table 3-460. Function system\_lowpower\_reset**

<b>Function name</b>	system_lowpower_reset
<b>Function prototype</b>	void system_lowpower_reset(uint8_t lowpower_mode);
<b>Function descriptions</b>	the state of the low power mode management
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lowpower_mode</b>	the low power mode state
<i>SCB_LPM_SLEEP_EXIT_ISR</i>	if chose this para, the system will exit low power mode by exiting from ISR

<code>SCB_LPM_DEEPSLEEP_P</code>	if chose this para, the system will enter the SLEEP mode
<code>SCB_LPM_WAKE_BY_ALL_INT</code>	if chose this para, the lowpower mode only can be woke up by the enable interrupts
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* the system will exit low power mode by exiting from ISR */
system_lowpower_reset (SCB_LPM_SLEEP_EXIT_ISR);
```

### **systick\_clksource\_set**

The description of `systick_clksource_set` is shown as below:

**Table 3-461. Function `systick_clksource_set`**

<b>Function name</b>	<code>systick_clksource_set</code>
<b>Function prototype</b>	<code>void systick_clksource_set(uint32_t systick_clksource);</code>
<b>Function descriptions</b>	set the systick clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>systick_clksource</code>	the systick clock source needed to choose
<code>SYSTICK_CLKSOURC_E_HCLK</code>	systick clock source is from HCLK
<code>SYSTICK_CLKSOURC_E_HCLK_DIV8</code>	systick clock source is from HCLK/8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

```

/* systick clock source is HCLK/8 */

systick_clksource_set(SYSTICK_CLKSOURCE_HCLK_DIV8);

```

## 3.18. PMU

According to the Power management unit (PMU), provides three types of power saving modes, including Sleep, Deep-sleep and Standby mode. The PMU registers are listed in chapter [3.18.1](#), the PMU firmware functions are introduced in chapter [3.18.2](#).

### 3.18.1. Descriptions of Peripheral registers

PMU registers are listed in the table shown as below:

**Table 3-462. PMU Registers**

Registers	Descriptions
PMU_CTL	Control register
PMU_CS	Control and status register

### 3.18.2. Descriptions of Peripheral functions

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

**Table 3-463. PMU firmware function**

Function name	Function description
pmu_deinit	deinitialize the PMU
pmu_lvd_select	select low voltage detector threshold
pmu_ldo_output_select	select LDO output voltage
pmu_lvd_disable	disable PMU lvd
pmu_highdriver_switch_select	switch high-driver mode
pmu_highdriver_mode_enable	enable high-driver mode
pmu_highdriver_mode_disable	disable high-driver mode
pmu_lowdriver_mode_enable	enable low-driver mode in deep-sleep mode
pmu_lowdriver_mode_disable	disable low-driver mode in deep-sleep mode
pmu_lowpower_driver_config	in deep-sleep mode, driver mode when use low power LDO
pmu_normalpower_driver_config	in deep-sleep mode, driver mode when use normal power LDO

Function name	Function description
pmu_to_sleepmode	PMU work at sleep mode
pmu_to_deepsleepmode	PMU work at deepsleep mode
pmu_to_standbymode	pmu work at standby mode
pmu_wakeup_pin_enable	enable wakeup pin
pmu_wakeup_pin_disable	disable wakeup pin
pmu_backup_write_enable	enable backup domain write
pmu_backup_write_disable	disable backup domain write
pmu_flag_clear	clear flag bit
pmu_flag_get	get flag state

### pmu\_deinit

The description of pmu\_deinit is shown as below:

**Table 3-464. Function pmu\_deinit**

<b>Function name</b>	pmu_deinit
<b>Function prototype</b>	void pmu_deinit(void);
<b>Function descriptions</b>	deinitialize the PMU
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset PMU */
pmu_deinit();
```

### **pmu\_lvd\_select**

The description of pmu\_lvd\_select is shown as below:

**Table 3-465. Function pmu\_lvd\_select**

<b>Function name</b>	pmu_lvd_select
<b>Function prototype</b>	void pmu_lvd_select(uint32_t lvdt_n);
<b>Function descriptions</b>	select low voltage detector threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>lvdt_n</i>	voltage threshold value
<i>PMU_LVDT_0</i>	voltage threshold is 2.2V
<i>PMU_LVDT_1</i>	voltage threshold is 2.3V
<i>PMU_LVDT_2</i>	voltage threshold is 2.4V
<i>PMU_LVDT_3</i>	voltage threshold is 2.5V
<i>PMU_LVDT_4</i>	voltage threshold is 2.6V
<i>PMU_LVDT_5</i>	voltage threshold is 2.7V
<i>PMU_LVDT_6</i>	voltage threshold is 2.8V
<i>PMU_LVDT_7</i>	voltage threshold is 2.9V
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select low voltage detector threshold as 2.9V */
pmu_lvd_select (PMU_LVDT_7);
```

### **pmu\_ldo\_output\_select**

The description of pmu\_ldo\_output\_select is shown as below:

**Table 3-466. Function pmu\_ldo\_output\_select**

<b>Function name</b>	pmu_ldo_output_select
<b>Function prototype</b>	void pmu_ldo_output_select(uint32_t ldo_output);
<b>Function descriptions</b>	internal voltage regulator (LDO) output voltage select
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ldo_output</b>	output voltage mode
<i>PMU_LDOVS_LOW</i>	output low voltage mode
<i>PMU_LDOVS_MID</i>	LDO output voltage mid mode
<i>PMU_LDOVS_NORMA L</i>	LDO output voltage high mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select output low voltage mode */
pmu_ldo_output_select (PMU_LDOVS_LOW);
```

### **pmu\_lvd\_disable**

The description of pmu\_lvd\_disable is shown as below:

**Table 3-467. Function pmu\_lvd\_disable**

<b>Function name</b>	pmu_lvd_disable
<b>Function prototype</b>	void pmu_lvd_disable (void);
<b>Function descriptions</b>	disable PMU lvd
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable PMU lvd */
pmu_lvd_disable();
```

### pmu\_highdriver\_switch\_select

The description of pmu\_highdriver\_switch\_select is shown as below:

**Table 3-468. Function pmu\_highdriver\_switch\_select**

Function name	pmu_highdriver_switch_select
Function prototype	void pmu_highdriver_switch_select(uint32_t highdr_switch);
Function descriptions	switch high-driver mode
Precondition	-
The called functions	pmu_flag_get()
Input parameter{in}	
highdr_switch	enable or disable high-driver mode switch
PMU_HIGHDR_SWITCH_NONE	disable high-driver mode switch
PMU_HIGHDR_SWITCH_EN	enable high-driver mode switch
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable high-driver mode switch */
pmu_highdriver_switch_select(PMU_HIGHDR_SWITCH_EN);
```

### **pmu\_highdriver\_mode\_enable**

The description of pmu\_highdriver\_mode\_enable is shown as below:

**Table 3-469. 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 3-470. 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\_lowdriver\_mode\_enable**

The description of pmu\_lowdriver\_mode\_enable is shown as below:

**Table 3-471. 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 3-472. 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 3-473. 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
<b>PMU_NORMALDR_LO WPWR</b>	normal driver when use low power LDO
<b>PMU_LOWDR_LOWP WR</b>	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 3-474. 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 3-475. Function pmu\_to\_sleepmode**

<b>Function name</b>	pmu_to_sleepmode
<b>Function prototype</b>	void pmu_to_sleepmode(uint8_t sleepmodecmd);
<b>Function descriptions</b>	PMU work at sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sleepmodecmd</b>	command to enter sleep mode
<b>WFI_CMD</b>	use WFI command
<b>WFE_CMD</b>	use WFE command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work at sleep mode */
pmu_to_sleepmode (WFI_CMD);
```

### **pmu\_to\_deepsleepmode**

The description of pmu\_to\_deepsleepmode is shown as below:

**Table 3-476. Function pmu\_to\_deepsleepmode**

<b>Function name</b>	pmu_to_deepsleepmode
<b>Function prototype</b>	void pmu_to_deepsleepmode(uint32_t ldo,uint8_t deepsleepmodecmd);
<b>Function descriptions</b>	PMU work at deepsleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ldo</b>	ldo work mode
<b>PMU_LDO_NORMAL</b>	LDO normal work when pmu enter deepsleep mode

<b>PMU_LDO_LOWPOW ER</b>	LDO work at low power mode when pmu enter deepsleep mode
<b>Input parameter{in}</b>	
<b>deepsleepmodecmd</b>	command to enter deepsleep mode
<b>WFI_CMD</b>	use WFI command
<b>WFE_CMD</b>	use WFE command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work at deepsleep mode */
pmu_to_deepsleepmode (PMU_LDO_NORMAL, WFI_CMD);
```

### **pmu\_to\_standbymode**

The description of pmu\_to\_standbymode is shown as below:

**Table 3-477. Function pmu\_to\_standbymode**

<b>Function name</b>	pmu_to_standbymode
<b>Function prototype</b>	void pmu_to_standbymode(uint8_t standbymodecmd);
<b>Function descriptions</b>	pmu work at standby mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>standbymodecmd</b>	command to enter standby mode
<b>WFI_CMD</b>	use WFI command
<b>WFE_CMD</b>	use WFE command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* PMU work at standby mode */
pmu_to_standby (WFI_CMD);
```

### **pmu\_backup\_write\_enable**

The description of pmu\_backup\_write\_enable is shown as below:

**Table 3-478. 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 3-479. 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 3-480. Function pmu\_wakeup\_pin\_enable**

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

Example:

```
/* enable wakeup pin */
pmu_wakeup_pin_enable();
```

### **pmu\_wakeup\_pin\_disable**

The description of pmu\_wakeup\_pin\_disable is shown as below:

**Table 3-481. Function pmu\_wakeup\_pin\_disable**

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

Example:

```
/* disable wakeup pin */
pmu_wakeup_pin_disable();
```

### **pmu\_flag\_get**

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

**Table 3-482. 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
<b>PMU_FLAG_WAKEUP</b>	wakeup flag

<i>PMU_FLAG_STANDBY</i>	standby flag
<i>PMU_FLAG_LVD</i>	low voltage detector status flag
<i>PMU_FLAG_LDOVSRF</i>	LDO voltage select ready flag
<i>PMU_FLAG_HDRF</i>	high-driver ready flag
<i>PMU_FLAG_HDSRF</i>	high-driver switch ready flag
<i>PMU_FLAG_LDRF</i>	low-driver mode ready flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get flag state */

FlagStatus status;

status = pmu_flag_get (PMU_FLAG_WAKEUP);
```

### **pmu\_flag\_clear**

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

**Table 3-483. Function pmu\_flag\_clear**

<b>Function name</b>	pmu_flag_clear
<b>Function prototype</b>	void pmu_flag_clear(uint32_t flag_reset);
<b>Function descriptions</b>	clear flag bit
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag_reset</b>	flag
<i>PMU_FLAG_RESET_WAKEUP</i>	reset wakeup flag
<i>PMU_FLAG_RESET_STANDBY</i>	reset standby flag
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* clear flag bit */
pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

## 3.19. 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 [3.19.1](#), the RCU firmware functions are introduced in chapter [3.19.2](#).

### 3.19.1. Descriptions of Peripheral registers

Table 3-484. 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(CL series)
RCU_CFG1	Clock configuration register 1
RCU_DSV	Deep-sleep mode voltage register
RCU_ADDCTL	Additional clock control register

Registers	Descriptions
RCU_ADDINT	Additional clock interrupt register
RCU_ADDAPB1RST	APB1 additional reset register
RCU_ADDAPB1EN	APB1 additional enable register

### 3.19.2. Descriptions of Peripheral functions

Table 3-485. RCU firmware function

Function name	Function description
rcu_deinit	deinitialize the RCU
rcu_periph_clock_enable	enable the peripherals clock
rcu_periph_clock_disable	disable the peripherals clock
rcu_periph_clock_sleep_enable	enable the peripherals clock when in sleep mode
rcu_periph_clock_sleep_disable	disable the peripherals clock when in sleep mode
rcu_periph_reset_enable	enable the peripherals reset
rcu_periph_reset_disable	disable the peripheral reset
rcu_bkp_reset_enable	enable the BKP domain reset
rcu_bkp_reset_disable	disable the BKP domain reset
rcu_system_clock_source_config	configure the system clock source
rcu_system_clock_source_get	get the system clock source
rcu_ahb_clock_config	configure the AHB clock prescaler selection
rcu_apb1_clock_config	configure the APB1 clock prescaler selection
rcu_apb2_clock_config	configure the APB2 clock prescaler selection
rcu_ckout0_config	configure the CK_OUT0 clock source
rcu_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

Function name	Function description
rcu_adc_clock_config	configure the ADC prescaler factor
rcu_usb_clock_config	configure the USB prescaler factor
rcu_rtc_clock_config	configure the RTC clock source selection
rcu_i2s1_clock_config	configure the I2S1 clock source selection
rcu_i2s2_clock_config	configure the I2S2 clock source selection
rcu_ck48m_clock_config	configure the CK48M clock selection
rcu_flag_get	get the clock stabilization and peripheral reset flags
rcu_all_reset_flag_clear	clear all the reset flag
rcu_interrupt_flag_get	get the clock stabilization interrupt and ckm flags
rcu_interrupt_flag_clear	clear the interrupt flags
rcu_interrupt_enable	enable the stabilization interrupt
rcu_interrupt_disable	disable the stabilization interrupt
rcu_lxtal_drive_capability_config	configure the LXTAL drive capability
rcu_osc_stab_wait	wait for oscillator stabilization flags is SET or oscillator startup is timeout
rcu_osc_on	turn on the oscillator
rcu_osc_off	turn off the oscillator
rcu_osc_bypass_mode_enable	enable the oscillator bypass mode
rcu_osc_bypass_mode_disable	disable the oscillator bypass mode
rcu_hxtal_clock_monitor_enable	enable the HXTAL clock monitor
rcu_hxtal_clock_monitor_disable	disable the HXTAL clock monitor
rcu_irc8m_adjust_value_set	set the IRC8M adjust value
rcu_deepsleep_voltage_set	set the deep-sleep mode voltage value
rcu_clock_freq_get	get the system clock, bus clock frequency

## rcu\_deinit

The description of rcu\_deinit is shown as below:

**Table 3-486. Function rcu\_deinit**

<b>Function name</b>	rcu_deinit
<b>Function prototype</b>	void rcu_deinit(void);
<b>Function descriptions</b>	deinitialize the RCU, reset the value of all RCU registers into initial values
<b>Precondition</b>	-
<b>The called functions</b>	rcu_osc_stab_wait
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset RCU */
rcu_deinit();
```

### **rcu\_periph\_clock\_enable**

The description of rcu\_periph\_clock\_enable is shown as below:

**Table 3-487. 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 rcu_periph_enum
<b>RCU_GPIOx</b>	GPIO ports clock (x=A,B,C,D,E,F,G)
<b>RCU_AF</b>	alternate function clock
<b>RCU_CRC</b>	CRC clock

<i>RCU_DMAt</i>	DMAt clock (t=0,1)
<i>RCU_ENET</i>	ENET clock (CL series)
<i>RCU_ENETTX</i>	ENETTX clock (CL series)
<i>RCU_ENETRX</i>	ENETRX clock (CL series)
<i>RCU_USBD</i>	USBD clock (HD、XD series)
<i>RCU_USBFS</i>	USBFS clock (CL series)
<i>RCU_EXMC</i>	EXMC clock
<i>RCU_TIMERt</i>	TIMERt clock (t=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGT</i>	WWDGT clock
<i>RCU_SPIt</i>	SPIt clock (t=0,1,2)
<i>RCU_USARTt</i>	USARTt clock (t=0,1,2)
<i>RCU_UARTt</i>	UARTt clock (t=3,4)
<i>RCU_I2Cx</i>	I2Cx clock (x=0,1)
<i>RCU_CANt</i>	CANt clock (t=0,1) CAN1 is only available for CL series
<i>RCU_PMU</i>	PMU clock
<i>RCU_DAC</i>	DAC clock
<i>RCU_RTC</i>	RTC clock
<i>RCU_ADCt</i>	ADCt clock (t=0,1,2) ADC2 is not available for CL series
<i>RCU_SDIO</i>	SDIO clock (HD、XD series)
<i>RCU_CTC</i>	CTC clock
<i>RCU_BKPI</i>	BKP interface clock
<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 3-488. Function rcu\_periph\_clock\_disable**

<b>Function name</b>	rcu_periph_clock_disable
<b>Function prototype</b>	void rcu_periph_clock_disable(rcu_periph_enum periph);
<b>Function descriptions</b>	disable the peripherals clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to rcu_periph_enum
<i>RCU_GPIOx</i>	GPIO ports clock (x=A,B,C,D,E,F,G)
<i>RCU_AF</i>	alternate function clock
<i>RCU_CRC</i>	CRC clock
<i>RCU_DMAx</i>	DMAx clock (x=0,1)
<i>RCU_ENET</i>	ENET clock (CL series)
<i>RCU_ENETTX</i>	ENETTX clock (CL series)
<i>RCU_ENETRX</i>	ENETRX clock (CL series)
<i>RCU_USBD</i>	USBD clock (HD、XD series)
<i>RCU_USBFS</i>	USBFS clock (CL series)
<i>RCU_EXMC</i>	EXMC clock
<i>RCU_TIMERx</i>	TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGT</i>	WWDGT clock
<i>RCU_SPIx</i>	SPIx clock (x=0,1,2)
<i>RCU_USARTx</i>	USARTx clock (x=0,1,2)
<i>RCU_UARTx</i>	UARTx clock (x=3,4)
<i>RCU_I2Cx</i>	I2Cx clock (x=0,1)
<i>RCU_CANx</i>	CANx clock (x=0,1) CAN1 is only available for CL series
<i>RCU_PMU</i>	PMU clock

<i>RCU_DAC</i>	DAC clock
<i>RCU_RTC</i>	RTC clock
<i>RCU_ADCx</i>	ADCx clock (x=0,1,2) ADC2 is not available for CL series
<i>RCU_SDIO</i>	SDIO clock (HD、XD series)
<i>RCU_CTC</i>	CTC clock
<i>RCU_BKPI</i>	BKP interface clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the USART0 clock */
rcu_periph_clock_disable(RCU_USART0);
```

### **rcu\_periph\_clock\_sleep\_enable**

The description of `rcu_periph_clock_sleep_enable` is shown as below:

**Table 3-489. Function `rcu_periph_clock_sleep_enable`**

<b>Function name</b>	<code>rcu_periph_clock_sleep_enable</code>
<b>Function prototype</b>	<code>void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);</code>
<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 <code>rcu_periph_sleep_enum</code>
<i>RCU_FMC_SLP</i>	FMC clock
<i>RCU_SRAM_SLP</i>	SRAM clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* enable the FMC clock when in sleep mode */
rcu_periph_clock_sleep_enable(RCU_FMC_SLP);
```

### **rcu\_periph\_clock\_sleep\_disable**

The description of `rcu_periph_clock_sleep_disable` is shown as below:

**Table 3-490. 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 <code>rcu_periph_sleep_enum</code>
<i>RCU_FMC_SLP</i>	FMC clock
<i>RCU_SRAM_SLP</i>	SRAM clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the FMC clock when in sleep mode */
rcu_periph_clock_sleep_disable(RCU_FMC_SLP);
```

### **rcu\_periph\_reset\_enable**

The description of `rcu_periph_reset_enable` is shown as below:

**Table 3-491. 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 rcu_periph_reset_enum
<i>RCU_GPIOxRST</i>	reset GPIO ports clock (x=A,B,C,D,E,F,G)
<i>RCU_AFRST</i>	reset alternate function clock
<i>RCU_ENETRST</i>	reset ENET clock(CL series)
<i>RCU_USBDRST</i>	reset USBD clock(HD、XD series)
<i>RCU_USBFSRST</i>	reset USBFS clock(CL series)
<i>RCU_TIMERxRST</i>	reset TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGTRST</i>	reset WWDGT clock
<i>RCU_SPIxRST</i>	reset SPIx clock (x=0,1,2)
<i>RCU_USARTxRST</i>	reset USARTx clock (x=0,1,2)
<i>RCU_UARTxRST</i>	reset UARTx clock (x=3,4)
<i>RCU_I2CxRST</i>	reset I2Cx clock (x=0,1)
<i>RCU_CANxRST</i>	reset CANx clock (x=0,1) CAN1 is only available for CL series
<i>RCU_PMURST</i>	reset PMU clock
<i>RCU_DACRST</i>	reset DAC clock
<i>RCU_ADCxRST</i>	reset ADCx clock (x=0,1,2) ADC2 is not available for CL series
<i>RCU_CTCRST</i>	reset CTC clock
<i>RCU_BKPIRST</i>	reset BKPI clock
<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 3-492. 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 <code>rcu_periph_reset_enum</code>
<i>RCU_GPIOxRST</i>	reset GPIO ports clock (x=A,B,C,D,E,F,G)
<i>RCU_AFRST</i>	reset alternate function clock
<i>RCU_ENETRST</i>	reset ENET clock(CL series)
<i>RCU_USBDRST</i>	reset USBD clock(HD、XD series)
<i>RCU_USBFSRST</i>	reset USBFS clock(CL series)
<i>RCU_TIMERxRST</i>	reset TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGTRST</i>	reset WWDGT clock
<i>RCU_SPIxRST</i>	reset SPIx clock (x=0,1,2)
<i>RCU_USARTxRST</i>	reset USARTx clock (x=0,1,2)
<i>RCU_UARTxRST</i>	reset UARTx clock (x=3,4)
<i>RCU_I2CxRST</i>	reset I2Cx clock (x=0,1)
<i>RCU_CANxRST</i>	reset CANx clock (x=0,1) CAN1 is only available for CL series
<i>RCU_PMURST</i>	reset PMU clock
<i>RCU_DACRST</i>	reset DAC clock
<i>RCU_ADCxRST</i>	reset ADCx clock (x=0,1,2) ADC2 is not available for CL series
<i>RCU_CTCRST</i>	reset CTC clock
<i>RCU_BKPIRST</i>	reset BKPI clock

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 reset */
rcu_periph_reset_disable(RCU_SPI0RST);
```

### **rcu\_bkp\_reset\_enable**

The description of `rcu_bkp_reset_enable` is shown as below:

**Table 3-493. Function `rcu_bkp_reset_enable`**

Function name	rcu_bkp_reset_enable
Function prototype	void rcu_bkp_reset_enable(void);
Function descriptions	enable the BKP domain reset
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the BKP domain */
rcu_bkp_reset_enable();
```

### **rcu\_bkp\_reset\_disable**

The description of `rcu_bkp_reset_disable` is shown as below:

**Table 3-494. Function `rcu_bkp_reset_disable`**

Function name	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 3-495. 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_CKSYSRC_IRC 8M</i>	select CK_IRC8M as the CK_SYS source
<i>RCU_CKSYSRC_HX TAL</i>	select CK_HXTAL as the CK_SYS source
<i>RCU_CKSYSRC_PLL</i>	select CK_PLL as the CK_SYS source
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* configure the CK_HXTAL as the CK_SYS source */

rcu_system_clock_source_config(RCU_CKSYSRC_HXTAL);
```

### **rcu\_system\_clock\_source\_get**

The description of `rcu_system_clock_source_get` is shown as below:

**Table 3-496. Function `rcu_system_clock_source_get`**

<b>Function name</b>	rcu_system_clock_source_get
<b>Function prototype</b>	uint32_t rcu_system_clock_source_get(void);
<b>Function descriptions</b>	get the system clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	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 3-497. Function `rcu_ahb_clock_config`**

<b>Function name</b>	rcu_ahb_clock_config
----------------------	----------------------

<b>Function prototype</b>	void rcu_ahb_clock_config(uint32_t ck_ahb);
<b>Function descriptions</b>	configure the AHB clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_ahb</b>	AHB clock prescaler selection
<i>RCU_AHB_CKSYS_DI Vx</i>	select CK_SYS / x, (x=1, 2, 4, 8, 16, 64, 128, 256, 512)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure CK_SYS/128 */
rcu_ahb_clock_config(RCU_AHB_CKSYS_DIV128);
```

### **rcu\_apb1\_clock\_config**

The description of rcu\_apb1\_clock\_config is shown as below:

**Table 3-498. Function rcu\_apb1\_clock\_config**

<b>Function name</b>	rcu_apb1_clock_config
<b>Function prototype</b>	void rcu_apb1_clock_config(uint32_t ck_apb1);
<b>Function descriptions</b>	configure the APB1 clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_apb1</b>	APB1 clock prescaler selection
<i>RCU_APB1_CKAHB_DIV1</i>	select CK_AHB as CK_APB1
<i>RCU_APB1_CKAHB_DIV2</i>	select CK_AHB/2 as CK_APB1

<i>RCU_APB1_CKAHB_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 3-499. Function `rcu_apb2_clock_config`**

<b>Function name</b>	<code>rcu_apb2_clock_config</code>
<b>Function prototype</b>	<code>void rcu_apb2_clock_config(uint32_t ck_apb2);</code>
<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>	select CK_AHB/8 as CK_APB2

<i>IV8</i>	
<i>RCU_APB2_CKAHB_D 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 3-500. 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 ONE</i>	no clock selected
<i>RCU_CKOUT0SRC_C KSYS</i>	select system clock CK_SYS
<i>RCU_CKOUT0SRC_IR C8M</i>	select high speed 8M internal oscillator clock
<i>RCU_CKOUT0SRC_H XTAL</i>	select HXTAL
<i>RCU_CKOUT0SRC_C KPLL_DIV2</i>	select (CK_PLL / 2) clock

<i>RCU_CKOUT0SRC_C_KPLL1</i>	select CK_PLL1 clock
<i>RCU_CKOUT0SRC_C_KPLL2_DIV2</i>	select (CK_PLL2 / 2) clock
<i>RCU_CKOUT0SRC_C_KPLL2</i>	select CK_PLL2 clock
<i>RCU_CKOUT0SRC_E_XT1</i>	select EXT1 clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the HXTAL as CK_OUT0 clock source */
rcu_ckout0_config(RCU_CKOUT0SRC_HXTAL);
```

### rcu\_pll\_config

The description of rcu\_pll\_config is shown as below:

**Table 3-501. Function rcu\_pll\_config**

<b>Function name</b>	rcu_pll_config
<b>Function prototype</b>	void rcu_pll_config(uint32_t pll_src, uint32_t pll_mul);
<b>Function descriptions</b>	configure the main PLL clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pll_src</b>	PLL clock source selection
<i>RCU_PLLSRC_IRC8M_DIV2</i>	IRC8M/2 clock is selected as source clock of PLL
<i>RCU_PLLSRC_HXTAL_IRC48M</i>	HXTAL or IRC48M is selected as source clock of PLL
<b>Input parameter{in}</b>	

<b>pll_mul</b>	PLL clock multiplication factor
<i>RCU_PLL_MULx</i>	PLL clock * x (XD series x = 2..63, CL series x = 2..14, 16..63, 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 3-502. Function `rcu_pllpresel_config`**

<b>Function name</b>	rcu_pllpresel_config
<b>Function prototype</b>	void rcu_pllpresel_config(uint32_t pll_presel);
<b>Function descriptions</b>	configure the PLL clock source preselection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pll_presel</b>	PLL clock source preselection
<i>RCU_PLLPRESRC_HXTAL</i>	HXTAL selected as PREDV0 source clock
<i>RCU_PLLPRESRC_IRC48M</i>	CK_PLL selected as PREDV0 input source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL clock source preselection */
```

rcu\_pllpresel\_config (RCU\_PLLPRESRC\_HXTAL);

### **rcu\_predv0\_config(HD、XD series)**

The description of rcu\_predv0\_config is shown as below:

**Table 3-503. 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
<b>RCU_PREDV0_DIVx</b>	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 series)**

The description of rcu\_predv0\_config is shown as below:

**Table 3-504. 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_H_XTAL_IRC48M</i>	select HXTAL or IRC48M as PREDV0 input source clock
<i>RCU_PREDV0SRC_C_KPLL1</i>	select CK_PLL1 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_DIV4);
```

### **rcu\_predv1\_config(CL series)**

The description of `rcu_predv1_config` is shown as below:

**Table 3-505. Function `rcu_predv1_config`**

<b>Function name</b>	<code>rcu_predv1_config</code>
<b>Function prototype</b>	<code>void rcu_predv1_config(uint32_t predv1_div);</code>
<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(CL series)

The description of rcu\_pll1\_config is shown as below:

**Table 3-506. 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(CL series)

The description of rcu\_pll2\_config is shown as below:

**Table 3-507. Function rcu\_pll2\_config**

<b>Function name</b>	rcu_pll2_config
<b>Function prototype</b>	void rcu_pll2_config(uint32_t pll_mul)
<b>Function descriptions</b>	configure the PLL2 clock

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pll_mul</b>	PLL clock multiplication factor
<i>RCU_PLL2_MULx</i>	PLL2 clock * x, (x = 8..14,16,20,18..32,40)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL2 clock */
rcu_pll2_config(RCU_PLL2_MUL8);
```

### **rcu\_adc\_clock\_config**

The description of **rcu\_adc\_clock\_config** is shown as below:

**Table 3-508. Function *rcu\_adc\_clock\_config***

<b>Function name</b>	<i>rcu_adc_clock_config</i>
<b>Function prototype</b>	void <i>rcu_adc_clock_config</i> (uint32_t <i>adc_psc</i> );
<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_CKAPB_2_DIV2</i>	$CK_{ADC} = CK_{APB2} / 2$
<i>RCU_CKADC_CKAPB_2_DIV4</i>	$CK_{ADC} = CK_{APB2} / 4$
<i>RCU_CKADC_CKAPB_2_DIV6</i>	$CK_{ADC} = CK_{APB2} / 6$
<i>RCU_CKADC_CKAPB</i>	$CK_{ADC} = CK_{APB2} / 8$

<i>2_DIV8</i>	
<i>RCU_CKADC_CKAPB2_DIV12</i>	$CK_{ADC} = CK_{APB2} / 12$
<i>RCU_CKADC_CKAPB2_DIV16</i>	$CK_{ADC} = CK_{APB2} / 16$
<i>RCU_CKADC_CKAHB_DIV5</i>	$CK_{ADC} = CK_{AHB} / 5$
<i>RCU_CKADC_CKAHB_DIV6</i>	$CK_{ADC} = CK_{AHB} / 6$
<i>RCU_CKADC_CKAHB_DIV10</i>	$CK_{ADC} = CK_{AHB} / 10$
<i>RCU_CKADC_CKAHB_DIV20</i>	$CK_{ADC} = CK_{AHB} / 20$
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the ADC prescaler factor */
rcu_adc_clock_config(RCU_CKADC_CKAPB2_DIV8);
```

### **rcu\_usb\_clock\_config**

The description of `rcu_usb_clock_config` is shown as below:

**Table 3-509. Function `rcu_usb_clock_config`**

<b>Function name</b>	<code>rcu_usb_clock_config</code>
<b>Function prototype</b>	<code>void rcu_usb_clock_config(uint32_t usb_psc);</code>
<b>Function descriptions</b>	configure the USB prescaler factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usb_psc</b>	USB prescaler factor

<i>RCU_CKUSB_CKPLL_DIV1_5</i>	CK_USBD/USBFS = CK_PLL / 1.5
<i>RCU_CKUSB_CKPLL_DIV1</i>	CK_USBD/USBFS = CK_PLL / 1
<i>RCU_CKUSB_CKPLL_DIV2_5</i>	CK_USBD/USBFS = CK_PLL / 2.5
<i>RCU_CKUSB_CKPLL_DIV2</i>	CK_USBD/USBFS = CK_PLL / 2
<i>RCU_CKUSB_CKPLL_DIV3</i>	CK_USBD/USBFS = CK_PLL / 3
<i>RCU_CKUSB_CKPLL_DIV3_5</i>	CK_USBD/USBFS = CK_PLL / 3.5
<i>RCU_CKUSB_CKPLL_DIV4</i>	CK_USBD/USBFS = CK_PLL / 4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the USB prescaler factor */
rcu_usb_clock_config(RCU_CKUSB_CKPLL_DIV2_5);
```

#### **rcu\_RTC\_clock\_config**

The description of `rcu_RTC_clock_config` is shown as below:

**Table 3-510. Function `rcu_RTC_clock_config`**

<b>Function name</b>	<code>rcu_RTC_clock_config</code>
<b>Function prototype</b>	<code>void rcu_RTC_clock_config(uint32_t rtc_clock_source);</code>
<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_RTC_SRC_NONE</i>	no clock selected
<i>RCU_RTC_SRC_LXTAL</i>	select CK_LXTAL as RTC source clock
<i>RCU_RTC_SRC_IRC40K</i>	select CK_IRC40K as RTC source clock
<i>RCU_RTC_SRC_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_RTC_SRC_IRC40K);
```

### rcu\_i2s1\_clock\_config(CL series)

The description of `rcu_i2s1_clock_config` is shown as below:

**Table 3-511. Function `rcu_i2s1_clock_config`**

<b>Function name</b>	<code>rcu_i2s1_clock_config</code>
<b>Function prototype</b>	<code>void rcu_i2s1_clock_config(uint32_t i2s_clock_source);</code>
<b>Function descriptions</b>	configure the I2S1 clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2s_clock_source</b>	I2S clock source selection
<i>RCU_I2S1SRC_CKSYS</i>	select system clock as I2S1 source clock
<i>RCU_I2S1SRC_CKPLL2_MUL2</i>	select CK_PLL2 * 2 as I2S1 source clock
<b>Output parameter{out}</b>	

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

Example:

```
/* configure the I2S1 clock source selection */

rcu_i2s1_clock_config(RCU_I2S1SRC_CKPLL2_MUL2);
```

### **rcu\_i2s2\_clock\_config(CL series)**

The description of rcu\_i2s2\_clock\_config is shown as below:

**Table 3-512. Function rcu\_i2s2\_clock\_config**

<b>Function name</b>	rcu_i2s2_clock_config
<b>Function prototype</b>	void rcu_i2s2_clock_config(uint32_t i2s_clock_source);
<b>Function descriptions</b>	configure the I2S2 clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2s_clock_source</b>	I2S clock source selection
<b>RCU_I2S2SRC_CKSYS</b>	select system clock as I2S2 source clock
<b>RCU_I2S2SRC_CKPLL2_MUL2</b>	select CK_PLL2 * 2 as I2S2 source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the I2S2 clock source selection */

rcu_i2s2_clock_config(RCU_I2S2SRC_CKPLL2_MUL2);
```

### **rcu\_ck48m\_clock\_config**

The description of rcu\_ck48m\_clock\_config is shown as below:

**Table 3-513. 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_CK PLL</i>	CK_PLL selected as CK48M source clock
<i>RCU_CK48MSRC_IRC48M</i>	CK_IRC48M selected as CK48M source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the CK48M clock source selection */
rcu_ck48m_clock_config (RCU_CK48MSRC_IRC48M);
```

### rcu\_flag\_get

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

**Table 3-514. 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 periphral reset flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	the clock stabilization and periphral reset flags, refer to rcu_flag_enum

<i>RCU_FLAG_IRC8MSTB</i>	IRC8M stabilization flag
<i>RCU_FLAG_HXTALSTB</i>	HXTAL stabilization flag
<i>RCU_FLAG_PLLSTB</i>	PLL stabilization flag
<i>RCU_FLAG_PLL1STB</i>	PLL1 stabilization flag(CL series only)
<i>RCU_FLAG_PLL2STB</i>	PLL2 stabilization flag(CL series only)
<i>RCU_FLAG_LXTALSTB</i>	LXTAL stabilization flag
<i>RCU_FLAG_IRC40KSTB</i>	IRC40K stabilization flag
<i>RCU_FLAG_IRC48MSTB</i>	IRC48M stabilization flag
<i>RCU_FLAG_EPRST</i>	external PIN reset flag
<i>RCU_FLAG_PORRST</i>	power reset flag
<i>RCU_FLAG_SWRST</i>	software reset flag
<i>RCU_FLAG_FWDGTRST</i>	free watchdog timer reset flag
<i>RCU_FLAG_WWDGTRST</i>	window watchdog timer reset flag
<i>RCU_FLAG_LPRST</i>	low-power reset flag
<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 3-515. 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 3-516. 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 rcu_int_flag_enum
<i>RCU_INT_FLAG_IRC4OKSTB</i>	IRC40K stabilization interrupt flag
<i>RCU_INT_FLAG_LXTALSTB</i>	LXTAL stabilization interrupt flag

<i>RCU_INT_FLAG_IRC8MSTB</i>	IRC8M stabilization interrupt flag
<i>RCU_INT_FLAG_HXTALSTB</i>	HXTAL stabilization interrupt flag
<i>RCU_INT_FLAG_PLLSTB</i>	PLL stabilization interrupt flag
<i>RCU_INT_FLAG_PLL1STB</i>	PLL1 stabilization interrupt flag(CL series only)
<i>RCU_INT_FLAG_PLL2STB</i>	PLL2 stabilization interrupt flag(CL series only)
<i>RCU_INT_FLAG_CKM</i>	HXTAL clock stuck interrupt flag
<i>RCU_INT_FLAG_IRC48MSTB</i>	IRC48M stabilization interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the clock stabilization interrupt flag */
if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){
}
```

### **rcu\_interrupt\_flag\_clear**

The description of `rcu_interrupt_flag_clear` is shown as below:

**Table 3-517. Function `rcu_interrupt_flag_clear`**

<b>Function name</b>	<code>rcu_interrupt_flag_clear</code>
<b>Function prototype</b>	<code>void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag)</code>
<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 rcu_int_flag_clear_enum
<i>RCU_INT_FLAG_IRC4OKSTB_CLR</i>	IRC40K stabilization interrupt flag clear
<i>RCU_INT_FLAG_LXTALSTB_CLR</i>	LXTAL stabilization interrupt flag clear
<i>RCU_INT_FLAG_IRC8MSTB_CLR</i>	IRC8M stabilization interrupt flag clear
<i>RCU_INT_FLAG_HXTALSTB_CLR</i>	HXTAL stabilization interrupt flag clear
<i>RCU_INT_FLAG_PLLSTB_CLR</i>	PLL stabilization interrupt flag clear
<i>RCU_INT_FLAG_PLL1STB_CLR</i>	PLL1 stabilization interrupt flag clear(CL series only)
<i>RCU_INT_FLAG_PLL2STB_CLR</i>	PLL2 stabilization interrupt flag clear(CL series only)
<i>RCU_INT_FLAG_CKM_CLR</i>	clock stuck interrupt flag clear
<i>RCU_INT_FLAG_IRC48MSTB_CLR</i>	IRC48M stabilization interrupt flag clear
<b>Output parameter{out}</b>	
-	
<b>Return value</b>	
-	

Example:

```
/* clear the interrupt HXTAL stabilization interrupt */
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

### **rcu\_interrupt\_enable**

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

**Table 3-518. 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 rcu_int_enum
<i>RCU_INT_IRC40KSTB</i>	IRC40K stabilization interrupt enable
<i>RCU_INT_LXTALSTB</i>	LXTAL stabilization interrupt enable
<i>RCU_INT_IRC8MSTB</i>	IRC8M stabilization interrupt enable
<i>RCU_INT_HXTALSTB</i>	HXTAL stabilization interrupt enable
<i>RCU_INT_PLLSTB</i>	PLL stabilization interrupt enable
<i>RCU_INT_PLL1STB</i>	PLL1 stabilization interrupt enable(CL series only)
<i>RCU_INT_PLL2STB</i>	PLL2 stabilization interrupt enable(CL series only)
<i>RCU_INT_IRC48MSTB</i>	IRC48M stabilization interrupt enable
<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 3-519. 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>	-

The called functions	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	clock stabilization interrupt, refer to rcu_int_enum
<i>RCU_INT_IRC40KSTB</i>	IRC40K stabilization interrupt enable
<i>RCU_INT_LXTALSTB</i>	LXTAL stabilization interrupt enable
<i>RCU_INT_IRC8MSTB</i>	IRC8M stabilization interrupt enable
<i>RCU_INT_HXTALSTB</i>	HXTAL stabilization interrupt enable
<i>RCU_INT_PLLSTB</i>	PLL stabilization interrupt enable
<i>RCU_INT_PLL1STB</i>	PLL1 stabilization interrupt enable(CL series only)
<i>RCU_INT_PLL2STB</i>	PLL2 stabilization interrupt enable(CL series only)
<i>RCU_INT_IRC48MSTB</i>	IRC48M stabilization interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the HXTAL stabilization interrupt */
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

### **rcu\_lxtal\_drive\_capability\_config**

The description of `rcu_lxtal_drive_capability_config` is shown as below:

**Table 3-520. 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_GHDRI</i>	medium high driving capability
<i>RCU_LXTAL_HIGHDRI</i>	higher driving capability
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the LXTAL drive capability */
rcu_lxtal_drive_capability_config (RCU_LXTAL_LOWDRI);
```

### **rcu\_osc\_stab\_wait**

The description of **rcu\_osc\_stab\_wait** is shown as below:

**Table 3-521. Function **rcu\_osc\_stab\_wait****

<b>Function name</b>	rcu_osc_stab_wait
<b>Function prototype</b>	ErrStatus rcu_osc_stab_wait(rcu_osc_type_enum osci);
<b>Function descriptions</b>	wait for oscillator stabilization flags is SET or oscillator startup is timeout
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <b>rcu_osc_type_enum</b>
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<i>RCU_IRC8M</i>	internal 8M RC oscillators(IRC8M)
<i>RCU_IRC48M</i>	internal 48M RC oscillators(IRC48M)
<i>RCU_IRC40K</i>	internal 40K RC oscillator(IRC40K)
<i>RCU_PLL_CK</i>	phase locked loop(PLL)

<i>RCU_PLL1_CK</i>	phase locked loop 1(CL series only)
<i>RCU_PLL2_CK</i>	phase locked loop 2(CL series only)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```

/* wait for oscillator stabilization flag */

if(SUCCESS == rcu_osci_stab_wait(RCU_HXTAL)){
}

```

### **rcu\_osci\_on**

The description of **rcu\_osci\_on** is shown as below:

**Table 3-522. Function **rcu\_osci\_on****

<b>Function name</b>	rcu_osci_on
<b>Function prototype</b>	void rcu_osci_on(rcu_osci_type_enum osci);
<b>Function descriptions</b>	turn on the oscillator
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <b>rcu_osci_type_enum</b>
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<i>RCU_IRC8M</i>	internal 8M RC oscillators(IRC8M)
<i>RCU_IRC48M</i>	internal 48M RC oscillators(IRC48M)
<i>RCU_IRC40K</i>	internal 40K RC oscillator(IRC40K)
<i>RCU_PLL_CK</i>	phase locked loop(PLL)
<i>RCU_PLL1_CK</i>	phase locked loop 1(CL series only)
<i>RCU_PLL2_CK</i>	phase locked loop 2(CL series only)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn on the high speed crystal oscillator */
```

```
rcu_osc_on(RCU_HXTAL);
```

### rcu\_osc\_off

The description of rcu\_osc\_off is shown as below:

**Table 3-523. Function rcu\_osc\_off**

Function name	rcu_osc_off
Function prototype	void rcu_osc_off(rcu_osc_type_enum osci);
Function descriptions	turn off the oscillator
Precondition	-
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to rcu_osc_type_enum
RCU_HXTAL	high speed crystal oscillator(HXTAL)
RCU_LXTAL	low speed crystal oscillator(LXTAL)
RCU_IRC8M	internal 8M RC oscillators(IRC8M)
RCU_IRC48M	internal 48M RC oscillators(IRC48M)
RCU_IRC40K	internal 40K RC oscillator(IRC40K)
RCU_PLL_CK	phase locked loop(PLL)
RCU_PLL1_CK	phase locked loop 1(CL series only)
RCU_PLL2_CK	phase locked loop 2(CL series only)
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* turn off the high speed crystal oscillator */
rcu_osc_i_off(RCU_HXTAL);
```

### **rcu\_osc\_i\_bypass\_mode\_enable**

The description of `rcu_osc_i_bypass_mode_enable` is shown as below:

**Table 3-524. Function `rcu_osc_i_bypass_mode_enable`**

<b>Function name</b>	rcu_osc_i_bypass_mode_enable
<b>Function prototype</b>	void rcu_osc_i_bypass_mode_enable(rcu_osc_i_type_enum osci);
<b>Function descriptions</b>	enable the oscillator bypass mode
<b>Precondition</b>	HXTALEN or LXTALEN must be reset before it
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <code>rcu_osc_i_type_enum</code>
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the high speed crystal oscillator bypass mode */
rcu_osc_i_bypass_mode_enable(RCU_HXTAL);
```

### **rcu\_osc\_i\_bypass\_mode\_disable**

The description of `rcu_osc_i_bypass_mode_disable` is shown as below:

**Table 3-525. Function `rcu_osc_i_bypass_mode_disable`**

<b>Function name</b>	rcu_osc_i_bypass_mode_disable
<b>Function prototype</b>	void rcu_osc_i_bypass_mode_disable(rcu_osc_i_type_enum osci);

<b>Function descriptions</b>	disable the oscillator bypass mode
<b>Precondition</b>	HXTALEN or LXTALEN must be reset before it
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to rcu_osc_type_enum
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the high speed crystal oscillator bypass mode */
rcu_osc_bypass_mode_disable(RCU_HXTAL);
```

### **rcu\_hxtal\_clock\_monitor\_enable**

The description of **rcu\_hxtal\_clock\_monitor\_enable** is shown as below:

**Table 3-526. Function **rcu\_hxtal\_clock\_monitor\_enable****

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

Example:

```
/* enable the HXTAL clock monitor */

rcu_hxtal_clock_monitor_enable();
```

### **rcu\_hxtal\_clock\_monitor\_disable**

The description of `rcu_hxtal_clock_monitor_disable` is shown as below:

**Table 3-527. Function `rcu_hxtal_clock_monitor_disable`**

<b>Function name</b>	rcu_hxtal_clock_monitor_disable
<b>Function prototype</b>	void rcu_hxtal_clock_monitor_disable(void);
<b>Function descriptions</b>	disable the HXTAL clock monitor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the HXTAL clock monitor */

rcu_hxtal_clock_monitor_disable();
```

### **rcu\_irc8m\_adjust\_value\_set**

The description of `rcu_irc8m_adjust_value_set` is shown as below:

**Table 3-528. Function `rcu_irc8m_adjust_value_set`**

<b>Function name</b>	rcu_irc8m_adjust_value_set
<b>Function prototype</b>	void rcu_irc8m_adjust_value_set(uint32_t irc8m_adjval);
<b>Function descriptions</b>	set the IRC8M adjust value
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
irc8m_adjval	IRC8M adjust value, must be between 0 and 0x1F
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the IRC8M adjust value */
rcu_irc8m_adjust_value_set(0x10);
```

### rcu\_deepsleep\_voltage\_set

The description of rcu\_deepsleep\_voltage\_set is shown as below:

**Table 3-529. Function rcu\_deepsleep\_voltage\_set**

<b>Function name</b>	rcu_deepsleep_voltage_set
<b>Function prototype</b>	void rcu_deepsleep_voltage_set(uint32_t dsvol);
<b>Function descriptions</b>	set the deep-sleep mode voltage value
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>dsvol</b>	deep sleep mode voltage
<i>RCU_DEEPSLEEP_V_1_0</i>	the core voltage is 1.0V in deep-sleep mode
<i>RCU_DEEPSLEEP_V_0_9</i>	the core voltage is 0.9V in deep-sleep mode
<i>RCU_DEEPSLEEP_V_0_8</i>	the core voltage is 0.8V in deep-sleep mode
<i>RCU_DEEPSLEEP_V_0_7</i>	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 3-530. Function rcu\_clock\_freq\_get**

<b>Function name</b>	rcu_clock_freq_get
<b>Function prototype</b>	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);
<b>Function descriptions</b>	get the system clock, bus clock frequency
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>clock</b>	the clock frequency which to get
<b>CK_SYS</b>	system clock frequency
<b>CK_AHB</b>	AHB clock frequency
<b>CK_APB1</b>	APB1 clock frequency
<b>CK_APB2</b>	APB2 clock frequency
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ck_freq</b>	clock frequency of system, AHB, APB1, APB2

Example:

```
uint32_t temp_freq;
/* get the system clock frequency */
temp_freq = rcu_clock_freq_get(CK_SYS);
```

## 3.20. 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 [3.20.1](#), the RTC firmware functions are introduced in chapter [3.20.2](#).

### 3.20.1. Descriptions of Peripheral registers

RTC registers are listed in the table shown as below:

**Table 3-531. RTC Registers**

Registers	Descriptions
RTC_INTEN	Interrupt enable register
RTC_CTL	Control register
RTC_PSCH	Prescaler high register
RTC_PSCL	Prescaler low register
RTC_DIVH	Divider high register
RTC_DIVL	Divider low register
RTC_CNTH	counter high register
RTC_CNTL	counter low register
RTC_ALRMH	Alarm high register
RTC_ALRML	Alarm low register

### 3.20.2. Descriptions of Peripheral functions

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

**Table 3-532. RTC firmware function**

Function name	Function description
rtc_interrupt_enable	enable RTC interrupt
rtc_interrupt_disable	disable RTC interrupt
rtc_configuration_mode_enter	enter RTC configuration mode
rtc_configuration_mode_exit	exit RTC configuration mode
rtc_lwoff_wait	wait RTC last write operation finished flag set

Function name	Function description
rtc_register_sync_wait	wait RTC registers synchronized flag set
rtc_counter_get	get RTC counter value
rtc_counter_set	set RTC counter value
rtc_prescaler_set	set RTC prescaler value
rtc_alarm_config	set RTC alarm value
rtc_divider_get	get RTC divider value
rtc_flag_get	get RTC flag status
rtc_flag_clear	clear RTC flag status

## rtc\_interrupt\_enable

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

**Table 3-533. Function rtc\_interrupt\_enable**

Function name	rtc_interrupt_enable
Function prototype	void rtc_interrupt_enable(uint32_t interrupt);
Function descriptions	enable RTC interrupt
Precondition	before using this function, you must call rtc_lwoff_wait ( ) function (wait until LWOFF flag is set).
The called functions	-
<b>Input parameter{in}</b>	
interrupt	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 3-534. 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 <code>rtc_lwoff_wait()</code> function (wait until LWOFF flag is set).
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which RTC interrupt to disable
<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\_configuration\_mode\_enter**

The description of `rtc_configuration_mode_enter` is shown as below:

**Table 3-535. Function rtc\_configuration\_mode\_enter**

<b>Function name</b>	rtc_configuration_mode_enter
<b>Function prototype</b>	void rtc_configuration_mode_enter(void);
<b>Function descriptions</b>	enter 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:

```
/* enter RTC configuration mode */
rtc_configuration_mode_enter();
```

### **rtc\_configuration\_mode\_exit**

The description of rtc\_configuration\_mode\_exit is shown as below:

**Table 3-536. 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 3-537. 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 3-538. 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 3-539. 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>	
<b>uint32_t</b>	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 3-540. 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 3-541. 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 <code>rtc_lwoff_wait()</code> 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 3-542. 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 <code>rtc_lwoff_wait()</code> 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 3-543. Function `rtc_divider_get`**

<b>Function name</b>	rtc_divider_get
<b>Function prototype</b>	uint32_t rtc_divider_get(void);
<b>Function descriptions</b>	get RTC divider value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	the value of RTC divider

Example:

```
/* get the current RTC divider value */
uint32_t rtc_divider_value;
rtc_divider_value = rtc_divider_get ( );
```

### **rtc\_flag\_get**

The description of `rtc_flag_get` is shown as below:

**Table 3-544. 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> <i>W</i>	overflow interrupt flag
<i>RTC_FLAG_RSYN</i>	registers synchronized flag
<i>RTC_FLAG_LWOF</i>	last write operation finished flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the RTC overflow interrupt status */

FlagStatus alarm_status;

alarm_status = rtc_flag_get (RTC_FLAG_ALARM);
```

### rtc\_flag\_clear

The description of `rtc_flag_clear` is shown as below:

**Table 3-545. 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>	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);
```

## 3.21. 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 [3.21.13.21.1](#), the SDIO firmware functions are introduced in chapter [3.21.2](#).

### 3.21.1. Descriptions of Peripheral registers

SDIO registers are listed in the table shown as below:

**Table 3-546. 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

Registers	Descriptions
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

### 3.21.2. Descriptions of Peripheral functions

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

**Table 3-547. 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

Function name	Function description
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
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)

Function name	Function description
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_clock_config	configure the SDIO clock

## sdio\_deinit

The description of sdio\_deinit is shown as below:

**Table 3-548. Function sdio\_deinit**

Function name	sdio_deinit
Function prototype	void sdio_deinit(void);
Function descriptions	deinitialize the SDIO
Precondition	-
The called functions	-
<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 3-549. 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
<i>SDIO_SDIOCLKEDGE_RISING</i>	select the rising edge of the SDIOCLK to generate SDIO_CLK
<i>SDIO_SDIOCLKEDGE_FALLING</i>	select the falling edge of the SDIOCLK to generate SDIO_CLK
<b>Input parameter{in}</b>	
<b>clock_bypass</b>	clock bypass
<i>SDIO_CLOCKBYPASS_ENABLE</i>	clock bypass
<i>SDIO_CLOCKBYPASS_DISABLE</i>	no bypass
<b>Input parameter{in}</b>	
<b>clock_powersave</b>	SDIO_CLK clock dynamic switch on/off for power saving
<i>SDIO_CLOCKPWRSAVE_ENABLE</i>	SDIO_CLK closed when bus is idle
<i>SDIO_CLOCKPWRSAVE_DISABLE</i>	SDIO_CLK clock is always on
<b>Input parameter{in}</b>	
<b>clock_division</b>	clock division, less than 512
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* configure the SDIO clock */

sdio_clock_config(SDIO_SDIOCLKEDGE_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 3-550. 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>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

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 3-551. 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>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-552. Function `sdio_bus_mode_set`**

<b>Function name</b>	sdio_bus_mode_set
<b>Function prototype</b>	void sdio_bus_mode_set(uint32_t bus_mode);
<b>Function descriptions</b>	set different SDIO card bus mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>bus_mode</b>	SDIO card bus mode
<i>SDIO_BUSMODE_1BIT</i> <i>T</i>	1-bit SDIO card bus mode
<i>SDIO_BUSMODE_4BIT</i> <i>T</i>	4-bit SDIO card bus mode
<i>SDIO_BUSMODE_8BIT</i> <i>T</i>	8-bit SDIO card bus mode
<b>Output parameter{out}</b>	

-	-
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 3-553. Function `sdio_power_state_set`**

<b>Function name</b>	sdio_power_state_set
<b>Function prototype</b>	void sdio_power_state_set(uint32_t power_state);
<b>Function descriptions</b>	set the SDIO power state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>power_state</b>	SDIO power state
<b>SDIO_POWER_ON</b>	SDIO power on
<b>SDIO_POWER_OFF</b>	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 3-554. Function sdio\_power\_state\_get**

<b>Function name</b>	sdio_power_state_get
<b>Function prototype</b>	uint32_t sdio_power_state_get(void);
<b>Function descriptions</b>	get the SDIO power state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	SDIO_POWER_ON / SDIO_POWER_OFF

Example:

```
/* get the SDIO power state */
uint32_t sdio_power_value;
sdio_power_value = sdio_power_state_get();
```

### **sdio\_clock\_enable**

The description of sdio\_clock\_enable is shown as below:

**Table 3-555. 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>	
-	-

Return value	
-	-

Example:

```
/* enable SDIO_CLK clock output */

sdio_clock_enable();
```

### **sdio\_clock\_disable**

The description of sdio\_clock\_disable is shown as below:

**Table 3-556. 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 3-557. 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_RESPONSETYP_E_NO</i>	no response
<i>SDIO_RESPONSETYP_E_SHORT</i>	short response
<i>SDIO_RESPONSETYP_E_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 3-558. Function `sdio_wait_type_set`**

<b>Function name</b>	<code>sdio_wait_type_set</code>
<b>Function prototype</b>	<code>void sdio_wait_type_set(uint32_t wait_type);</code>
<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_INT ERRUPT</i>	wait interrupt
<i>SDIO_WAITTYPE_DAT AEND</i>	wait the end of data transfer
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the command state machine wait type */
sdio_wait_type_set(SDIO_WAITTYPE_NO);
```

### **sdio\_csm\_enable**

The description of **sdio\_csm\_enable** is shown as below:

**Table 3-559. 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 3-560. 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 3-561. Function sdio\_command\_index\_get**

<b>Function name</b>	sdio_command_index_get
<b>Function prototype</b>	uint8_t sdio_command_index_get(void);
<b>Function descriptions</b>	get the last response command index
<b>Precondition</b>	-

<b>The called functions</b>	
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	last response command index

Example:

```
/* get SDIO command index */

uint8_t sdio_command_value;

sdio_command_value = sdio_command_index_get();
```

### **sdio\_response\_get**

The description of `sdio_response_get` is shown as below:

**Table 3-562. Function `sdio_response_get`**

<b>Function name</b>	sdio_response_get
<b>Function prototype</b>	uint32_t sdio_response_get(uint32_t response);
<b>Function descriptions</b>	get the response for the last received command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>response</b>	SDIO response
<i>SDIO_RESPONSE0</i>	card response[31:0]/card response[127:96]
<i>SDIO_RESPONSE1</i>	card response[95:64]
<i>SDIO_RESPONSE2</i>	card response[63:32]
<i>SDIO_RESPONSE3</i>	card response[31:1], plus bit 0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

uint32_t	response for the last received command
----------	--

Example:

```
/* store the CID0 numbers */

uint32_t sdio_cid[0];

sdio_cid[0] = sdio_response_get(SDIO_RESPONSE0);
```

### **sdio\_data\_config**

The description of **sdio\_data\_config** is shown as below:

**Table 3-563. 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
<i>SDIO_DATABLOCKSIZE_1BYTE</i>	block size = 1 byte
<i>SDIO_DATABLOCKSIZE_2BYTES</i>	block size = 2 bytes
<i>SDIO_DATABLOCKSIZE_4BYTES</i>	block size = 4 bytes
<i>SDIO_DATABLOCKSIZE_8BYTES</i>	block size = 8 bytes
<i>SDIO_DATABLOCKSIZE_16BYTES</i>	block size = 16 bytes

<code>SDIO_DATABLOCKSIZE_32BYTES</code>	block size = 32 bytes
<code>SDIO_DATABLOCKSIZE_64BYTES</code>	block size = 64 bytes
<code>SDIO_DATABLOCKSIZE_128BYTES</code>	block size = 128 bytes
<code>SDIO_DATABLOCKSIZE_256BYTES</code>	block size = 256 bytes
<code>SDIO_DATABLOCKSIZE_512BYTES</code>	block size = 512 bytes
<code>SDIO_DATABLOCKSIZE_1024BYTES</code>	block size = 1024 bytes
<code>SDIO_DATABLOCKSIZE_2048BYTES</code>	block size = 2048 bytes
<code>SDIO_DATABLOCKSIZE_4096BYTES</code>	block size = 4096 bytes
<code>SDIO_DATABLOCKSIZE_8192BYTES</code>	block size = 8192 bytes
<code>SDIO_DATABLOCKSIZE_16384BYTES</code>	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 3-564. Function `sdio_data_transfer_config`**

Function name	<code>sdio_data_transfer_config</code>

<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 transmission */
sdio_data_transfer_config(SDIO_TRANSDIRECTI ON_TOSDIO,
SDIO_TRANSMODE_BLOCK);
```

### **sdio\_dsm\_enable**

The description of **sdio\_dsm\_enable** is shown as below:

**Table 3-565. 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 3-566. Function sdio\_dsm\_disable**

<b>Function name</b>	sdio_dsm_disable
<b>Function prototype</b>	void sdio_dsm_disable(void);
<b>Function descriptions</b>	disable the DSM(data state machine)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the DSM(data state machine) */
```

```
sdio_dsm_disable();
```

### **sdio\_data\_write**

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

**Table 3-567. Function sdio\_data\_write**

<b>Function name</b>	sdio_data_write
<b>Function prototype</b>	void sdio_data_write(uint32_t data);
<b>Function descriptions</b>	write data(one word) to the transmit FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>data</b>	32-bit data write to card
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* write data(one word) to the transmit FIFO */
sdio_data_write(0x0000 0001);
```

### **sdio\_data\_read**

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

**Table 3-568. Function sdio\_data\_read**

<b>Function name</b>	sdio_data_read
<b>Function prototype</b>	uint32_t sdio_data_read(void);
<b>Function descriptions</b>	read data(one word) from the receive FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-

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 3-569. 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 3-570. Function sdio\_data\_counter\_get**

<b>Function name</b>	sdio_fifo_counter_get
<b>Function prototype</b>	uint32_t sdio_fifo_counter_get(void);
<b>Function descriptions</b>	get the number of words remaining to be written or read from FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
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 3-571. Function sdio\_dma\_enable**

<b>Function name</b>	sdio_dma_enable
<b>Function prototype</b>	void sdio_dma_enable(void);
<b>Function descriptions</b>	enable the DMA request for SDIO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* enable the SDIO DMA */

sdio_dma_enable();
```

### **sdio\_dma\_disable**

The description of `sdio_dma_disable` is shown as below:

**Table 3-572. Function `sdio_dma_disable`**

<b>Function name</b>	sdio_dma_disable
<b>Function prototype</b>	void sdio_dma_disable(void);
<b>Function descriptions</b>	disable the DMA request for SDIO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the SDIO DMA */

sdio_dma_disable();
```

### **sdio\_flag\_get**

The description of `sdio_flag_get` is shown as below:

**Table 3-573. Function `sdio_flag_get`**

<b>Function name</b>	sdio_flag_get
<b>Function prototype</b>	FlagStatus sdio_flag_get(uint32_t flag);
<b>Function descriptions</b>	get the flags state of SDIO

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flags state of SDIO
<i>SDIO_FLAG_CCRCER</i> <i>R</i>	command response received (CRC check failed) flag
<i>SDIO_FLAG_DTCRCE</i> <i>RR</i>	data block sent/received (CRC check failed) flag
<i>SDIO_FLAG_CMDTMO</i> <i>UT</i>	command response timeout flag
<i>SDIO_FLAG_DTTMOU</i> <i>T</i>	data timeout flag
<i>SDIO_FLAG_TXURE</i>	transmit FIFO underrun error occurs flag
<i>SDIO_FLAG_RXORE</i>	received FIFO overrun error occurs flag
<i>SDIO_FLAG_CMDREC</i> <i>V</i>	command response received (CRC check passed) flag
<i>SDIO_FLAG_CMDSEN</i> <i>D</i>	command sent (no response required) flag
<i>SDIO_FLAG_DTEND</i>	data end (data counter, SDIO_DATACNT, is zero) flag
<i>SDIO_FLAG_STBITE</i>	start bit error in the bus flag
<i>SDIO_FLAG_DTBLSK</i> <i>ND</i>	data block sent/received (CRC check passed) flag
<i>SDIO_FLAG_CMDRUN</i>	command transmission in progress flag
<i>SDIO_FLAG_TXRUN</i>	data transmission in progress flag
<i>SDIO_FLAG_RXRUN</i>	data reception in progress flag
<i>SDIO_FLAG_TFH</i>	transmit FIFO is half empty flag: at least 8 words can be written into the FIFO
<i>SDIO_FLAG_RFH</i>	receive FIFO is half full flag: at least 8 words can be read in the FIFO
<i>SDIO_FLAG_TFF</i>	transmit FIFO is full flag
<i>SDIO_FLAG_RFF</i>	receive FIFO is full flag
<i>SDIO_FLAG_TFE</i>	transmit FIFO is empty flag

<i>SDIO_FLAG_RFE</i>	receive FIFO is empty flag
<i>SDIO_FLAG_TXDTVAL</i>	data is valid in transmit FIFO flag
<i>SDIO_FLAG_RXDTVA L</i>	data is valid in receive FIFO flag
<i>SDIO_FLAG_SDIOINT</i>	SD I/O interrupt received flag
<i>SDIO_FLAG_ATAEND</i>	CE-ATA command completion signal received (only for CMD61) flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the flags state of SDIO */

FlagStatus flag_value;

flag_value = sdio_flag_get(SDIO_FLAG_RFH);
```

### **sdio\_flag\_clear**

The description of `sdio_flag_clear` is shown as below:

**Table 3-574. Function `sdio_flag_clear`**

<b>Function name</b>	<code>sdio_flag_clear</code>
<b>Function prototype</b>	<code>void sdio_flag_clear(uint32_t flag);</code>
<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_CCRCER R</i>	command response received (CRC check failed) flag
<i>SDIO_FLAG_DTCRCE RR</i>	data block sent/received (CRC check failed) flag
<i>SDIO_FLAG_CMDTMO</i>	command response timeout flag

<b>UT</b>	
<i>SDIO_FLAG_DTTMOU</i> <i>T</i>	data timeout flag
<i>SDIO_FLAG_TXURE</i>	transmit FIFO underrun error occurs flag
<i>SDIO_FLAG_RXORE</i>	received FIFO overrun error occurs flag
<i>SDIO_FLAG_CMDREC</i> <i>V</i>	command response received (CRC check passed) flag
<i>SDIO_FLAG_CMDSEN</i> <i>D</i>	command sent (no response required) flag
<i>SDIO_FLAG_DTEND</i>	data end (data counter, SDIO_DATACNT, is zero) flag
<i>SDIO_FLAG_STBITE</i>	start bit error in the bus flag
<i>SDIO_FLAG_DTBKLE</i> <i>ND</i>	data block sent/received (CRC check passed) flag
<i>SDIO_FLAG_SDIOINT</i>	SD I/O interrupt received flag
<i>SDIO_FLAG_ATAEND</i>	CE-ATA command completion signal received (only for CMD61) flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the pending flags of SDIO */
sdio_flag_clear(SDIO_FLAG_DTCRCERR);
```

### **sdio\_interrupt\_enable**

The description of `sdio_interrupt_enable` is shown as below:

**Table 3-575. Function `sdio_interrupt_enable`**

<b>Function name</b>	<code>sdio_interrupt_enable</code>
<b>Function prototype</b>	<code>void sdio_interrupt_enable(uint32_t int_flag);</code>
<b>Function descriptions</b>	enable the SDIO interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<code>int_flag</code>	interrupt flags state of SDIO
<code>SDIO_INT_CCRCERR</code>	SDIO CCRCERR interrupt
<code>SDIO_INT_DTCRCER</code> <i>R</i>	SDIO DTCRCERR interrupt
<code>SDIO_INT_CMDTMOU</code> <i>T</i>	SDIO CMDTMOU interrupt
<code>SDIO_INT_DTTMOUT</code>	SDIO DTTMOUT interrupt
<code>SDIO_INT_TXURE</code>	SDIO TXURE interrupt
<code>SDIO_INT_RXORE</code>	SDIO_INT_RXORE
<code>SDIO_INT_CMDRECV</code>	SDIO CMDRECV interrupt
<code>SDIO_INT_CMDSEND</code>	SDIO CMDSEND interrupt
<code>SDIO_INT_DTEND</code>	SDIO DTEND interrupt
<code>SDIO_INT_STBITE</code>	SDIO STBITE interrupt
<code>SDIO_INT_DTBLKEND</code>	SDIO DTBLKEND interrupt
<code>SDIO_INT_CMDRUN</code>	SDIO CMDRUN interrupt
<code>SDIO_INT_TXRUN</code>	SDIO TXRUN interrupt
<code>SDIO_INT_RXRUN</code>	SDIO RXRUN interrupt
<code>SDIO_INT_TFH</code>	SDIO TFH interrupt
<code>SDIO_INT_RFH</code>	SDIO RFH interrupt
<code>SDIO_INT_TFF</code>	SDIO TFF interrupt
<code>SDIO_INT_RFF</code>	SDIO RFF interrupt
<code>SDIO_INT_TFE</code>	SDIO TFE interrupt
<code>SDIO_INT_RFE</code>	SDIO RFE interrupt
<code>SDIO_INT_TXDTVAL</code>	SDIO TXDTVAL interrupt
<code>SDIO_INT_RXDTVAL</code>	SDIO RXDTVAL interrupt
<code>SDIO_INT_SDIOINT</code>	SDIO SDIOINT interrupt
<code>SDIO_INT_ATAEND</code>	SDIO ATAEND interrupt
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable the SDIO corresponding interrupts */

sdio_interrupt_enable(SDIO_INT_CCRCERR | SDIO_INT_DTTMOUT | SDIO_INT_RXORE
| SDIO_INT_DTEND | SDIO_INT_STBITE);
```

### **sdio\_interrupt\_disable**

The description of `sdio_interrupt_disable` is shown as below:

**Table 3-576. Function `sdio_interrupt_disable`**

<b>Function name</b>	sdio_interrupt_disable
<b>Function prototype</b>	void sdio_interrupt_disable(uint32_t int_flag);
<b>Function descriptions</b>	disable the SDIO interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt flags state of SDIO
<b>SDIO_INT_CCRCERR</b>	SDIO CCRCERR interrupt
<b>SDIO_INT_DTCRCER R</b>	SDIO DTCRCERR interrupt
<b>SDIO_INT_CMDTMOU T</b>	SDIO CMDTMOU interrupt
<b>SDIO_INT_DTTMOUT</b>	SDIO DTTMOUT interrupt
<b>SDIO_INT_TXURE</b>	SDIO TXURE interrupt
<b>SDIO_INT_RXORE</b>	SDIO_INT_RXORE
<b>SDIO_INT_CMDRECV</b>	SDIO CMDRECV interrupt
<b>SDIO_INT_CMDSEND</b>	SDIO CMDSEND interrupt
<b>SDIO_INT_DTEND</b>	SDIO DTEND interrupt
<b>SDIO_INT_STBITE</b>	SDIO STBITE interrupt

<i>SDIO_INT_DTBKEND</i>	SDIO DTBKEND 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 3-577. 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>	-

The called functions	-
Input parameter{in}	
<b>int_flag</b>	interrupt flags state of SDIO
<b>SDIO_INT_FLAG_CCR CERR</b>	SDIO CCRCERR interrupt flag
<b>SDIO_INT_FLAG_DTC RCERR</b>	SDIO DTCRCERR interrupt flag
<b>SDIO_INT_FLAG_CMD TMOUT</b>	SDIO CMDTMOUT interrupt flag
<b>SDIO_INT_FLAG_DTT MOUT</b>	SDIO DTTMOUT interrupt flag
<b>SDIO_INT_FLAG_TXU RE</b>	SDIO TXURE interrupt flag
<b>SDIO_INT_FLAG_RXO RE</b>	SDIO_INT_RXORE flag
<b>SDIO_INT_FLAG_CMD RECV</b>	SDIO CMDRECV interrupt flag
<b>SDIO_INT_FLAG_CMD SEND</b>	SDIO CMDSEND interrupt flag
<b>SDIO_INT_FLAG_DTE ND</b>	SDIO DTEND interrupt flag
<b>SDIO_INT_FLAG_STB TE</b>	SDIO STBITE interrupt flag
<b>SDIO_INT_FLAG_DTB LKEND</b>	SDIO DTBLKEND interrupt flag
<b>SDIO_INT_FLAG_CMD RUN</b>	SDIO CMDRUN interrupt flag
<b>SDIO_INT_FLAG_TXR UN</b>	SDIO TXRUN interrupt flag
<b>SDIO_INT_FLAG_RXR UN</b>	SDIO RXRUN interrupt flag
<b>SDIO_INT_FLAG_TFH</b>	SDIO TFH interrupt flag
<b>SDIO_INT_FLAG_RFH</b>	SDIO RFH interrupt flag

<i>SDIO_INT_FLAG_TFF</i>	SDIO TFF interrupt flag
<i>SDIO_INT_FLAG_RFF</i>	SDIO RFF interrupt flag
<i>SDIO_INT_FLAG_TFE</i>	SDIO TFE interrupt flag
<i>SDIO_INT_FLAG_RFE</i>	SDIO RFE interrupt flag
<i>SDIO_INT_FLAG_TXDTVAL</i> <i>TVAL</i>	SDIO TXDTVAL interrupt flag
<i>SDIO_INT_FLAG_RXDTVAL</i> <i>TVAL</i>	SDIO RXDTVAL interrupt flag
<i>SDIO_INT_FLAG_SDIOINT</i> <i>OINT</i>	SDIO SDIOINT interrupt flag
<i>SDIO_INT_FLAG_ATAEND</i>	SDIO ATAEND interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the interrupt flags state of SDIO */
FlagStatus flag_value;
flag_value = sdio_interrupt_flag_get(SDIO_INT_FLAG_DTEND);
```

### **sdio\_interrupt\_flag\_clear**

The description of `sdio_interrupt_flag_clear` is shown as below:

**Table 3-578. Function `sdio_interrupt_flag_clear`**

<b>Function name</b>	<code>sdio_interrupt_flag_clear</code>
<b>Function prototype</b>	<code>void sdio_interrupt_flag_clear(uint32_t int_flag);</code>
<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 3-579. 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 3-580. 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 3-581. 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 3-582. Function sdio\_stop\_readwait\_disable**

Function name	sdio_stop_readwait_disable
---------------	----------------------------

<b>Function prototype</b>	void sdio_stop_readwait_disable(void);
<b>Function descriptions</b>	disable the function that stop the read wait process(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-583. Function `sdio_readwait_type_set`**

<b>Function name</b>	sdio_readwait_type_set
<b>Function prototype</b>	void sdio_readwait_type_set(uint32_t readwait_type);
<b>Function descriptions</b>	set the read wait type(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>readwait_type</b>	SD I/O read wait type
<b>SDIO_READWAITTYPE_P_E_CLK</b>	read wait control by stopping SDIO_CLK
<b>SDIO_READWAITTYPE_P_E_DAT2</b>	read wait control using SDIO_DAT[2]
<b>Output parameter{out}</b>	
-	-

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 3-584. Function sdio\_operation\_enable**

<b>Function name</b>	sdio_operation_enable
<b>Function prototype</b>	void sdio_operation_enable(void);
<b>Function descriptions</b>	enable the SD I/O mode specific operation(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the SD I/O mode specific operation(SD I/O only) */
sdio_operation_enable();
```

### **sdio\_operation\_disable**

The description of sdio\_operation\_disable is shown as below:

**Table 3-585. Function sdio\_operation\_disable**

<b>Function name</b>	sdio_operation_disable
<b>Function prototype</b>	void sdio_operation_disable(void);
<b>Function descriptions</b>	disable the SD I/O mode specific operation(SD I/O only)

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

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 3-586. Function `sdio_suspend_enable`**

<b>Function name</b>	sdio_suspend_enable
<b>Function prototype</b>	void sdio_suspend_enable(void);
<b>Function descriptions</b>	enable the SD I/O suspend operation(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the SD I/O suspend operation(SD I/O only) */
void sdio_suspend_enable();
```

### **sdio\_suspend\_disable**

The description of sdio\_suspend\_disable is shown as below:

**Table 3-587. Function sdio\_suspend\_disable**

<b>Function name</b>	sdio_suspend_disable
<b>Function prototype</b>	void sdio_suspend_disable(void);
<b>Function descriptions</b>	disable the SD I/O suspend operation(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the SD I/O suspend operation(SD I/O only) */
sdio_suspend_disable();
```

### **sdio\_ceata\_command\_enable**

The description of sdio\_ceata\_command\_enable is shown as below:

**Table 3-588. Function sdio\_ceata\_command\_enable**

<b>Function name</b>	sdio_ceata_command_enable
<b>Function prototype</b>	void sdio_ceata_command_enable(void);
<b>Function descriptions</b>	enable the CE-ATA command(CE-ATA only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	

-	-
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 3-589. 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 3-590. 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 3-591. 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 3-592. 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 3-593. 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>	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the CE-ATA command completion signal(CE-ATA only) */

sdio_ceata_command_completion_disable();
```

## 3.22. 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 [3.22.1](#), the SPI/I2S firmware functions are introduced in chapter [3.22.2](#).

### 3.22.1. Descriptions of Peripheral registers

SPI/I2S registers are listed in the table shown as below:

Table 3-594. SPI/I2S Registers

Registers	Descriptions
SPI_CTL0	SPI control register 0
SPI_CTL1	SPI control register 1
SPI_STAT	SPI status register
SPI_DATA	SPI data register
SPI_CRCPOLY	SPI CRC polynomial register
SPI_RCRC	SPI receive CRC register
SPI_TCRC	SPI transmit CRC register
SPI_I2SCTL	SPI/I2S control register
SPI_I2SPSC	SPI/I2S clock prescaler register
SPI_QCTL	Quad-SPI mode control register

### 3.22.2. Descriptions of Peripheral functions

SPI/I2S firmware functions are listed in the table shown as below:

**Table 3-595. 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_crc_polynomial_set	set SPI CRC polynomial
spi_crc_polynomial_get	get SPI CRC polynomial
spi_crc_on	turn on SPI CRC function
spi_crc_off	turn off SPI CRC function
spi_crc_next	SPI next data is CRC value
spi_crc_get	get SPI CRC send value or receive value
spi_ti_mode_enable	enable SPI TI mode
spi_ti_mode_disable	disable SPI TI mode

Function name	Function description
spi_nssp_mode_enable	enable SPI NSS pulse mode
spi_nssp_mode_disable	disable SPI NSS pulse mode
qspi_enable	enable quad wire SPI
qspi_disable	disable quad wire SPI
qspi_write_enable	enable quad wire SPI write
qspi_read_enable	enable quad wire SPI read
qspi_io23_output_enable	enable quad wire SPI_IO2 and SPI_IO3 pin output
qspi_io23_output_disable	disable quad wire SPI_IO2 and SPI_IO3 pin output
spi_i2s_interrupt_enable	enable SPI and I2S interrupt
spi_i2s_interrupt_disable	disable SPI and I2S interrupt
spi_i2s_interrupt_flag_get	get SPI and I2S interrupt status
spi_i2s_flag_get	get SPI and I2S flag status
spi_crc_error_clear	clear SPI CRC error flag status

## Structure spi\_parameter\_struct

Table 3-596. 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_FRAMESIZE_16BIT, SPI_FRAMESIZE_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 3-597. Function spi\_i2s\_deinit**

<b>Function name</b>	spi_i2s_deinit
<b>Function prototype</b>	void spi_i2s_deinit(uint32_t spi_periph);
<b>Function descriptions</b>	reset SPI and I2S peripheral
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
spi_periph	SPI/I2S peripheral
SPIx	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset SPI0 */
spi_i2s_deinit(SPI0);
```

### spi\_struct\_para\_init

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

**Table 3-598. 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>	
*spi_struct	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 3-599. Function spi\_init**

<b>Function name</b>	spi_init
<b>Function prototype</b>	void spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);
<b>Function descriptions</b>	initialize SPI peripheral parameter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1,2
<b>Input parameter{in}</b>	
spi_struct	SPI parameter initialization stuct, the structure members can refer to members of the structure <a href="#">Table 3-596. spi_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* initialize SPI0 */

spi_parameter_struct spi_init_struct;

spi_init_struct.trans_mode          = SPI_TRANSMODE_BDTRANSMIT;
spi_init_struct.device_mode         = SPI_MASTER;
spi_init_struct.frame_size         = SPI_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 3-600. 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>	
spi_periph	SPI peripheral
SPIx	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 3-601. 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>	
spi_periph	SPI peripheral
SPIx	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 3-602. 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
<b>SPIx</b>	x=1,2
<b>Input parameter{in}</b>	
<b>i2s_mode</b>	I2S operation mode
<b>I2S_MODE_SLAVE TX</b>	I2S slave transmit mode
<b>I2S_MODE_SLAVE RX</b>	I2S slave receive mode
<b>I2S_MODE_MASTER TX X</b>	I2S master transmit mode
<b>I2S_MODE_MASTER RX X</b>	I2S master receive mode
<b>Input parameter{in}</b>	
<b>i2s_standard</b>	I2S standard
<b>I2S_STD_PHILLIPS</b>	I2S phillips standard
<b>I2S_STD_MSB</b>	I2S MSB standard
<b>I2S_STD_LSB</b>	I2S LSB standard
<b>I2S_STD_PCMSHORT</b>	I2S PCM short standard
<b>I2S_STD_PCMLONG</b>	I2S PCM long standard
<b>Input parameter{in}</b>	
<b>i2s_ckpl</b>	I2S idle state clock polarity
<b>I2S_CKPL_LOW</b>	I2S clock polarity low level
<b>I2S_CKPL_HIGH</b>	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 3-603. Function i2s\_psc\_config**

<b>Function name</b>	i2s_psc_config
<b>Function prototype</b>	void i2s_psc_config(uint32_t spi_periph, uint32_t i2s_audiosample, uint32_t i2s_frameformat, uint32_t i2s_mckout);
<b>Function descriptions</b>	configure I2S prescaler
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S peripheral
<i>SPIx</i>	x=1,2
<b>Input parameter{in}</b>	
<b>i2s_audiosample</b>	I2S audio sample rate
<i>I2S_AUDIOSAMPLE_8K</i>	audio sample rate is 8KHz
<i>I2S_AUDIOSAMPLE_11K</i>	audio sample rate is 11KHz
<i>I2S_AUDIOSAMPLE_16K</i>	audio sample rate is 16KHz
<i>I2S_AUDIOSAMPLE_22K</i>	audio sample rate is 22KHz
<i>I2S_AUDIOSAMPLE_32K</i>	audio sample rate is 32KHz
<i>I2S_AUDIOSAMPLE_44K</i>	audio sample rate is 44KHz
<i>I2S_AUDIOSAMPLE_48K</i>	audio sample rate is 48KHz
<i>I2S_AUDIOSAMPLE_96K</i>	audio sample rate is 96KHz

	<i>6K</i>
<i>I2S_AUDIOSAMPLE_192K</i>	audio sample rate is 192KHz
<b>Input parameter{in}</b>	
<i>i2s_frameformat</i>	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>	
<i>i2s_mckout</i>	I2S master clock output
<i>I2S_MCKOUT_ENABLE</i>	I2S master clock output enable
<i>I2S_MCKOUT_DISABLE</i>	I2S master clock output disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure I2S1 prescaler */

i2s_psc_config(SPI1, I2S_AUDIOSAMPLE_44K, I2S_FRAMEFORMAT_DT16B_CH16B,
I2S_MCKOUT_DISABLE);
```

### i2s\_enable

The description of i2s\_enable is shown as below:

**Table 3-604. Function i2s\_enable**

<b>Function name</b>	i2s_enable
----------------------	------------

<b>Function prototype</b>	void i2s_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable I2S
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	I2S peripheral
SPIx	x=1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2S1*/
i2s_enable(SPI1);
```

### i2s\_disable

The description of i2s\_disable is shown as below:

**Table 3-605. 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>	
spi_periph	I2S peripheral
SPIx	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 3-606. 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
<b>SPIx</b>	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 3-607. 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
<b>SPIx</b>	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 3-608. 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
<b>SPIx</b>	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 3-609. 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>	
spi_periph	SPI peripheral
SPIx	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 3-610. 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>	-

Input parameter{in}	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 transmit data DMA function */
spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);
```

### **spi\_dma\_disable**

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

**Table 3-611. 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>	-
Input parameter{in}	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
Input parameter{in}	
<b>dma</b>	SPI DMA mode
<i>SPI_DMA_TRANSMIT</i>	SPI transmit data use DMA

<i>SPI_DMA_RECEIVE</i>	SPI receive data use DMA
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 transmit data DMA function */
spi_dma_disable(SPI0, SPI_DMA_TRANSMIT);
```

### **spi\_i2s\_data\_frame\_format\_config**

The description of *spi\_i2s\_data\_frame\_format\_config* is shown as below:

**Table 3-612. Function *spi\_i2s\_data\_frame\_format\_config***

<b>Function name</b>	<i>spi_i2s_data_frame_format_config</i>
<b>Function prototype</b>	void <i>spi_i2s_data_frame_format_config</i> (uint32_t <i>spi_periph</i> , uint16_t <i>frame_format</i> );
<b>Function descriptions</b>	configure SPI/I2S data frame format
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>spi_periph</i>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<i>frame_format</i>	SPI frame size
<i>SPI_FRAMESIZE_16BIT</i>	SPI frame size is 16 bits
<i>SPI_FRAMESIZE_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_FRAMESIZE_16BIT);
```

### spi\_i2s\_data\_transmit

The description of spi\_i2s\_data\_transmit is shown as below:

**Table 3-613. Function spi\_i2s\_data\_transmit**

<b>Function name</b>	spi_i2s_data_transmit
<b>Function prototype</b>	void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);
<b>Function descriptions</b>	SPI transmit data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1,2
<b>Input parameter{in}</b>	
<b>data</b>	16-bit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 transmit data */

uint16_t spi0_send_array[] = {0x5050,0XA0A0};

spi_i2s_data_transmit(SPI0, spi0_send_array[0]);
```

### spi\_i2s\_data\_receive

The description of spi\_i2s\_data\_receive is shown as below:

**Table 3-614. 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>	
spi_periph	SPI peripheral
SPIx	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	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 3-615. 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>	
spi_periph	SPI peripheral
SPIx	x=0,1,2
<b>Input parameter{in}</b>	

<b>transfer_direction</b>	SPI transfer direction
<b>SPI_BIDIRECTIONAL_TRANSMIT</b>	SPI work in transmit-only mode
<b>SPI_BIDIRECTIONAL_RECEIVE</b>	SPI work in receive-only mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 works in transmit-only mode */
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

### spi\_crc\_polynomial\_set

The description of spi\_crc\_polynomial\_set is shown as below:

**Table 3-616. 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
<b>SPIx</b>	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 3-617. 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>	
spi_periph	SPI peripheral
SPIx	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	16-bit CRC polynomial (0-0xFFFF)

Example:

```
/* get SPI0 CRC polynomial */
uint16_t crc_val;
crc_val = spi_crc_polynomial_get(SPI0);
```

### **spi\_crc\_on**

The description of spi\_crc\_on is shown as below:

**Table 3-618. 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
<b>SPIx</b>	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 3-619. 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
<b>SPIx</b>	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 3-620. 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>	
spi_periph	SPI peripheral
SPIx	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 3-621. 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>	-

Input parameter{in}	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
Input parameter{in}	
<b>crc</b>	SPI crc value
<i>SPI_CRC_TX</i>	get transmit crc value
<i>SPI_CRC_RX</i>	get receive crc value
Output parameter{out}	
-	-
Return value	
<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\_ti\_mode\_enable**

The description of **spi\_ti\_mode\_enable** is shown as below:

**Table 3-622. 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>	-
Input parameter{in}	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
Output parameter{out}	
-	-

Return value	
-	-

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 3-623. 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
<b>SPIx</b>	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 3-624. 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
<b>SPIx</b>	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 3-625. 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
<b>SPIx</b>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 NSS pulse mode */
spi_nssp_mode_disable(SPI0);
```

### **qspi\_enable**

The description of qspi\_enable is shown as below:

**Table 3-626. Function qspi\_enable**

<b>Function name</b>	qspi_enable
<b>Function prototype</b>	void qspi_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable quad wire SPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 quad wire mode */
qspi_enable(SPI0);
```

### **qspi\_disable**

The description of qspi\_disable is shown as below:

**Table 3-627. Function qspi\_disable**

<b>Function name</b>	qspi_disable
<b>Function prototype</b>	qspi_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable quad wire SPI
<b>Precondition</b>	-

<b>The called functions</b>		-
<b>Input parameter{in}</b>		
<b>spi_periph</b>		SPI peripheral
<b>SPIx</b>		x=0
<b>Output parameter{out}</b>		
-		-
<b>Return value</b>		
-		-

Example:

```
/* disable SPI0 quad wire mode */
qspi_disable(SPI0);
```

### **qspi\_write\_enable**

The description of qspi\_write\_enable is shown as below:

**Table 3-628. Function qspi\_write\_enable**

<b>Function name</b>	qspi_write_enable
<b>Function prototype</b>	void qspi_write_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable quad wire SPI write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0
<b>Output parameter{out}</b>	
-	
<b>Return value</b>	
-	

Example:

```
/* enable SPI0 quad wire write */
```

```
qspi_write_enable(SPI0);
```

### **qspi\_read\_enable**

The description of qspi\_read\_enable is shown as below:

**Table 3-629. Function qspi\_read\_enable**

<b>Function name</b>	qspi_read_enable
<b>Function prototype</b>	void qspi_read_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable quad wire SPI read
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 quad wire read */
qspi_read_enable(SPI0);
```

### **qspi\_io23\_output\_enable**

The description of qspi\_io23\_output\_enable is shown as below:

**Table 3-630. Function qspi\_io23\_output\_enable**

<b>Function name</b>	qspi_io23_output_enable
<b>Function prototype</b>	void qspi_io23_output_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable SPI_IO2 and SPI_IO3 pin output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 SPI_IO2 and SPI_IO3 pin output */
qspi_io23_output_enable(SPI0);
```

### **qspi\_io23\_output\_disable**

The description of qspi\_io23\_output\_disable is shown as below:

**Table 3-631. Function qspi\_io23\_output\_disable**

<b>Function name</b>	qspi_io23_output_disable
<b>Function prototype</b>	void qspi_io23_output_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable SPI_IO2 and SPI_IO3 pin output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 SPI_IO2 and SPI_IO3 pin output */
qspi_io23_output_disable(SPI0);
```

### spi\_i2s\_interrupt\_enable

The description of spi\_i2s\_interrupt\_enable is shown as below:

**Table 3-632. 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>	
spi_periph	SPI peripheral
SPIx	x=0,1,2
<b>Input parameter{in}</b>	
interrupt	SPI/I2S interrupt
SPI_I2S_INT_TBE	transmit buffer empty interrupt
SPI_I2S_INT_RBNE	receive buffer not empty interrupt
SPI_I2S_INT_ERR	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
<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 3-633. 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
<b>SPIx</b>	x=0,1,2
<b>Input parameter{in}</b>	
<b>interrupt</b>	SPI/I2S interrupt
<b>SPI_I2S_INT_TBE</b>	transmit buffer empty interrupt
<b>SPI_I2S_INT_RBNE</b>	receive buffer not empty interrupt
<b>SPI_I2S_INT_ERR</b>	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 3-634. 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
<b>SPIx</b>	x=0,1,2
<b>Input parameter{in}</b>	
<b>interrupt</b>	SPI/I2S interrupt flag status
<b>SPI_I2S_INT_FLAG_T BE</b>	transmit buffer empty interrupt
<b>SPI_I2S_INT_FLAG_R BNE</b>	receive buffer not empty interrupt
<b>SPI_I2S_INT_FLAG_R XORERR</b>	overrun interrupt
<b>SPI_INT_FLAG_CONF ERR</b>	config error interrupt
<b>SPI_INT_FLAG_CRCE RR</b>	CRC error interrupt
<b>I2S_INT_FLAG_TXUR ERR</b>	underrun error interrupt
<b>SPI_I2S_INT_FLAG_F ERR</b>	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 3-635. 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
<b>SPIx</b>	x=0,1,2
<b>Input parameter{in}</b>	
<b>flag</b>	SPI/I2S flag status
<b>SPI_FLAG_TBE</b>	transmit buffer empty flag
<b>SPI_FLAG_RBNE</b>	receive buffer not empty flag
<b>SPI_FLAG_TRANS</b>	transmit on-going flag
<b>SPI_I2S_INT_FLAG_RXORERR</b>	receive overrun error flag
<b>SPI_FLAG_CONFERR</b>	mode config error flag
<b>SPI_FLAG_CRCERR</b>	CRC error flag
<b>SPI_FLAG_FREE</b>	format error flag
<b>I2S_FLAG_TBE</b>	transmit buffer empty flag
<b>I2S_FLAG_RBNE</b>	receive buffer not empty flag
<b>I2S_FLAG_TRANS</b>	transmit on-going flag
<b>I2S_FLAG_RXORERR</b>	overrun error flag
<b>I2S_FLAG_TXURERR</b>	underrun error flag
<b>I2S_FLAG_CH</b>	channel side flag
<b>I2S_FLAG_FERR</b>	format error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

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

Example:

```
/* get SPI0 transmit buffer empty flag status */

while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));

spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
```

### **spi\_crc\_error\_clear**

The description of spi\_crc\_error\_clear is shown as below:

**Table 3-636. 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
<b>SPIx</b>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear SPI0 CRC error flag status */

spi_crc_error_clear(SPI0);
```

## **3.23. 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 [3.23.1](#), the TIMER firmware functions are introduced in chapter [3.23.2](#).

## 3.23.1. Descriptions of Peripheral registers

TIMERx registers are listed in the table shown as below:

**Table 3-637. 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

## 3.23.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

**Table 3-638. TIMERx firmware function**

Function name	Function description
timer_deinit	deinit a timer
timer_struct_para_init	initialize the parameters of TIMER init parameter struct with the default values
timer_init	initialize TIMER counter
timer_enable	enable a timer
timer_disable	disable a timer
timer_auto_reload_shadow_enable	enable the auto reload shadow function
timer_auto_reload_shadow_disable	disable the auto reload shadow function
timer_update_event_enable	enable the update event
timer_update_event_disable	disable the update event
timer_counter_alignment	set TIMER counter alignment mode
timer_counter_up_direction	set TIMER counter up direction
timer_counter_down_direction	set TIMER counter down direction
timer_prescaler_config	configure TIMER prescaler
timer_repetition_value_config	configure TIMER repetition register value
timer_autoreload_value_config	configure TIMER autoreload register value
timer_counter_value_config	configure TIMER counter register value
timer_counter_read	read TIMER counter value
timer_prescaler_read	read TIMER prescaler value
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

Function name	Function description
timer_break_struct_para_init	initialize the parameters of TIMER break parameter struct with the default values
timer_break_config	configure TIMER break function
timer_break_enable	enable TIMER break function
timer_break_disable	disable TIMER break function
timer_automatic_output_enable	enable TIMER output automatic function
timer_automatic_output_disable	disable TIMER output automatic function
timer_primary_output_config	configure TIMER primary output function
timer_channel_control_shadow_config	channel capture/compare control shadow register enable
timer_channel_control_shadow_update_config	configure TIMER channel control shadow register update control
timer_channel_output_struct_para_init	initialize the parameters of TIMER channel output parameter struct with the default values
timer_channel_output_config	configure TIMER channel output function
timer_channel_output_mode_config	configure TIMER channel output compare mode
timer_channel_output_pulse_value_config	configure TIMER channel output pulse value
timer_channel_output_shadow_config	configure TIMER channel output shadow function
timer_channel_output_fast_config	configure TIMER channel output fast function
timer_channel_output_clear_config	configure TIMER channel output clear function
timer_channel_output_polarity_config	configure TIMER channel output polarity
timer_channel_complementary_output_polarity_config	configure TIMER channel complementary output polarity
timer_channel_output_state_config	configure TIMER channel enable state
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

Function name	Function description
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_interrupt_enable	enable the TIMER interrupt
timer_interrupt_disable	disable the TIMER interrupt
timer_interrupt_flag_get	get timer interrupt flag

Function name	Function description
timer_interrupt_flag_clear	clear TIMER interrupt flag
timer_flag_get	get TIMER flags
timer_flag_clear	clear TIMER flags

### Structure timer\_parameter\_struct

Table 3-639. Structure timer\_parameter\_struct

Member name	Function description
prescaler	prescaler value (0~65535)
alignedmode	aligned mode (TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH)
counterdirection	counter direction (TIMER_COUNTER_UP, TIMER_COUNTER_DOWN)
period	period value (0~65535)
clockdivision	clock division value (TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4)
repetitioncounter	the counter repetition value (0~255)

### Structure timer\_break\_parameter\_struct

Table 3-640. 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 3-641. Structure timer\_oc\_parameter\_struct**

Member name	Function description
outputstate	channel output state (TIMER_CCX_ENABLE, TIMER_CCX_DISABLE)
outputnstate	channel complementary output state (TIMER_CCXN_ENABLE, TIMER_CCXN_DISABLE)
ocpolarity	channel output polarity (TIMER_OC_POLARITY_HIGH, TIMER_OC_POLARITY_LOW)
ocnpolarity	channel complementary output polarity (TIMER_OCN_POLARITY_HIGH, TIMER_OCN_POLARITY_LOW)
ocidlestate	idle state of channel output (TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH)
ocnidlestate	idle state of channel complementary output (TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH)

### Structure timer\_ic\_parameter\_struct

**Table 3-642. 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 3-643. Function timer\_deinit**

Function name	timer_deinit
Function prototype	void timer_deinit(uint32_t timer_periph);
Function descriptions	deinit a TIMER
Precondition	-

<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 3-644. 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>	
initpara	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-639. 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 3-645. 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>	
timer_periph	TIMER peripheral
TIMERx(x=0..13)	TIMER peripheral selection
<b>Input parameter{in}</b>	
initpara	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-639. Structure timer_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize TIMER0 */

timer_parameter_struct timer_initpara;

timer_initpara.prescaler      = 107;
timer_initpara.alignedmode    = TIMER_COUNTER_EDGE;
timer_initpara.counterdirection = TIMER_COUNTER_UP;
timer_initpara.period         = 999;
timer_initpara.clockdivision  = TIMER_CKDIV_DIV1;
timer_initpara.repetitioncounter = 1;
```

```
timer_init(TIMER0,&timer_initpara);
```

## **timer\_enable**

The description of timer\_enable is shown as below:

**Table 3-646. 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>	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 */
timer_enable (TIMER0);
```

## **timer\_disable**

The description of timer\_disable is shown as below:

**Table 3-647. Function timer\_disable**

<b>Function name</b>	timer_disable
<b>Function prototype</b>	void timer_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable a timer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 */
timer_disable (TIMER0);
```

#### **timer\_auto\_reload\_shadow\_enable**

The description of timer\_auto\_reload\_shadow\_enable is shown as below:

**Table 3-648. Function timer\_auto\_reload\_shadow\_enable**

<b>Function name</b>	timer_auto_reload_shadow_enable
<b>Function prototype</b>	void timer_auto_reload_shadow_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable the auto reload shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the TIMER0 auto reload shadow function */
timer_auto_reload_shadow_enable (TIMER0);
```

### **timer\_auto\_reload\_shadow\_disable**

The description of timer\_auto\_reload\_shadow\_disable is shown as below:

**Table 3-649. Function timer\_auto\_reload\_shadow\_disable**

<b>Function name</b>	timer_auto_reload_shadow_disable
<b>Function prototype</b>	void timer_auto_reload_shadow_disable (uint32_t timer_periph);
<b>Function descriptions</b>	disable the auto reload shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the TIMER0 auto reload shadow function */
timer_auto_reload_shadow_disable (TIMER0);
```

### **timer\_update\_event\_enable**

The description of timer\_update\_event\_enable is shown as below:

**Table 3-650. Function timer\_update\_event\_enable**

<b>Function name</b>	timer_update_event_enable
<b>Function prototype</b>	void timer_update_event_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable the update event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral

<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 the update event */

timer_update_event_enable (TIMER0);
```

### **timer\_update\_event\_disable**

The description of timer\_update\_event\_disable is shown as below:

**Table 3-651. Function timer\_update\_event\_disable**

<b>Function name</b>	timer_update_event_disable
<b>Function prototype</b>	void timer_update_event_disable (uint32_t timer_periph);
<b>Function descriptions</b>	disable the update event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 the update event */

timer_update_event_disable (TIMER0);
```

### **timer\_counter\_alignment**

The description of timer\_counter\_alignment is shown as below:

**Table 3-652. Function timer\_counter\_alignment**

<b>Function name</b>	timer_counter_alignment
<b>Function prototype</b>	void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);
<b>Function descriptions</b>	set TIMER counter alignment mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>aligned</b>	alignment mode
<i>TIMER_COUNTER_EDGE</i>	No center-aligned mode (edge-aligned mode). The direction of the counter is specified by the DIR bit.
<i>TIMER_COUNTER_CENTER_DOWN</i>	Center-aligned and counting down assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0register). Only when the counter is counting down, compare interrupt flag of channels can be set.
<i>TIMER_COUNTER_CENTER_UP</i>	Center-aligned and counting up assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0register). Only when the counter is counting up, compare interrupt flag of channels can be set.
<i>TIMER_COUNTER_CENTER_BOTH</i>	Center-aligned and counting up/down assert mode. The counter counts under center-aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0 register). Both when the counter is counting up and counting down, compare interrupt flag of channels can be set.
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set TIMER0 counter center-aligned and counting up assert mode */
timer_counter_alignment (TIMER0, TIMER_COUNTER_CENTER_UP);
```

### **timer\_counter\_up\_direction**

The description of timer\_counter\_up\_direction is shown as below:

**Table 3-653. 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 3-654. 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 3-655. Function timer\_prescaler\_config**

<b>Function name</b>	timer_prescaler_config
<b>Function prototype</b>	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint8_t pscreload);
<b>Function descriptions</b>	configure TIMER prescaler
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>prescaler</b>	prescaler value (0~65535)
<b>Input parameter{in}</b>	
<b>pscreload</b>	prescaler reload mode
<i>TIMER_PSC_RELOAD_NOW</i>	the prescaler is loaded right now
<i>TIMER_PSC_RELOAD_UPDATE</i>	the prescaler is loaded at the next update event
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* configure TIMER0 prescaler */
timer_prescaler_config(TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

#### **timer\_repetition\_value\_config**

The description of timer\_repetition\_value\_config is shown as below:

**Table 3-656. 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>	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
repetition	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 3-657. Function timer\_autoreload\_value\_config**

<b>Function name</b>	timer_autoreload_value_config
<b>Function prototype</b>	void timer_autoreload_value_config(uint32_t timer_periph, uint16_t autoreload);
<b>Function descriptions</b>	configure TIMER autoreload register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx(x=0..13)</b>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>autoreload</b>	the counter auto-reload value (0-0xFFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER autoreload register value */
timer_autoreload_value_config (TIMER0, 3000);
```

### **timer\_counter\_value\_config**

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

**Table 3-658. Function timer\_counter\_value\_config**

<b>Function name</b>	timer_counter_value_config
<b>Function prototype</b>	void timer_counter_value_config(uint32_t timer_periph, uint16_t counter);
<b>Function descriptions</b>	configure TIMER counter register value
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Input parameter{in}	
<b>counter</b>	the counter value (0-0xFFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 counter register value */

timer_counter_value_config (TIMER0);
```

### **timer\_counter\_read**

The description of timer\_counter\_read is shown as below:

**Table 3-659. 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>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
<b>uint32_t</b>	counter value(0x0000~0xFFFF)

Example:

```
/* read TIMER0 counter value */

uint32_t i = 0;

i = timer_counter_read (TIMER0);
```

### **timer\_prescaler\_read**

The description of timer\_prescaler\_read is shown as below:

**Table 3-660. 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>	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	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 3-661. 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
<b>TIMERx(x=0..8,11)</b>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>spmode</b>	pulse mode
<b>TIMER_SP_MODE_SIN GLE</b>	single pulse mode
<b>TIMER_SP_MODE_RE PETITIVE</b>	repetitive pulse mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 single pulse mode */
timer_single_pulse_mode_config (TIMER0, TIMER_SP_MODE_SINGLE);
```

### **timer\_update\_source\_config**

The description of timer\_update\_source\_config is shown as below:

**Table 3-662. Function timer\_update\_source\_config**

<b>Function name</b>	timer_update_source_config
<b>Function prototype</b>	void timer_update_source_config(uint32_t timer_periph, uint32_t update);
<b>Function descriptions</b>	configure TIMER update source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral

<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>update</b>	update source
<i>TIMER_UPDATE_SRC_GLOBAL</i>	Any of the following events generate an update interrupt or DMA request: - The UPG bit is set - The counter generates an overflow or underflow event - The slave mode controller generates an update event
<i>TIMER_UPDATE_SRC_REGULAR</i>	Only counter overflow/underflow generates an update interrupt or DMA request.
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER update only by counter overflow/underflow */

timer_update_source_config (TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

### **timer\_dma\_enable**

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

**Table 3-663. 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 3-664. Function timer\_dma\_disable**

<b>Function name</b>	timer_dma_disable
<b>Function prototype</b>	void timer_dma_disable (uint32_t timer_periph, uint16_t dma);
<b>Function descriptions</b>	disable the TIMER DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>dma</b>	timer DMA source disable
<i>TIMER_DMA_UPD</i>	update DMA disable, 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)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the TIMER0 update DMA */
timer_dma_disable (TIMER0, TIMER_DMA_UPD);
```

#### **timer\_channel\_dma\_request\_source\_select**

The description of timer\_channel\_dma\_request\_source\_select is shown as below:

**Table 3-665. Function timer\_channel\_dma\_request\_source\_select**

<b>Function name</b>	timer_channel_dma_request_source_select
<b>Function prototype</b>	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
<b>Function descriptions</b>	channel DMA request source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx</b>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>dma_request</b>	channel DMA request source selection
<i>TIMER_DMAREQUEST_CHANNELEVENT</i>	DMA request of channel n is sent when channel y event occurs

<i>TIMER_DMAREQUEST_UPDATEEVENT</i>	DMA request of channel n is sent when update event occurs
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* TIMER0 channel DMA request of channel n is sent when channel y event occurs */

timer_channel_dma_request_source_select(TIMER0,
TIMER_DMAREQUEST_CHANNELEVENT);
```

### **timer\_dma\_transfer\_config**

The description of timer\_dma\_transfer\_config is shown as below:

**Table 3-666. Function timer\_dma\_transfer\_config**

<b>Function name</b>	timer_dma_transfer_config
<b>Function prototype</b>	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
<b>Function descriptions</b>	configure the TIMER DMA transfer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>dma_baseaddr</b>	DMA transfer access start address
<i>TIMER_DMACFG_DMA_TA_CTL0</i>	DMA transfer address is TIMER_CTL0, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CTL1</i>	DMA transfer address is TIMER_CTL1, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_SMCFG</i>	DMA transfer address is TIMER_SMCFG, TIMERx(x=0..4,7)

<i>TIMER_DMACFG_DMA_TA_DMINTEN</i>	DMA transfer address is TIMER_DMINTEN, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_INTF</i>	DMA transfer address is TIMER_INTF, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_SWEVG</i>	DMA transfer address is TIMER_SWEVG, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CHCTL0</i>	DMA transfer address is TIMER_CHCTL0, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CHCTL1</i>	DMA transfer address is TIMER_CHCTL1, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CHCTL2</i>	DMA transfer address is TIMER_CHCTL2, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CNT</i>	DMA transfer address is TIMER_CNT, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_PSC</i>	DMA transfer address is TIMER_PSC, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CAR</i>	MA transfer address is TIMER_CAR, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CREP</i>	DMA transfer address is TIMER_CREP, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMA_TA_CH0CV</i>	DMA transfer address is TIMER_CH0CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CH1CV</i>	DMA transfer address is TIMER_CH1CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CH2CV</i>	DMA transfer address is TIMER_CH2CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CH3CV</i>	DMA transfer address is TIMER_CH3CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CCHP</i>	DMA transfer address is TIMER_CCHP, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMA_TA_DMACFG</i>	DMA transfer address is TIMER_DMACFG, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_DMATB</i>	DMA transfer address is TIMER_DMATB, TIMERx(x=0..4,7)

Input parameter{in}	
<b>dma_lenth</b>	DMA transfer count
<b>TIMER_DMACFG_DMA_TC_xTRANSFER</b>	x=1..18, DMA transfer x time
Output parameter{out}	
-	-
Return value	
-	-

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 3-667. Function timer\_event\_software\_generate**

<b>Function name</b>	timer_event_software_generate
<b>Function prototype</b>	void timer_event_software_generate(uint32_t timer_periph, uint16_t event);
<b>Function descriptions</b>	software generate events
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx</b>	please refer to the following parameters
Input parameter{in}	
<b>event</b>	the timer software event generation sources
<b>TIMER_EVENT_SRC_UPG</b>	update event,TIMERx(x=0..13)
<b>TIMER_EVENT_SRC_CH0G</b>	channel 0 capture or compare event generation,TIMERx(x=0..4,7..13)

<i>TIMER_EVENT_SRC_C_H1G</i>	channel 1 capture or compare event generation,TIMERx(x=0..4,7,8,11)
<i>TIMER_EVENT_SRC_C_H2G</i>	channel 2 capture or compare event generation,TIMERx(x=0..4,7)
<i>TIMER_EVENT_SRC_C_H3G</i>	channel 3 capture or compare event generation,TIMERx(x=0..4,7)
<i>TIMER_EVENT_SRC_C_MTG</i>	channel commutation event generation,TIMERx(x=0,7)
<i>TIMER_EVENT_SRC_T_RGG</i>	trigger event generation,TIMERx(x=0..4,7,8,11)
<i>TIMER_EVENT_SRC_B_RKG</i>	break event generation,TIMERx(x=0,7)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* software generate update event*/
timer_event_software_generate (TIMER0, TIMER_EVENT_SRC_UPG);
```

#### **timer\_break\_struct\_para\_init**

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

**Table 3-668. Function timer\_break\_struct\_para\_init**

<b>Function name</b>	timer_break_struct_para_init
<b>Function prototype</b>	void timer_break_struct_para_init(timer_break_parameter_struct* breakpara);
<b>Function descriptions</b>	initialize the parameters of TIMER break parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>breakpara</b>	TIMER break parameter struct, the structure members can refer to <a href="#">Table</a>

	<a href="#"><u>3-640. Structure timer_break_parameter_struct.</u></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 3-669. Function timer\_break\_config**

<b>Function name</b>	timer_break_config
<b>Function prototype</b>	void timer_break_config(uint32_t timer_periph, timer_break_parameter_struct* breakpara);
<b>Function descriptions</b>	configure TIMER break function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>breakpara</b>	TIMER break parameter struct, the structure members can refer to <a href="#"><u>Table 3-640. Structure timer_break_parameter_struct.</u></a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

```

/* configure TIMER0 break function */

timer_break_parameter_struct timer_breakpara;

timer_breakpara.runoffstate      = TIMER_ROS_STATE_DISABLE;
timer_breakpara.ideloffstate     = TIMER_IOS_STATE_DISABLE ;
timer_breakpara.deadtime         = 255;
timer_breakpara.breakpolarity    = TIMER_BREAK_POLARITY_LOW;
timer_breakpara.outputautostate = TIMER_OUTAUTO_ENABLE;
timer_breakpara.protectmode     = TIMER_CCHP_PROT_0;
timer_breakpara.breakstate      = TIMER_BREAK_ENABLE;

timer_break_config(TIMER0, &timer_breakpara);

```

### **timer\_break\_enable**

The description of timer\_break\_enable is shown as below:

**Table 3-670. 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-field in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 break function*/
```

```
timer_break_enable (TIMER0);
```

### **timer\_break\_disable**

The description of timer\_break\_disable is shown as below:

**Table 3-671. 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-filled in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
timer_periph	TIMER peripheral
TIMERx(x=0,7)	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 break function*/
timer_break_disable (TIMER0);
```

### **timer\_automatic\_output\_enable**

The description of timer\_automatic\_output\_enable is shown as below:

**Table 3-672. 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-filled 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 3-673. 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-field 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 3-674. 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 3-675. 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
<b>TIMERx(x=0,7)</b>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<b>ENABLE</b>	enable function
<b>DISABLE</b>	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 3-676. Function timer\_channel\_control\_shadow\_update\_config**

<b>Function name</b>	timer_channel_control_shadow_update_config
<b>Function prototype</b>	void timer_channel_control_shadow_update_config(uint32_t timer_periph, uint8_t ccuctl);
<b>Function descriptions</b>	configure commutation control shadow register update control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral

<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>ccuctl</b>	channel control shadow register update control
<i>TIMER_UPDATECTL_CU</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 3-677. Function timer\_channel\_output\_struct\_para\_init**

<b>Function name</b>	timer_channel_output_struct_para_init
<b>Function prototype</b>	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpara);
<b>Function descriptions</b>	initialize the parameters of TIMER channel output parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ocpara</b>	TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-641. 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 3-678. Function timer\_channel\_output\_config**

<b>Function name</b>	timer_channel_output_config
<b>Function prototype</b>	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara);
<b>Function descriptions</b>	configure TIMER channel output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<b>TIMER_CH_0</b>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<b>TIMER_CH_1</b>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<b>TIMER_CH_2</b>	TIMER channel 2 (TIMERx(x=0..4,7))
<b>TIMER_CH_3</b>	TIMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<b>ocpara</b>	TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-641. 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 3-679. 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 ( <i>TIMERx(x=0..4,7..13)</i> )
<i>TIMER_CH_1</i>	TIMER channel 1 ( <i>TIMERx(x=0..4,7,8,11)</i> )
<i>TIMER_CH_2</i>	TIMER channel 2 ( <i>TIMERx(x=0..4,7)</i> )
<i>TIMER_CH_3</i>	TIMER channel 3 ( <i>TIMERx(x=0..4,7)</i> )

Input parameter{in}	
<b>ocmode</b>	channel output compare mode
<i>TIMER_OC_MODE_TIMING</i>	timing mode
<i>TIMER_OC_MODE_ACTIVE</i>	set the channel output
<i>TIMER_OC_MODE_INACTIVE</i>	clear the channel output
<i>TIMER_OC_MODE_TOGGLE</i>	toggle on match
<i>TIMER_OC_MODE_LOW</i>	force low mode
<i>TIMER_OC_MODE_HIGH</i>	force high mode
<i>TIMER_OC_MODE_PWM0</i>	PWM mode 0
<i>TIMER_OC_MODE_PWM1</i>	PWM mode 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel PWM 0 mode */
timer_channel_output_mode_config(TIMER0, TIMER_CH_0, TIMER_OC_MODE_PWM0);
```

#### **timer\_channel\_output\_pulse\_value\_config**

The description of timer\_channel\_output\_pulse\_value\_config is shown as below:

**Table 3-680. 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 ( <i>TIMERx</i> (x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 ( <i>TIMERx</i> (x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 ( <i>TIMERx</i> (x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 ( <i>TIMERx</i> (x=0..4,7))
<b>Input parameter{in}</b>	
<b>pulse</b>	channel output pulse value (0~65535)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 output pulse value */

timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

#### **timer\_channel\_output\_shadow\_config**

The description of `timer_channel_output_shadow_config` is shown as below:

**Table 3-681. Function `timer_channel_output_shadow_config`**

<b>Function name</b>	<code>timer_channel_output_shadow_config</code>
<b>Function prototype</b>	<code>void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);</code>
<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 ( <i>TIMERx</i> (x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 ( <i>TIMERx</i> (x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 ( <i>TIMERx</i> (x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 ( <i>TIMERx</i> (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 3-682. 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 ( <i>TIMERx</i> (x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 ( <i>TIMERx</i> (x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 ( <i>TIMERx</i> (x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 ( <i>TIMERx</i> (x=0..4,7))
<b>Input parameter{in}</b>	
<b>ocfast</b>	channel output fast function
<i>TIMER_OC_FAST_ENABLE</i>	channel output fast function enable
<i>TIMER_OC_FAST_DISABLE</i>	channel output fast function disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 output fast function */
timer_channel_output_fast_config (TIMER0, TIMER_CH_0, TIMER_OC_FAST_ENABLE);
```

### **timer\_channel\_output\_clear\_config**

The description of timer\_channel\_output\_clear\_config is shown as below:

**Table 3-683. Function timer\_channel\_output\_clear\_config**

<b>Function name</b>	timer_channel_output_clear_config
<b>Function prototype</b>	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);
<b>Function descriptions</b>	configure TIMER channel output clear function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER periphera
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<b>TIMER_CH_0</b>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<b>TIMER_CH_1</b>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<b>TIMER_CH_2</b>	TIMER channel 2 (TIMERx(x=0..4,7))
<b>TIMER_CH_3</b>	TIMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<b>occlear</b>	channel output clear function
<b>TIMER_OC_CLEAR_ENABLE</b>	channel output clear function enable
<b>TIMER_OC_CLEAR_DISABLE</b>	channel output clear function disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 output clear function */

timer_channel_output_clear_config (TIMER0, TIMER_CH_0,
TMR_OC_CLEAR_ENABLE);
```

### timer\_channel\_output\_polarity\_config

The description of timer\_channel\_output\_polarity\_config is shown as below:

**Table 3-684. 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 ( <i>TIMERx(x=0..4,7..13)</i> )
<i>TIMER_CH_1</i>	TIMER channel 1 ( <i>TIMERx(x=0..4,7,8,11)</i> )
<i>TIMER_CH_2</i>	TIMER channel 2 ( <i>TIMERx(x=0..4,7)</i> )
<i>TIMER_CH_3</i>	TIMER channel 3 ( <i>TIMERx(x=0..4,7)</i> )
<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 3-685. Function timer\_channel\_complementary\_output\_polarity\_config**

<b>Function name</b>	timer_channel_complementary_output_polarity_config
<b>Function prototype</b>	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
<b>Function descriptions</b>	configure TIMER channel complementary output polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
channel	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>	
ocpolarity	channel complementary output polarity
<i>TIMER_OCN_POLARIT_Y_HIGH</i>	channel complementary output polarity is high
<i>TIMER_OCN_POLARIT_Y_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 3-686. 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
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<b>TIMER_CH_0</b>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<b>TIMER_CH_1</b>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<b>TIMER_CH_2</b>	TIMER channel 2 (TIMERx(x=0..4,7))
<b>TIMER_CH_3</b>	TIMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<b>state</b>	TIMER channel enable state
<b>TIMER_CCX_ENABLE</b>	channel enable
<b>TIMER_CCX_DISABLE</b>	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 3-687. Function timer\_channel\_complementary\_output\_state\_config**

<b>Function name</b>	timer_channel_complementary_output_state_config
<b>Function prototype</b>	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
<b>Function descriptions</b>	configure TIMER channel complementary output enable state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0
<i>TIMER_CH_1</i>	TIMER channel 1
<i>TIMER_CH_2</i>	TIMER channel 2
<b>Input parameter{in}</b>	
<b>state</b>	TIMER channel complementary output enable state
<i>TIMER_CCXN_ENABLE</i>	channel complementary enable
<i>TIMER_CCXN_DISABLE</i>	channel complementary disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure TIMER0 channel 0 complementary output enable state */

timer_channel_complementary_output_state_config (TIMER0, TIMER_CH_0,
TIMER_CCXN_ENABLE);
  
```

### **timer\_channel\_input\_struct\_para\_init**

The description of timer\_channel\_input\_struct\_para\_init is shown as below:

**Table 3-688. Function timer\_channel\_input\_struct\_para\_init**

<b>Function name</b>	timer_channel_input_struct_para_init
<b>Function prototype</b>	void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);
<b>Function descriptions</b>	initialize the parameters of TIMER channel input parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
icpara	TIMER channel intput parameter struct, the structure members can refer to <a href="#">Table 3-642. Structure timer_ic_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* initialize TIMER channel input parameter struct with a default value */

timer_ic_parameter_struct timer_icinitpara;

timer_channel_input_struct_para_init(timer_icinitpara);
  
```

### **timer\_input\_capture\_config**

The description of timer\_input\_capture\_config is shown as below:

**Table 3-689. Function timer\_input\_capture\_config**

<b>Function name</b>	timer_input_capture_config
----------------------	----------------------------

<b>Function prototype</b>	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);
<b>Function descriptions</b>	configure TIMER input capture parameter
<b>Precondition</b>	-
<b>The called functions</b>	timer_channel_input_capture_prescaler_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 ( <i>TIMERx</i> (x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 ( <i>TIMERx</i> (x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 ( <i>TIMERx</i> (x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 ( <i>TIMERx</i> (x=0..4,7))
<b>Input parameter{in}</b>	
<b>icpara</b>	TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-642. Structure timer_ic_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure TIMER0 input capture parameter */
timer_ic_parameter_struct timer_icinitpara;
timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;
timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);

```

### timer\_channel\_input\_capture\_prescaler\_config

The description of timer\_channel\_input\_capture\_prescaler\_config is shown as below:

**Table 3-690. Function timer\_channel\_input\_capture\_prescaler\_config**

<b>Function name</b>	timer_channel_input_capture_prescaler_config
<b>Function prototype</b>	void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler);
<b>Function descriptions</b>	configure TIMER channel input capture prescaler value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 ( <i>TIMERx(x=0..4,7..13)</i> )
<i>TIMER_CH_1</i>	TIMER channel 1 ( <i>TIMERx(x=0..4,7,8,11)</i> )
<i>TIMER_CH_2</i>	TIMER channel 2 ( <i>TIMERx(x=0..4,7)</i> )
<i>TIMER_CH_3</i>	TIMER channel 3 ( <i>TIMERx(x=0..4,7)</i> )
<b>Input parameter{in}</b>	
<b>prescaler</b>	channel input capture prescaler value
<i>TIMER_IC_PSC_DIV1</i>	no prescaler
<i>TIMER_IC_PSC_DIV2</i>	divided by 2
<i>TIMER_IC_PSC_DIV4</i>	divided by 4
<i>TIMER_IC_PSC_DIV8</i>	divided by 8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

/\* configure TIMER0 channel 0 input capture prescaler value \*/

```
timer_channel_input_capture_prescaler_config (TIMER0, TIMER_CH_0,
    TIMER_IC_PSC_DIV2);
```

### **timer\_channel\_capture\_value\_register\_read**

The description of timer\_channel\_capture\_value\_register\_read is shown as below:

**Table 3-691. Function timer\_channel\_capture\_value\_register\_read**

<b>Function name</b>	timer_channel_capture_value_register_read
<b>Function prototype</b>	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
<b>Function descriptions</b>	read TIMER channel capture compare register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<b>TIMER_CH_0</b>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<b>TIMER_CH_1</b>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<b>TIMER_CH_2</b>	TIMER channel 2 (TIMERx(x=0..4,7))
<b>TIMER_CH_3</b>	TIMER channel 3 (TIMERx(x=0..4,7))
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	channel capture compare register value (0x0000~0xFFFF)

Example:

```
/* read TIMER0 channel 0 capture compare register value */
uint32_t ch0_value = 0;
ch0_value = timer_channel_capture_value_register_read (TIMER0, TIMER_CH_0);
```

### timer\_input\_pwm\_capture\_config

The description of timer\_input\_pwm\_capture\_config is shown as below:

**Table 3-692. 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>	
timer_periph	TIMER peripheral
TIMERx(x=0..4,7,8,11)	TIMER peripheral selection
<b>Input parameter{in}</b>	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0
TIMER_CH_1	TIMER channel 1
<b>Input parameter{in}</b>	
icpwm	TIMER channel input pwm parameter struct, the structure members can refer to <a href="#">Table 3-642. Structure timer_ic_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure TIMER0 input pwm capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;

```

timer\_input\_pwm\_capture\_config (TIMER0, TIMER\_CH\_0, &timer\_icinitpara);

### **timer\_hall\_mode\_config**

The description of timer\_hall\_mode\_config is shown as below:

**Table 3-693. Function timer\_hall\_mode\_config**

<b>Function name</b>	timer_hall_mode_config
<b>Function prototype</b>	void timer_hall_mode_config(uint32_t timer_periph, uint8_t hallmode);
<b>Function descriptions</b>	configure TIMER hall sensor mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
timer_periph	TIMER peripheral
TIMERx(x=0..4,7)	TIMER peripheral selection
<b>Input parameter{in}</b>	
hallmode	TIMER hall sensor mode state
TIMER_HALLINTERFA CE_ENABLE	TIMER hall sensor mode enable
TIMER_HALLINTERFA CE_DISABLE	TIMER hall sensor mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 hall sensor mode */
timer_hall_mode_config (TIMER0, TIMER_HALLINTERFACE_ENABLE);
```

### **timer\_input\_trigger\_source\_select**

The description of timer\_input\_trigger\_source\_select is shown as below:

**Table 3-694. 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_EL_ITI0</i>	Internal trigger input 0 (ITI0, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS_EL_ITI1</i>	Internal trigger input 0 (ITI1, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS_EL_ITI2</i>	Internal trigger input 0 (ITI2, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS_EL_ITI3</i>	Internal trigger input 0 (ITI3, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS_EL_CI0F_ED</i>	CI0 edge flag (CI0F_ED, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS_EL_CI0FE0</i>	channel 0 input Filtered output (CI0FE0, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS_EL_CI1FE1</i>	channel 1 input Filtered output (CI1FE1, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS_EL_ETIFP</i>	External trigger input filter output(ETIFP, TIMERx(x=0..4,7) )
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select TIMER0 input trigger source */
```

---

```
timer_input_trigger_source_select (TIMER0, TIMER_SMCFG_TRGSEL_ITIO);
```

### **timer\_master\_output\_trigger\_source\_select**

The description of timer\_master\_output\_trigger\_source\_select is shown as below:

**Table 3-695. Function timer\_master\_output\_trigger\_source\_select**

<b>Function name</b>	timer_master_output_trigger_source_select
<b>Function prototype</b>	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
<b>Function descriptions</b>	select TIMER master mode output trigger source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>outrigger</b>	master mode control
<i>TIMER_TRI_OUT_SRC_RESET</i>	Reset. When the UPG bit in the TIMERx_SWEVG register is set or a reset is generated by the slave mode controller, a TRGO pulse occurs. And in the latter case, the signal on TRGO is delayed compared to the actual reset
<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 3-696. Function timer\_slave\_mode\_select**

<b>Function name</b>	timer_slave_mode_select
<b>Function prototype</b>	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
<b>Function descriptions</b>	select TIMER slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</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_ENCODER_MODE_DE0</i>	encoder mode 0
<i>TIMER_ENCODER_MODE_DE1</i>	encoder mode 1

<i>TIMER_ENCODER_MODE0</i>	encoder 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_ENCODER_MODE0);
```

#### **timer\_master\_slave\_mode\_config**

The description of timer\_master\_slave\_mode\_config is shown as below:

**Table 3-697. 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>	
<i>timer_periph</i>	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	

<b>masterslave</b>	master slave mode state
<i>TIMER_MASTER_SLAVE_MODE_ENABLE</i>	master slave mode enable
<i>TIMER_MASTER_SLAVE_MODE_DISABLE</i>	master slave mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 master slave mode */
timer_master_slave_mode_config (TIMER0, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

#### **timer\_external\_trigger\_config**

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

**Table 3-698. Function timer\_external\_trigger\_config**

<b>Function name</b>	timer_external_trigger_config
<b>Function prototype</b>	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER external trigger input
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2

<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
<b>Input parameter{in}</b>	
<b>expolarity</b>	external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
<b>Input parameter{in}</b>	
<b>extfilter</b>	external trigger filter control (0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 external trigger input */

timer_external_trigger_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TIMER_ETP_FALLING, 10);
```

### **timer\_quadrature\_decoder\_mode\_config**

The description of timer\_quadrature\_decoder\_mode\_config is shown as below:

**Table 3-699. Function timer\_quadrature\_decoder\_mode\_config**

<b>Function name</b>	timer_quadrature_decoder_mode_config
<b>Function prototype</b>	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity);
<b>Function descriptions</b>	configure TIMER quadrature decoder mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral

<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>decomode</b>	quadrature decoder mode
<i>TIMER_ENCODER_MODE0</i>	counter counts on CI0FE0 edge depending on CI1FE1 level
<i>TIMER_ENCODER_MODE1</i>	counter counts on CI1FE1 edge depending on CI0FE0 level
<i>TIMER_ENCODER_MODE2</i>	counter counts on both CI0FE0 and CI1FE1 edges depending on the level of the other input
<b>Input parameter{in}</b>	
<b>ic0polarity</b>	IC0 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
<b>Input parameter{in}</b>	
<b>ic1polarity</b>	IC1 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 quadrature decoder mode */
timer_quadrature_decoder_mode_config (TIMER0, TIMER_ENCODER_MODE0,
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

### **timer\_internal\_clock\_config**

The description of timer\_internal\_clock\_config is shown as below:

**Table 3-700. 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 3-701. Function timer\_internal\_trigger\_as\_external\_clock\_config**

<b>Function name</b>	timer_internal_trigger_as_external_clock_config
<b>Function prototype</b>	void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);
<b>Function descriptions</b>	configure TIMER the internal trigger as external clock input
<b>Precondition</b>	-
<b>The called functions</b>	timer_input_trigger_source_select
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	

<b>intrigger</b>	trigger selection
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI0</i>	Internal trigger input 0 (ITI0)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI1</i>	Internal trigger input 0 (ITI1)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI2</i>	Internal trigger input 0 (ITI2)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI3</i>	Internal trigger input 0 (ITI3)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 the internal trigger ITI0 as external clock input */
timer_internal_trigger_as_external_clock_config(TIMER0, TIMER_SMCFG_TRGSEL_ITI0);

timer_external_trigger_as_external_clock_config
```

The description of `timer_external_trigger_as_external_clock_config` is shown as below:

**Table 3-702. 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 exttrigger, 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
<b>Input parameter{in}</b>	
<b>exttrigger</b>	external trigger selection

<i>TIMER_SMCFG_TRGS</i> <i>EL_CI0F_ED</i>	CI0 edge flag (CI0F_ED)
<i>TIMER_SMCFG_TRGS</i> <i>EL_CI0FE0</i>	channel 0 input Filtered output (CI0FE0)
<i>TIMER_SMCFG_TRGS</i> <i>EL_CI1FE1</i>	channel 1 input Filtered output (CI1FE1)
<b>Input parameter{in}</b>	
<b>expolarity</b>	external trigger polarity
<i>TIMER_IC_POLARITY_RISING</i>	active high or rising edge active
<i>TIMER_IC_POLARITY_FALLING</i>	active low or falling edge active
<b>Input parameter{in}</b>	
<b>extfilter</b>	external trigger filter control (0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 the external trigger CI0FE0 as external clock input */

timer_external_trigger_as_external_clock_config (TIMER0,
TIMER_SMCFG_TRGSEL_CI0FE0, TIMER_IC_POLARITY_RISING, 0);
```

#### **timer\_external\_clock\_mode0\_config**

The description of timer\_external\_clock\_mode0\_config is shown as below:

**Table 3-703. Function timer\_external\_clock\_mode0\_config**

<b>Function name</b>	timer_external_clock_mode0_config
<b>Function prototype</b>	void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external clock mode0
<b>Precondition</b>	-

<b>The called functions</b>		timer_external_trigger_config
<b>Input parameter{in}</b>		
<b>timer_periph</b>		TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>		TIMER peripheral selection
<b>Input parameter{in}</b>		
<b>extprescaler</b>		ETI external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>		no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>		divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>		divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>		divided by 8
<b>Input parameter{in}</b>		
<b>expolarity</b>		ETI external trigger polarity
<i>TIMER_ETP_FALLING</i>		active low or falling edge active
<i>TIMER_ETP_RISING</i>		active high or rising edge active
<b>Input parameter{in}</b>		
<b>extfilter</b>		ETI external trigger filter control (0~15)
<b>Output parameter{out}</b>		
-		-
<b>Return value</b>		
-		-

Example:

```
/* configure TIMER0 the external clock mode0 */
timer_external_clock_mode0_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TIMER_ETP_FALLING, 0);
```

### **timer\_external\_clock\_mode1\_config**

The description of timer\_external\_clock\_mode1\_config is shown as below:

Table 3-704. 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 3-705. 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>	
timer_periph	TIMER peripheral
TIMERx(x=0..4,7)	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 3-706. 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_DISA BLE</i>	no effect
<i>TIMER_CHVSEL_ENAB LE</i>	when write the CHxVAL register, if the write value is same as the CHxVAL value, the write access is ignored
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 write CHxVAL register selection */
timer_write_chxval_register_config(TIMER0, TIMER_CHVSEL_ENABLE);
```

#### **timer\_output\_value\_selection\_config**

The description of **timer\_output\_value\_selection\_config** is shown as below:

**Table 3-707. Function timer\_output\_value\_selection\_config**

<b>Function name</b>	timer_output_value_selection_config
<b>Function prototype</b>	void timer_output_value_selection_config(uint32_t timer_periph, uint16_t outsel);
<b>Function descriptions</b>	configure TIMER output value selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral

<i>TIMERx (x=0..7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>outsel</b>	output value selection
<i>TIMER_OUTSEL_DISABLE</i>	no effect
<i>TIMER_OUTSEL_ENABLE</i>	if POEN and IOS is 0, the output disabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER output value selection */

timer_output_value_selection_config(TIMER0, TIMER_OUTSEL_ENABLE);
```

### **timer\_interrupt\_enable**

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

**Table 3-708. Function timer\_interrupt\_enable**

<b>Function name</b>	timer_interrupt_enable
<b>Function prototype</b>	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable the TIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	timer interrupt enable source
<i>TIMER_INT_UP</i>	update interrupt enable, TIMERx (x=0..13)
<i>TIMER_INT_CH0</i>	channel 0 interrupt enable, TIMERx (x=0..4,7..13)

<i>TIMER_INT_CH1</i>	channel 1 interrupt enable, TIMERx (x=0..4,7,8,11)
<i>TIMER_INT_CH2</i>	channel 2 interrupt enable, TIMERx (x=0..4,7)
<i>TIMER_INT_CH3</i>	channel 3 interrupt enable , TIMERx (x=0..4,7)
<i>TIMER_INT_CMT</i>	commutation interrupt enable, TIMERx (x=0,7)
<i>TIMER_INT_TRG</i>	trigger interrupt enable, TIMERx (x=0..4,7,8,11)
<i>TIMER_INT_BRK</i>	break interrupt enable, TIMERx (x=0,7)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the TIMER0 update interrupt */

timer_interrupt_enable (TIMER0, TIMER_INT_UP);
```

### **timer\_interrupt\_disable**

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

**Table 3-709. Function timer\_interrupt\_disable**

<b>Function name</b>	timer_interrupt_disable
<b>Function prototype</b>	void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	disable the TIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	timer interrupt disable source
<i>TIMER_INT_UP</i>	update interrupt disable, TIMERx(x=0..13)
<i>TIMER_INT_CH0</i>	channel 0 interrupt disable, TIMERx(x=0..4,7..13)

<i>TIMER_INT_CH1</i>	channel 1 interrupt disable, TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_CH2</i>	channel 2 interrupt disable, TIMERx(x=0..4,7)
<i>TIMER_INT_CH3</i>	channel 3 interrupt disable, TIMERx(x=0..4,7)
<i>TIMER_INT_CMT</i>	commutation interrupt disable, TIMERx(x=0,7)
<i>TIMER_INT_TRG</i>	trigger interrupt disable, TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_BRK</i>	break interrupt disable, TIMERx(x=0,7)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the TIMER0 update interrupt */

timer_interrupt_disable (TIMER0, TIMER_INT_UP);
```

### **timer\_interrupt\_flag\_get**

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

**Table 3-710. Function timer\_interrupt\_flag\_get**

<b>Function name</b>	timer_interrupt_flag_get
<b>Function prototype</b>	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	get timer interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>timer_periph</i>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<i>interrupt</i>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag,TIMERx(x=0..13)

<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag,TIMERx(x=0..4,7..13)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag,TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag,TIMERx(x=0..4,7)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag,TIMERx(x=0..4,7)
<i>TIMER_INT_FLAG_CM_T</i>	channel commutation interrupt flag,TIMERx(x=0,7)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag,TIMERx(x=0,7,8,11)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag,TIMERx(x=0,7)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get 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 3-711. Function `timer_interrupt_flag_clear`**

<b>Function name</b>	timer_interrupt_flag_clear
<b>Function prototype</b>	void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	clear TIMER interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	

<b>interrupt</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag,TIMERx(x=0..13)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag,TIMERx(x=0..4,7..13)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag,TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag,TIMERx(x=0..4,7)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag,TIMERx(x=0..4,7)
<i>TIMER_INT_FLAG_CM_T</i>	channel commutation interrupt flag,TIMERx(x=0,7)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag,TIMERx(x=0,7,8,11)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag,TIMERx(x=0,7)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TIMER0 update interrupt flag */
timer_interrupt_flag_clear (TIMER0, TIMER_INT_FLAG_UP);
```

### **timer\_flag\_get**

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

**Table 3-712. 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

Input parameter{in}	
<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)
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TIMER0 update flags */

FlagStatus Flag_status = RESET;

Flag_status = timer_flag_get (TIMER0, TIMER_FLAG_UP);
```

### **timer\_flag\_clear**

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

**Table 3-713. Function timer\_flag\_clear**

<b>Function name</b>	timer_flag_clear
<b>Function prototype</b>	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
<b>Function descriptions</b>	clear TIMER flags

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>flag</b>	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag,TIMERx(x=0..13)
<i>TIMER_FLAG_CH0</i>	channel 0 flag,TIMERx(x=0..4,7..13)
<i>TIMER_FLAG_CH1</i>	channel 1 flag,TIMERx(x=0..4,7,8,11)
<i>TIMER_FLAG_CH2</i>	channel 2 flag,TIMERx(x=0..4,7)
<i>TIMER_FLAG_CH3</i>	channel 3 flag,TIMERx(x=0..4,7)
<i>TIMER_FLAG_CMT</i>	channel commutation flag,TIMERx(x=0,7)
<i>TIMER_FLAG_TRG</i>	trigger flag,TIMERx(x=0,7,8,11)
<i>TIMER_FLAG_BRK</i>	break flag,TIMERx(x=0,7)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag,TIMERx(x=0..4,7..11)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag,TIMERx(x=0..4,7,8,11)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag,TIMERx(x=0..4,7)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag,TIMERx(x=0..4,7)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TIMER0 update flags */

timer_flag_clear (TIMER0, TIMER_FLAG_UP);
```

## 3.24. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter [3.24.1](#), the USART firmware functions are introduced in chapter [3.24.2](#).

### 3.24.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

**Table 3-714. USART Registers**

Registers	Descriptions
USART_STAT0	Status register 0
USART_DATA	Data register
USART_BAUD	Baud rate register
USART_CTL0	Control register 0
USART_CTL1	Control register 1
USART_CTL2	Control register 2
USART_GP	Guard time and prescaler register
USART_CTL3	Control register 3
USART_RT	Receiver timeout register
USART_STAT1	Status register 1

### 3.24.2. Descriptions of Peripheral functions

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

**Table 3-715. USART firmware function**

Function name	Function description
uart_deinit	reset USART/UART
uart_baudrate_set	configure USART baud rate value
uart_parity_config	configure USART parity
uart_word_length_set	configure USART word length
uart_stop_bit_set	configure USART stop bit length

Function name	Function description
uart_enable	enable USART
uart_disable	disable USART
uart_transmit_config	configure USART transmitter
uart_receive_config	configure USART receiver
uart_data_first_config	data is transmitted/received with the LSB/MSB first
uart_invert_config	configure USART inverted
uart_receiver_timeout_enable	enable receiver timeout
uart_receiver_timeout_disable	disable receiver timeout
uart_receiver_timeout_threshold_config	configure receiver timeout threshold
uart_data_transmit	USART transmit data function
uart_data_receive	USART receive data function
uart_address_config	configure the address of the USART in wake up by address match mode
uart_mute_mode_enable	enable mute mode
uart_mute_mode_disable	disable mute mode
uart_mute_mode_wakeup_config	configure wakeup method in mute mode
uart_lin_mode_enable	enable LIN mode
uart_lin_mode_disable	disable LIN mode
uart_lin_break_dectcion_length_config	configure LIN break frame length
uart_send_break	send break frame
uart_halfduplex_enable	enable half duplex mode
uart_halfduplex_disable	disable half duplex mode
uart_synchronous_clock_enable	enable CK pin in synchronous mode
uart_synchronous_clock_disable	disable CK pin in synchronous mode
uart_synchronous_clock_config	configure USART synchronous mode parameters
uart_guard_time_config	configure guard time value in smartcard mode

Function name	Function description
uart_smartcard_mode_enable	enable smartcard mode
uart_smartcard_mode_disable	disable smartcard mode
uart_smartcard_mode_nack_enable	enable NACK in smartcard mode
uart_smartcard_mode_nack_disable	disable NACK in smartcard mode
uart_smartcard_autoretry_config	configure smartcard auto-retry number
uart_block_length_config	configure block length
uart_irda_mode_enable	enable IrDA mode
uart_irda_mode_disable	disable IrDA mode
uart_prescaler_config	configure the peripheral clock prescaler in USART IrDA low-power mode
uart_irda_lowpower_config	configure IrDA low-power
uart_hardware_flow_rts_config	configure hardware flow control RTS
uart_hardware_flow_cts_config	configure hardware flow control CTS
uart_dma_receive_config	configure USART DMA reception
uart_dma_transmit_config	configure USART DMA transmission
uart_flag_get	get flag in STAT0/STAT1 register
uart_flag_clear	clear flag in STAT0/STAT1 register
uart_interrupt_enable	enable USART interrupt
uart_interrupt_disable	disable USART interrupt
uart_interrupt_flag_get	get USART interrupt flag status
uart_interrupt_flag_clear	clear USART interrupt flag

## uart\_deinit

The description of `uart_deinit` is shown as below:

**Table 3-716. Function `uart_deinit`**

Function name	uart_deinit
Function prototype	void uart_deinit(uint32_t usart_periph);
Function descriptions	reset USART/UART

<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>uart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset USART0 */

uart_deinit (USART0);
```

### **uart\_baudrate\_set**

The description of `uart_baudrate_set` is shown as below:

**Table 3-717. Function `uart_baudrate_set`**

<b>Function name</b>	uart_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>uart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>baudval</b>	baud rate value
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* configure USART0 baud rate value */
uart_baudrate_set(USART0, 115200);
```

### uart\_parity\_config

The description of `uart_parity_config` is shown as below:

**Table 3-718. Function `uart_parity_config`**

<b>Function name</b>	uart_parity_config
<b>Function prototype</b>	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
<b>Function descriptions</b>	configure USART parity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>paritycfg</b>	configure USART parity
<i>USART_PM_NONE</i>	no parity
<i>USART_PM_ODD</i>	odd parity
<i>USART_PM_EVEN</i>	even parity
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART parity */
uart_parity_config(USART0, USART_PM EVEN);
```

### **uart\_word\_length\_set**

The description of `uart_word_length_set` is shown as below:

**Table 3-719. Function `uart_word_length_set`**

<b>Function name</b>	uart_word_length_set
<b>Function prototype</b>	void <code>uart_word_length_set(uint32_t usart_periph, uint32_t wlen)</code> ;
<b>Function descriptions</b>	configure USART word length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>usart_periph</code>	USARTx/UARTx peripheral
<code>USARTx</code>	x=0,1,2
<code>UARTx</code>	x=3,4
<b>Input parameter{in}</b>	
<code>wlen</code>	USART word length
<code>USART_WL_8BIT</code>	8 bits
<code>USART_WL_9BIT</code>	9 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 word length */
uart_word_length_set(USART0, USART_WL_9BIT);
```

### **uart\_stop\_bit\_set**

The description of `uart_stop_bit_set` is shown as below:

**Table 3-720. Function usart\_stop\_bit\_set**

<b>Function name</b>	usart_stop_bit_set
<b>Function prototype</b>	void usart_stop_bit_set(uint32_t usart_periph, uint32_t strlen);
<b>Function descriptions</b>	configure USART stop bit length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>strlen</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 3-721. 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>uart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 */
uart_enable(USART0);
```

### **uart\_disable**

The description of **uart\_disable** is shown as below:

**Table 3-722. Function **uart\_disable****

<b>Function name</b>	uart_disable
<b>Function prototype</b>	void uart_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>uart_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:

```
/* disable USART0 */
uart_disable(USART0);
```

### uart\_transmit\_config

The description of `uart_transmit_config` is shown as below:

**Table 3-723. Function `uart_transmit_config`**

<b>Function name</b>	uart_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
<b>USARTx</b>	x=0,1,2
<b>UARTx</b>	x=3,4
<b>Input parameter{in}</b>	
<b>txconfig</b>	enable or disable USART transmitter
<b>USART_TRANSMIT_ENABLE</b>	enable USART transmission
<b>USART_TRANSMIT_DISABLE</b>	enable USART transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 transmitter */
```

```
uart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

### **uart\_receive\_config**

The description of `uart_receive_config` is shown as below:

**Table 3-724. Function `uart_receive_config`**

<b>Function name</b>	uart_receive_config
<b>Function prototype</b>	void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);
<b>Function descriptions</b>	configure USART receiver
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>rxconfig</b>	enable or disable USART receiver
<i>USART_RECEIVE_ENABLE</i>	enable USART reception
<i>USART_RECEIVE_DISABLE</i>	disable USART reception
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 receiver */

uart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

### **uart\_data\_first\_config**

The description of `uart_data_first_config` is shown as below:

**Table 3-725. 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
<b>USARTx</b>	x=0,1,2
<b>Input parameter{in}</b>	
<b>msbf</b>	LSB first or MSB first
<b>USART_MSBF_LSB</b>	LSB first
<b>USART_MSBF_MSB</b>	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 3-726. 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>	-

Input parameter{in}	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
Input parameter{in}	
<b>invertpara</b>	refer to enum usart_invert_enum
<i>USART_DINV_ENABLE</i> <i>E</i>	data bit level inversion
<i>USART_DINV_DISABLE</i> <i>E</i>	data bit level not inversion
<i>USART_TXPIN_ENABLE</i> <i>LE</i>	TX pin level inversion
<i>USART_TXPIN_DISABLE</i> <i>LE</i>	TX pin level not inversion
<i>USART_RXPIN_ENABLE</i> <i>LE</i>	RX pin level inversion
<i>USART_RXPIN_DISABLE</i> <i>LE</i>	RX pin level not inversion
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART inversion */
uart_invert_config(USART0, USART_DINV_ENABLE);
```

#### **uart\_receiver\_timeout\_enable**

The description of `uart_receiver_timeout_enable` is shown as below:

**Table 3-727. Function `uart_receiver_timeout_enable`**

<b>Function name</b>	uart_receiver_timeout_enable
<b>Function prototype</b>	void <code>uart_receiver_timeout_enable(uint32_t usart_periph);</code>
<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
<b>USARTx</b>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable receiver timeout of USART */
uart_receiver_timeout_enable(USART0);
```

#### **uart\_receiver\_timeout\_disable**

The description of `uart_receiver_timeout_disable` is shown as below:

**Table 3-728. Function `uart_receiver_timeout_disable`**

<b>Function name</b>	uart_receiver_timeout_disable
<b>Function prototype</b>	void uart_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
<b>USARTx</b>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable receiver timeout of USART */
uart_receiver_timeout_disable(USART0);
```

### **uart\_receiver\_timeout\_threshold\_config**

The description of `uart_receiver_timeout_threshold_config` is shown as below:

**Table 3-729. Function `uart_receiver_timeout_threshold_config`**

<b>Function name</b>	uart_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>	
usart_periph	USARTx peripheral
USARTx	x=0,1,2
<b>Input parameter{in}</b>	
rtimeout	timeout value
0-0xFFFFFFF	timeout value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the receiver timeout threshold of USART0 */
uart_receiver_timeout_threshold_config(USART0, 115200*3);
```

### **uart\_data\_transmit**

The description of `uart_data_transmit` is shown as below:

**Table 3-730. Function `uart_data_transmit`**

<b>Function name</b>	uart_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
<b>USARTx</b>	x=0,1,2
<b>UARTx</b>	x=3,4
<b>Input parameter{in}</b>	
<b>data</b>	data of transmission
<b>0-0x1FF</b>	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 3-731. Function usart\_data\_receive**

<b>Function name</b>	usart_data_receive
<b>Function prototype</b>	void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);
<b>Function descriptions</b>	USART receive data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral

<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<i>uint32_t</i>	data of received(0-0xFF)

Example:

```
/* USART0 receive data */

uint16_t temp;

temp = usart_data_receive(USART0);
```

### **usart\_address\_config**

The description of `usart_address_config` is shown as below:

**Table 3-732. 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
<i>0-0xFF</i>	address of USART/UART
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* configure address of the USART0 */
uart_address_config(USART0, 0x00);
```

### **uart\_mute\_mode\_enable**

The description of `uart_mute_mode_enable` is shown as below:

**Table 3-733. Function `uart_mute_mode_enable`**

<b>Function name</b>	uart_mute_mode_enable
<b>Function prototype</b>	void usart_mute_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable mute mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 receiver in mute mode */
uart_mute_mode_enable(USART0);
```

### **uart\_mute\_mode\_disable**

The description of `uart_mute_mode_disable` is shown as below:

**Table 3-734. Function `uart_mute_mode_disable`**

<b>Function name</b>	uart_mute_mode_disable
<b>Function prototype</b>	void usart_mute_mode_disable(uint32_t usart_periph);

<b>Function descriptions</b>	disable mute mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 receiver in mute mode */

uart_mute_mode_disable(USART0);
```

#### **uart\_mute\_mode\_wakeup\_config**

The description of **uart\_mute\_mode\_wakeup\_config** is shown as below:

**Table 3-735. Function **uart\_mute\_mode\_wakeup\_config****

<b>Function name</b>	uart_mute_mode_wakeup_config
<b>Function prototype</b>	void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);
<b>Function descriptions</b>	configure wakeup method in mute mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<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 */
uart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

### **uart\_lin\_mode\_enable**

The description of `uart_lin_mode_enable` is shown as below:

**Table 3-736. Function `uart_lin_mode_enable`**

<b>Function name</b>	uart_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
<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 LIN mode enable */
```

```
uart_lin_mode_enable(USART0);
```

### **uart\_lin\_mode\_disable**

The description of `uart_lin_mode_disable` is shown as below:

**Table 3-737. Function `uart_lin_mode_disable`**

<b>Function name</b>	uart_lin_mode_disable
<b>Function prototype</b>	void usart_lin_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable LIN mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 LIN mode disable */

uart_lin_mode_disable(USART0);
```

### **uart\_lin\_break\_decton\_length\_config**

The description of `uart_lin_break_decton_length_config` is shown as below:

**Table 3-738. Function `uart_lin_break_decton_length_config`**

<b>Function name</b>	uart_lin_break_decton_length_config
<b>Function prototype</b>	void usart_lin_break_decton_length_config(uint32_t usart_periph, uint32_t lrlen);
<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
<b>USARTx</b>		x=0,1,2
<b>UARTx</b>		x=3,4
<b>Input parameter{in}</b>		
<b>Iblen</b>		two methods be used to enter or exit the mute mode
<b>USART_LBLEN_10B</b>		break frame length is 10 bits
<b>USART_LBLEN_11B</b>		break frame length is 11 bits
<b>Output parameter{out}</b>		
-		-
<b>Return value</b>		
-		-

Example:

```
/* configure LIN break frame length */
uart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

### **uart\_send\_break**

The description of `uart_send_break` is shown as below:

**Table 3-739. Function `uart_send_break`**

<b>Function name</b>	uart_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
<b>USARTx</b>	x=0,1,2
<b>UARTx</b>	x=3,4

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 send break frame */
uart_send_break(USART0);
```

### **uart\_halfduplex\_enable**

The description of `uart_halfduplex_enable` is shown as below:

**Table 3-740. Function `uart_halfduplex_enable`**

<b>Function name</b>	uart_halfduplex_enable
<b>Function prototype</b>	void usart_halfduplex_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable half duplex mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 half duplex mode*/
uart_halfduplex_enable(USART0);
```

### **uart\_halfduplex\_disable**

The description of `uart_halfduplex_disable` is shown as below:

**Table 3-741. Function usart\_halfduplex\_disable**

<b>Function name</b>	usart_halfduplex_disable
<b>Function prototype</b>	void usart_halfduplex_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable half duplex mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 half duplex mode*/
usart_halfduplex_disable(USART0);
```

#### **usart\_synchronous\_clock\_enable**

The description of usart\_synchronous\_clock\_enable is shown as below:

**Table 3-742. Function usart\_synchronous\_clock\_enable**

<b>Function name</b>	usart_synchronous_clock_enable
<b>Function prototype</b>	void usart_synchronous_clock_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable CK pin in synchronous mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 CK pin in synchronous mode */
uart_synchronous_clock_enable(USART0);
```

### **uart\_synchronous\_clock\_disable**

The description of `uart_synchronous_clock_disable` is shown as below:

**Table 3-743. Function `uart_synchronous_clock_disable`**

<b>Function name</b>	uart_synchronous_clock_disable
<b>Function prototype</b>	void uart_synchronous_clock_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable CK pin in synchronous mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	USARTx peripheral
<b>USARTx</b>	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 CK pin in synchronous mode */
uart_synchronous_clock_disable(USART0);
```

### **uart\_synchronous\_clock\_config**

The description of `uart_synchronous_clock_config` is shown as below:

Table 3-744. Function usart\_synchronous\_clock\_config

<b>Function name</b>	usart_synchronous_clock_config
<b>Function prototype</b>	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
<b>Function descriptions</b>	configure USART synchronous mode parameters
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<b>USARTx</b>	x=0,1,2
<b>Input parameter{in}</b>	
<b>clen</b>	CK length
<b>USART_CLEN_NONE</b>	there are 7 CK pulses for an 8 bit frame and 8 CK pulses for a 9 bit frame
<b>USART_CLEN_EN</b>	there are 8 CK pulses for an 8 bit frame and 9 CK pulses for a 9 bit frame
<b>Input parameter{in}</b>	
<b>cph</b>	clock phase
<b>USART_CPH_1CK</b>	first clock transition is the first data capture edge
<b>USART_CPH_2CK</b>	second clock transition is the first data capture edge
<b>Input parameter{in}</b>	
<b>cpl</b>	clock polarity
<b>USART_CPL_LOW</b>	steady low value on CK pin
<b>USART_CPL_HIGH</b>	steady high value on CK pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 synchronous mode parameters */

usart_synchronous_clock_config(USART0,USART_CLEN_EN,USART_CPH_2CK,
```

USART\_CPL\_HIGH);

### **uart\_guard\_time\_config**

The description of usart\_guard\_time\_config is shown as below:

**Table 3-745. 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
<b>USARTx</b>	x=0,1,2
<b>Input parameter{in}</b>	
<b>guat</b>	guard time value
<b>value</b>	0-0xFF
<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);
```

### **uart\_smartcard\_mode\_enable**

The description of usart\_smartcard\_mode\_enable is shown as below:

**Table 3-746. Function usart\_smartcard\_mode\_enable**

<b>Function name</b>	usart_smartcard_mode_enable
<b>Function prototype</b>	void usart_smartcard_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable smartcard mode

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 smartcard mode enable */
uart_smartcard_mode_enable(USART0);
```

### **uart\_smartcard\_mode\_disable**

The description of `uart_smartcard_mode_disable` is shown as below:

**Table 3-747. Function `uart_smartcard_mode_disable`**

<b>Function name</b>	uart_smartcard_mode_disable
<b>Function prototype</b>	void uart_smartcard_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 smartcard mode disable */

uart_smartcard_mode_disable(USART0);
```

### **uart\_smartcard\_mode\_nack\_enable**

The description of `uart_smartcard_mode_nack_enable` is shown as below:

**Table 3-748. Function `uart_smartcard_mode_nack_enable`**

<b>Function name</b>	uart_smartcard_mode_nack_enable
<b>Function prototype</b>	void <code>uart_smartcard_mode_nack_enable(uint32_t usart_periph);</code>
<b>Function descriptions</b>	enable NACK in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>usart_periph</code>	USARTx peripheral
<code>USARTx</code>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 NACK in smartcard mode */

uart_smartcard_mode_nack_enable(USART0);
```

### **uart\_smartcard\_mode\_nack\_disable**

The description of `uart_smartcard_mode_nack_disable` is shown as below:

**Table 3-749. Function `uart_smartcard_mode_nack_disable`**

<b>Function name</b>	uart_smartcard_mode_nack_disable
<b>Function prototype</b>	void <code>uart_smartcard_mode_nack_disable(uint32_t usart_periph);</code>
<b>Function descriptions</b>	disable NACK in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>usart_periph</b>	USARTx peripheral
<b>USARTx</b>	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 NACK in smartcard mode */
uart_smartcard_mode_nack_disable(USART0);
```

### **uart\_smartcard\_autoretry\_config**

The description of `uart_smartcard_autoretry_config` is shown as below:

**Table 3-750. Function `uart_smartcard_autoretry_config`**

<b>Function name</b>	uart_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>	-
Input parameter{in}	
<b>usart_periph</b>	USARTx peripheral
<b>USARTx</b>	x=0,1,2
Input parameter{in}	
<b>scrtnum</b>	smartcard auto-retry number
<b>0-0xFFFFFFFF</b>	smartcard auto-retry number
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* configure smartcard auto-retry number */

uart_smartcard_autoretry_config (USART0, 0xFFFFFFFF);
```

### uart\_block\_length\_config

The description of `uart_block_length_config` is shown as below:

**Table 3-751. Function `uart_block_length_config`**

<b>Function name</b>	uart_block_length_config
<b>Function prototype</b>	void uart_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
<b>USARTx</b>	x=0,1,2
<b>Input parameter{in}</b>	
<b>bl</b>	block length
<b>0-0xFFFFFFFF</b>	block length
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure block length in Smartcard T=1 reception */

uart_block_length_config(USART0, 0xFFFFFFFF);
```

### uart\_irda\_mode\_enable

The description of `uart_irda_mode_enable` is shown as below:

**Table 3-752. Function usart\_irda\_mode\_enable**

<b>Function name</b>	usart_irda_mode_enable
<b>Function prototype</b>	void usart_irda_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable IrDA mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 IrDA mode */
usart_irda_mode_enable(USART0);
```

### **usart\_irda\_mode\_disable**

The description of usart\_irda\_mode\_disable is shown as below:

**Table 3-753. Function usart\_irda\_mode\_disable**

<b>Function name</b>	usart_irda_mode_disable
<b>Function prototype</b>	void usart_irda_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable IrDA mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2

<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 IrDA mode */

uart_irda_mode_disable(USART0);
```

### **uart\_prescaler\_config**

The description of `uart_prescaler_config` is shown as below:

**Table 3-754. Function `uart_prescaler_config`**

<b>Function name</b>	uart_prescaler_config
<b>Function prototype</b>	void usart_prescaler_config(uint32_t usart_periph, uint8_t psc);
<b>Function descriptions</b>	configure the peripheral clock prescaler in USART IrDA low-power mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>psc</b>	0x00-0xFF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the USART0 peripheral clock prescaler */
```

```
uart_prescaler_config(USART0, 0x00);
```

### **uart\_irda\_lowpower\_config**

The description of `uart_irda_lowpower_config` is shown as below:

**Table 3-755. Function `uart_irda_lowpower_config`**

<b>Function name</b>	uart_irda_lowpower_config
<b>Function prototype</b>	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
<b>Function descriptions</b>	configure IrDA low-power
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>irlp</b>	IrDA low-power or normal
<i>USART_IRLP_LOW</i>	low-power
<i>USART_IRLP_NORMAL</i>	normal
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 IrDA low-power */
uart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

### **uart\_hardware\_flow\_rts\_config**

The description of `uart_hardware_flow_rts_config` is shown as below:

**Table 3-756. 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
<b>USARTx</b>	x=0,1,2
<b>Input parameter{in}</b>	
<b>rtsconfig</b>	enable or disable RTS
<b>USART_RTS_ENABLE</b>	enable RTS
<b>USART_RTS_DISABLE</b>	disable RTS
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 hardware flow control RTS */
usart_hardware_flow_cts_config(USART0, USART_RTS_ENABLE);
```

### **usart\_hardware\_flow\_cts\_config**

The description of usart\_hardware\_flow\_cts\_config is shown as below:

**Table 3-757. 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>uart_periph</b>	USARTx peripheral
<b>USARTx</b>	x=0,1,2
<b>Input parameter{in}</b>	
<b>ctsconfig</b>	enable or disable CTS
<b>USART_CTS_ENABLE</b>	enable CTS
<b>USART_CTS_DISABLE</b>	disable CTS
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 hardware flow control CTS */
uart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

### **uart\_dma\_receive\_config**

The description of `uart_dma_receive_config` is shown as below:

**Table 3-758. Function `uart_dma_receive_config`**

<b>Function name</b>	uart_dma_receive_config
<b>Function prototype</b>	void <code>uart_dma_receive_config(uint32_t usart_periph, uint32_t dmacmd)</code> ;
<b>Function descriptions</b>	configure USART DMA reception
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uart_periph</b>	USARTx/UARTx peripheral
<b>USARTx</b>	x=0,1,2

<i>UARTx</i>	x=3
<b>Input parameter{in}</b>	
<b>dmacmd</b>	enable or disable DMA for reception
<i>USART_DENR_ENABLE</i>	DMA enable for reception
<i>USART_DENR_DISABLE</i>	DMA disable for reception
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 DMA enable for reception */
uart_dma_receive_config(USART0, USART_DENR_ENABLE);
```

### **uart\_dma\_transmit\_config**

The description of `uart_dma_transmit_config` is shown as below:

**Table 3-759. Function `uart_dma_transmit_config`**

<b>Function name</b>	uart_dma_transmit_config
<b>Function prototype</b>	void <code>uart_dma_transmit_config(uint32_t usart_periph, uint32_t dmacmd);</code>
<b>Function descriptions</b>	configure USART DMA transmission
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3
<b>Input parameter{in}</b>	
<b>dmacmd</b>	enable or disable DMA for transmission
<i>USART_DENT_ENABLE</i>	DMA enable for transmission

<i>E</i>	
<i>USART_DENT_DISAB LE</i>	DMA disable for transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 DMA enable for transmission */

uart_dma_transmit_config(USART0, USART_DENT_ENABLE);
```

### uart\_flag\_get

The description of `uart_flag_get` is shown as below:

**Table 3-760. Function `uart_flag_get`**

<b>Function name</b>	uart_flag_get
<b>Function prototype</b>	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
<b>Function descriptions</b>	get flag in STAT0/STAT1 register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags, refer to <code>usart_flag_enum</code>
<i>USART_FLAG_CTS</i>	CTS change flag
<i>USART_FLAG_LBD</i>	LIN break detected flag
<i>USART_FLAG_TBE</i>	transmit data buffer empty flag
<i>USART_FLAG_TC</i>	transmission complete flag
<i>USART_FLAG_RBNE</i>	read data buffer not empty flag

<i>USART_FLAG_IDLE</i>	IDLE frame detected flag
<i>USART_FLAG_ORER</i> <i>R</i>	overrun error flag
<i>USART_FLAG_NERR</i>	noise error flag
<i>USART_FLAG_FERR</i>	frame error flag
<i>USART_FLAG_PERR</i>	parity error flag
<i>USART_FLAG_BSY</i>	busy flag
<i>USART_FLAG_EB</i>	end of block flag
<i>USART_FLAG_RT</i>	receiver timeout flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get flag USART0 state */

FlagStatus status;

status = usart_flag_get(USART0,USART_FLAG_TBE);
```

### **usart\_flag\_clear**

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

**Table 3-761. 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
<b>USARTx</b>	x=0,1,2

<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags, refer to usart_flag_enum
<i>USART_FLAG_CTS</i>	CTS change flag
<i>USART_FLAG_LBD</i>	LIN break detected flag
<i>USART_FLAG_TC</i>	transmission complete
<i>USART_FLAG_RBNE</i>	read data buffer not empty
<i>USART_FLAG_EB</i>	end of block flag
<i>USART_FLAG_RT</i>	receiver timeout flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear USART0 flag */
uart_flag_clear(USART0,USART_FLAG_TC);
```

### **uart\_interrupt\_enable**

The description of **uart\_interrupt\_enable** is shown as below:

**Table 3-762. Function *uart\_interrupt\_enable***

<b>Function name</b>	uart_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>	
<i>interrupt</i>	USART interrupt
<i>USART_INT_PERR</i>	parity error interrupt
<i>USART_INT_TBE</i>	transmitter buffer empty interrupt
<i>USART_INT_TC</i>	transmission complete interrupt
<i>USART_INT_RBNE</i>	read data buffer not empty interrupt and overrun error interrupt
<i>USART_INT_IDLE</i>	IDLE line detected interrupt
<i>USART_INT_LBD</i>	LIN break detected interrupt
<i>USART_INT_ERR</i>	error interrupt
<i>USART_INT_CTS</i>	CTS interrupt
<i>USART_INT_RT</i>	receive timeout event interrupt
<i>USART_INT_EB</i>	end of block event interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 TBE interrupt */

uart_interrupt_enable(USART0, USART_INT_TBE);
```

### **uart\_interrupt\_disable**

The description of `uart_interrupt_disable` is shown as below:

**Table 3-763. Function `uart_interrupt_disable`**

<b>Function name</b>	<code>uart_interrupt_disable</code>
<b>Function prototype</b>	<code>void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);</code>
<b>Function descriptions</b>	disable USART interrupt
<b>Precondition</b>	-

The called functions	
Input parameter{in}	
<b>usart_periph</b>	USARTx/UARTx peripheral
<b>USARTx</b>	x=0,1,2
<b>UARTx</b>	x=3,4
Input parameter{in}	
<b>int_flag</b>	USART interrupt flag
<b>USART_INT_PERR</b>	parity error interrupt
<b>USART_INT_TBE</b>	transmitter buffer empty interrupt
<b>USART_INT_TC</b>	transmission complete interrupt
<b>USART_INT_RBNE</b>	read data buffer not empty interrupt and overrun error interrupt
<b>USART_INT_IDLE</b>	IDLE line detected interrupt
<b>USART_INT_LBD</b>	LIN break detected interrupt
<b>USART_INT_ERR</b>	error interrupt
<b>USART_INT_CTS</b>	CTS interrupt
<b>USART_INT_RT</b>	receive timeout event interrupt
<b>USART_INT_EB</b>	end of block event interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 TBE interrupt */

uart_interrupt_disable(USART0, USART_INT_TBE);
```

### **uart\_interrupt\_flag\_get**

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

**Table 3-764. Function **uart\_interrupt\_flag\_get****

Function name	uart_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 usart_interrupt_flag_enum
<i>USART_INT_FLAG_PE RR</i>	parity error interrupt and flag
<i>USART_INT_FLAG_TB E</i>	transmitter buffer empty interrupt and flag
<i>USART_INT_FLAG_TC</i>	transmission complete interrupt and flag
<i>USART_INT_FLAG_RB NE</i>	read data buffer not empty interrupt and flag
<i>USART_INT_FLAG_RB NE_ORERR</i>	read data buffer not empty interrupt and overrun error flag
<i>USART_INT_FLAG_ID LE</i>	IDLE line detected interrupt and flag
<i>USART_INT_FLAG_LB D</i>	LIN break detected interrupt and flag
<i>USART_INT_FLAG_CT S</i>	CTS interrupt and flag
<i>USART_INT_FLAG_ER R_ORERR</i>	error interrupt and overrun error
<i>USART_INT_FLAG_ER R_NERR</i>	error interrupt and noise error flag
<i>USART_INT_FLAG_ER R_FERR</i>	error interrupt and frame error flag

<b>USART_INT_FLAG_EB</b>	end of block event interrupt flag
<b>USART_INT_FLAG_RT</b>	receive timeout event interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the USART0 interrupt flag status */

FlagStatus status;

status = usart_interrupt_flag_get(USART0, USART_INT_FLAG_RBNE);
```

### **usart\_interrupt\_flag\_clear**

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

**Table 3-765. 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
<b>USARTx</b>	x=0,1,2
<b>UARTx</b>	x=3,4
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag, refer to usart_interrupt_flag_enum
<b>USART_INT_FLAG_CTS</b>	CTS change flag
<b>USART_INT_FLAG_LBD</b>	LIN break detected flag

<i>USART_INT_FLAG_TC</i>	transmission complete
<i>USART_INT_FLAG_RBNE</i>	read data buffer not empty
<i>USART_INT_FLAG_EB</i>	end of block event interrupt flag
<i>USART_INT_FLAG_RT</i>	receive timeout event interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the USART0 interrupt enable bit status */

uart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

## 3.25. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.25.1](#), the FWDGT firmware functions are introduced in chapter [3.25.2](#).

### 3.25.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

**Table 3-766. WWDGT Registers**

Registers	Descriptions
WWDGT_CTL	Control register
WWDGT_CFG	Configuration register
WWDGT_STAT	Status register

### 3.25.2. Descriptions of Peripheral functions

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

**Table 3-767. 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 3-768. Function wwdgt\_deinit**

<b>Function name</b>	wwdgt_deinit
<b>Function prototype</b>	void wwdgt_deinit(void);
<b>Function descriptions</b>	reset the window watchdog timer configuration
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset the window watchdog timer configuration */
wwdgt_deinit();
```

### **wwdgt\_enable**

The description of wwdgt\_enable is shown as below:

**Table 3-769. 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 3-770. Function wwdgt\_counter\_update**

<b>Function name</b>	wwdgt_counter_update
<b>Function prototype</b>	void wwdgt_counter_update(uint16_t counter_value);
<b>Function descriptions</b>	configure the window watchdog timer counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
counter_value	0x00 - 0x7F
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* update WWDGT counter to 0x7F */

wwdgt_counter_update(127);
```

### wwdgt\_config

The description of wwdgt\_config is shown as below:

**Table 3-771. 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
<i>WWDGT_CFG_PSC_D IV1</i>	the time base of window watchdog counter = (PCLK1/4096)/1
<i>WWDGT_CFG_PSC_D IV2</i>	the time base of window watchdog counter = (PCLK1/4096)/2
<i>WWDGT_CFG_PSC_D IV4</i>	the time base of window watchdog counter = (PCLK1/4096)/4
<i>WWDGT_CFG_PSC_D IV8</i>	the time base of window watchdog counter = (PCLK1/4096)/8
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* confiure 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 3-772. Function wwdgt\_interrupt\_enable**

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

Example:

```
/* enable early wakeup interrupt of WWDGT */

wwdgt_interrupt_enable ( );
```

### **wwdgt\_flag\_get**

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

**Table 3-773. Function wwdgt\_flag\_get**

<b>Function name</b>	wwdgt_flag_get
----------------------	----------------

<b>Function prototype</b>	FlagStatus wwdgt_flag_get(void);
<b>Function descriptions</b>	check early wakeup interrupt state of WWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```

/* test if the counter value update has reached the 0x40 */

FlagStatus status;

status = wwdgt_flag_get ( );

if(status == RESET)

{
    ...

}

else
{
    ...
}

```

### **wwdgt\_flag\_clear**

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

**Table 3-774. 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>	-

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

Example:

```
/* clear early wakeup interrupt state of WWDGT */  
wwdgt_flag_clear( );
```

### 3.26. USBFS

Firmware function description of USBFS refers to document ***GD32F30x-Firmware-Library-USB User Manual\_V1.0.***

## 4. Revision history

**Table 4-1. Revision history**

Revision No.	Description	Date
1.0	Initial Release	Dec.26, 2018

## 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.