

**GigaDevice Semiconductor Inc.**

**GD32VF103  
RISC-V 32-bit MCU**

**Firmware Library  
User Guide**

Revision 1.3

(Jan. 2024)

# 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>19</b>
<b>1.1. Rules of User Manual and Firmware Library .....</b>	<b>19</b>
1.1.1. Peripherals.....	19
1.1.2. Naming rules.....	20
<b>2. Firmware Library Overview.....</b>	<b>21</b>
<b>2.1. File Structure of Firmware Library .....</b>	<b>21</b>
2.1.1. Docs Folder.....	22
2.1.2. Examples Folder .....	22
2.1.3. Firmware Folder.....	22
2.1.4. Template Folder .....	23
2.1.5. Utilities Folder .....	26
<b>2.2. File descriptions of Firmware Library .....</b>	<b>26</b>
<b>3. Firmware Library of Standard Peripherals .....</b>	<b>28</b>
<b>3.1. Overview of Firmware Library of Standard Peripherals.....</b>	<b>28</b>
<b>3.2. ADC .....</b>	<b>28</b>
3.2.1. Descriptions of Peripheral registers.....	28
3.2.2. Descriptions of Peripheral functions .....	29
<b>3.3. BKP.....</b>	<b>56</b>
3.3.1. Descriptions of Peripheral registers.....	57
3.3.2. Descriptions of Peripheral functions .....	57
<b>3.4. CAN .....</b>	<b>68</b>
3.4.1. Descriptions of Peripheral registers.....	68
3.4.2. Descriptions of Peripheral functions .....	69
<b>3.5. CRC .....</b>	<b>87</b>
3.5.1. Descriptions of Peripheral registers.....	87
3.5.2. Descriptions of Peripheral functions .....	88
<b>3.6. DAC .....</b>	<b>92</b>
3.6.1. Descriptions of Peripheral registers.....	错误!未定义书签。
3.6.2. Descriptions of Peripheral functions .....	错误!未定义书签。
<b>3.7. DBG .....</b>	<b>107</b>
3.7.1. Descriptions of Peripheral registers.....	107

3.7.2.	Descriptions of Peripheral functions .....	107
<b>3.8.</b>	<b>DMA .....</b>	<b>111</b>
3.8.1.	Descriptions of Peripheral registers .....	111
3.8.2.	Descriptions of Peripheral functions .....	112
<b>3.9.</b>	<b>ECLIC .....</b>	<b>132</b>
3.9.1.	Descriptions of Peripheral functions .....	132
<b>3.10.</b>	<b>EXMC .....</b>	<b>137</b>
3.10.1.	Descriptions of Peripheral registers .....	138
3.10.2.	Descriptions of Peripheral functions .....	138
<b>3.11.</b>	<b>EXTI.....</b>	<b>142</b>
3.11.1.	Descriptions of Peripheral registers .....	142
3.11.2.	Descriptions of Peripheral functions .....	142
<b>3.12.</b>	<b>FMC .....</b>	<b>149</b>
3.12.1.	Descriptions of Peripheral registers .....	149
3.12.2.	Descriptions of Peripheral functions .....	150
<b>3.13.</b>	<b>FWDGT.....</b>	<b>167</b>
3.13.1.	Descriptions of Peripheral registers .....	167
3.13.2.	Descriptions of Peripheral functions .....	167
<b>3.14.</b>	<b>GPIO.....</b>	<b>171</b>
3.14.1.	Descriptions of Peripheral registers .....	171
3.14.2.	Descriptions of Peripheral functions .....	172
<b>3.15.</b>	<b>I2C .....</b>	<b>183</b>
3.15.1.	Descriptions of Peripheral registers .....	184
3.15.2.	Descriptions of Peripheral functions .....	184
<b>3.16.</b>	<b>PMU.....</b>	<b>204</b>
3.16.1.	Descriptions of Peripheral registers .....	205
3.16.2.	Descriptions of Peripheral functions .....	205
<b>3.17.</b>	<b>RCU .....</b>	<b>212</b>
3.17.1.	Descriptions of Peripheral registers .....	212
3.17.2.	Descriptions of Peripheral functions .....	212
<b>3.18.</b>	<b>RTC .....</b>	<b>240</b>
3.18.1.	Descriptions of Peripheral registers .....	240
3.18.2.	Descriptions of Peripheral functions .....	240
<b>3.19.</b>	<b>SPI.....</b>	<b>249</b>
3.19.1.	Descriptions of Peripheral registers .....	249
3.19.2.	Descriptions of Peripheral functions .....	249
<b>3.20.</b>	<b>TIMER.....</b>	<b>272</b>
3.20.1.	Descriptions of Peripheral registers .....	273
3.20.2.	Descriptions of Peripheral functions .....	273

---

<b>3.21.</b>	<b>USART.....</b>	<b>327</b>
3.21.1.	Descriptions of Peripheral registers.....	327
3.21.2.	Descriptions of Peripheral functions .....	328
<b>3.22.</b>	<b>WWDGT.....</b>	<b>356</b>
3.22.1.	Descriptions of Peripheral registers.....	356
3.22.2.	Descriptions of Peripheral functions .....	356
<b>4.</b>	<b>Revision history .....</b>	<b>361</b>

## List of Figures

Figure 2-1. File structure of firmware library of GD32VF103 .....	21
Figure 2-2. Select peripheral example files .....	23
Figure 2-3. Copy the peripheral example files .....	24
Figure 2-4. Open the project file .....	24
Figure 2-5. Configure project files .....	25
Figure 2-6. Compile .....	26

## List of Tables

Table 1-1. Peripherals .....	19
Table 2-1. Function descriptions of Firmware Library .....	26
Table 3-1. Peripheral function format of Firmware Library .....	28
Table 3-2. ADC Registers .....	28
Table 3-3. ADC firmware function.....	29
Table 3-4. Function adc_deinit.....	30
Table 3-5. Function adc_mode_config .....	31
Table 3-6. Function adc_special_function_config.....	32
Table 3-7. Function adc_data_alignment_config.....	33
Table 3-8. Function adc_enable .....	33
Table 3-9. Function adc_disable .....	34
Table 3-10. Function adc_calibration_enable.....	34
Table 3-11. Function adc_tempsensor_vrefint_enable.....	35
Table 3-12. Function adc_tempsensor_vrefint_disable.....	35
Table 3-13. Function adc_dma_mode_enable.....	36
Table 3-14. Function adc_dma_mode_disable.....	36
Table 3-15. Function adc_discontinuous_mode_config .....	37
Table 3-16. Function adc_channel_length_config.....	37
Table 3-17. Function adc_regular_channel_config .....	38
Table 3-18. Function adc_inserted_channel_config .....	39
Table 3-19. Function adc_inserted_channel_offset_config .....	40
Table 3-20. Function adc_external_trigger_source_config .....	41
Table 3-21. Function adc_external_trigger_config.....	43
Table 3-22. Function adc_software_trigger_enable .....	43
Table 3-23. Function adc_regular_data_read .....	44
Table 3-24. Function adc_inserted_data_read .....	45
Table 3-25. Function adc_sync_mode_convert_value_read .....	45
Table 3-26. Function adc_watchdog_single_channel_enable.....	46
Table 3-27. Function adc_watchdog_group_channel_enable.....	46
Table 3-28. Function adc_watchdog_disable .....	47
Table 3-29. Function adc_watchdog_threshold_config .....	48
Table 3-30. Function adc_flag_get .....	48
Table 3-31. Function adc_flag_clear .....	49
Table 3-32. Function adc_regular_software_startconv_flag_get.....	50
Table 3-33. Function adc_inserted_software_startconv_flag_get.....	50
Table 3-34. Function adc_interrupt_flag_get.....	51
Table 3-35. Function adc_interrupt_flag_clear .....	51
Table 3-36. Function adc_interrupt_enable .....	52
Table 3-37. Function adc_interrupt_disable .....	53
Table 3-38. Function adc_resolution_config .....	53



Table 3-39. Function <code>adc_oversample_mode_config</code> .....	54
Table 3-40. Function <code>adc_oversample_mode_enable</code> .....	55
Table 3-41. Function <code>adc_oversample_mode_disable</code> .....	56
Table 3-42. BKP Registers .....	57
Table 3-43. BKP firmware function.....	57
Table 3-44. Enum <code>bkp_data_register_enum</code> .....	58
Table 3-45. Function <code>bkp_deinit</code> .....	59
Table 3-46. Function <code>bkp_data_write</code> .....	59
Table 3-47. Function <code>bkp_data_read</code> .....	60
Table 3-48. Function <code>bkp_rtc_calibration_output_enable</code> .....	60
Table 3-49. Function <code>bkp_rtc_calibration_output_disable</code> .....	61
Table 3-50. Function <code>bkp_rtc_signal_output_enable</code> .....	61
Table 3-51. Function <code>bkp_rtc_signal_output_disable</code> .....	62
Table 3-52. Function <code>bkp_rtc_output_select</code> .....	62
Table 3-53. Function <code>bkp_rtc_calibration_value_set</code> .....	63
Table 3-54. Function <code>bkp_tamper_detection_enable</code> .....	63
Table 3-55. Function <code>bkp_tamper_detection_disable</code> .....	64
Table 3-56. Function <code>bkp_tamper_active_level_set</code> .....	64
Table 3-57. Function <code>bkp_interrupt_enable</code> .....	65
Table 3-58. Function <code>bkp_interrupt_disable</code> .....	65
Table 3-59. Function <code>bkp_flag_get</code> .....	66
Table 3-60. Function <code>bkp_flag_clear</code> .....	66
Table 3-61. Function <code>bkp_interrupt_flag_get</code> .....	67
Table 3-62. Function <code>bkp_interrupt_flag_clear</code> .....	67
Table 3-63. CAN Registers .....	68
Table 3-64. CAN firmware function .....	69
Table 3-65. Structure <code>can_parameter_struct</code> .....	70
Table 3-66. Structure <code>can_transmit_message_struct</code> .....	70
Table 3-67. Structure <code>can_receive_message_struct</code> .....	70
Table 3-68. Structure <code>can_filter_parameter_struct</code> .....	71
Table 3-69. Function <code>can_deinit</code> .....	71
Table 3-70. Function <code>can_struct_para_init</code> .....	71
Table 3-71. Function <code>can_init</code> .....	72
Table 3-72. Function <code>can_filter_init</code> .....	73
Table 3-73. Function <code>can1_filter_start_bank</code> .....	73
Table 3-74. Function <code>can_debug_freeze_enable</code> .....	74
Table 3-75. Function <code>can_debug_freeze_disable</code> .....	74
Table 3-76. Function <code>can_time_trigger_mode_enable</code> .....	75
Table 3-77. Function <code>can_time_trigger_mode_disable</code> .....	75
Table 3-78. Function <code>can_message_transmit</code> .....	76
Table 3-79. Function <code>can_transmit_states</code> .....	76
Table 3-80. Function <code>can_transmission_stop</code> .....	77
Table 3-81. Function <code>can_message_receive</code> .....	78
Table 3-82. Function <code>can_fifo_release</code> .....	78

Table 3-83. Function can_receive_message_length_get .....	79
Table 3-84. Function can_working_mode_set.....	79
Table 3-85. Function can_wakeup .....	80
Table 3-86. Function can_error_get .....	81
Table 3-87. Function can_receive_error_number_get .....	81
Table 3-88. Function can_transmit_error_number_get .....	82
Table 3-89. Function can_interrupt_enable .....	82
Table 3-90. Function can_interrupt_disable .....	83
Table 3-91. Function can_flag_get .....	84
Table 3-92. Function can_flag_clear .....	85
Table 3-93. Function can_interrupt_flag_get.....	86
Table 3-94. Function can_interrupt_flag_clear .....	86
Table 3-95. CRC Registers .....	88
Table 3-96. CRC firmware function .....	88
Table 3-97. Function crc_deinit.....	88
Table 3-98. Function crc_data_register_reset.....	89
Table 3-99. Function crc_data_register_read.....	89
Table 3-100. Function crc_free_data_register_read.....	90
Table 3-101. Function crc_free_data_register_write.....	90
Table 3-102. Function crc_single_data_calculate .....	91
Table 3-103. Function crc_block_data_calculate .....	91
Table 3-104. DAC Registers .....	错误!未定义书签。
Table 3-105. DAC firmware function .....	错误!未定义书签。
Table 3-106. Function dac_deinit.....	错误!未定义书签。
Table 3-107. Function dac_enable .....	错误!未定义书签。
Table 3-108. Function dac_disable.....	错误!未定义书签。
Table 3-109. Function dac_dma_enable.....	错误!未定义书签。
Table 3-110. Function dac_dma_disable .....	错误!未定义书签。
Table 3-111. Function dac_output_buffer_enable .....	错误!未定义书签。
Table 3-112. Function dac_output_buffer_disable .....	错误!未定义书签。
Table 3-113. Function dac_output_value_get.....	错误!未定义书签。
Table 3-114. Function dac_data_set.....	错误!未定义书签。
Table 3-115. Function dac_trigger_enable .....	错误!未定义书签。
Table 3-116. Function dac_trigger_disable .....	错误!未定义书签。
Table 3-117. Function dac_trigger_source_config.....	错误!未定义书签。
Table 3-118. Function dac_software_trigger_enable .....	错误!未定义书签。
Table 3-119. Function dac_software_trigger_disable .....	错误!未定义书签。
Table 3-120. Function dac_wave_mode_config.....	错误!未定义书签。
Table 3-121. Function dac_wave_bit_width_config .....	错误!未定义书签。
Table 3-122. Function dac_lfsr_noise_config .....	错误!未定义书签。
Table 3-123. Function dac_triangle_noise_config.....	错误!未定义书签。
Table 3-124. Function dac_concurrent_enable .....	错误!未定义书签。
Table 3-125. Function dac_concurrent_disable .....	错误!未定义书签。
Table 3-126. Function dac_concurrent_software_trigger_enable .....	错误!未定义书签。



Table 3-127. Function <code>dac_concurrent_software_trigger_disable</code> .....	错误!未定义书签。
Table 3-128. Function <code>dac_concurrent_output_buffer_enable</code> .....	错误!未定义书签。
Table 3-129. Function <code>dac_concurrent_output_buffer_disable</code> .....	错误!未定义书签。
Table 3-130. DBG Registers .....	107
Table 3-131. DBG firmware function .....	107
Table 3-132. Enum <code>dbg_periph_enum</code> .....	107
Table 3-133. Function <code>dbg_deinit</code> .....	108
Table 3-134. Function <code>dbg_id_get</code> .....	108
Table 3-135. Function <code>dbg_low_power_enable</code> .....	109
Table 3-136. Function <code>dbg_low_power_disable</code> .....	109
Table 3-137. Function <code>dbg_periph_enable</code> .....	110
Table 3-138. Function <code>dbg_periph_disable</code> .....	111
Table 3-139. DMA Registers .....	111
Table 3-140. DMA firmware function .....	112
Table 3-141. Structure <code>dma_parameter_struct</code> .....	113
Table 3-142. Function <code>dma_deinit</code> .....	113
Table 3-143. Function <code>dma_para_init</code> .....	114
Table 3-144. Function <code>dma_init</code> .....	114
Table 3-145. Function <code>dma_circulation_enable</code> .....	115
Table 3-146. Function <code>dma_circulation_disable</code> .....	116
Table 3-147. Function <code>dma_memory_to_memory_enable</code> .....	116
Table 3-148. Function <code>dma_memory_to_memory_disable</code> .....	117
Table 3-149. Function <code>dma_channel_enable</code> .....	117
Table 3-150. Function <code>dma_channel_disable</code> .....	118
Table 3-151. Function <code>dma_periph_address_config</code> .....	119
Table 3-152. Function <code>dma_memory_address_config</code> .....	119
Table 3-153. Function <code>dma_transfer_number_config</code> .....	120
Table 3-154. Function <code>dma_transfer_number_get</code> .....	121
Table 3-155. Function <code>dma_priority_config</code> .....	121
Table 3-156. Function <code>dma_memory_width_config</code> .....	122
Table 3-157. Function <code>dma_periph_width_config</code> .....	123
Table 3-158. Function <code>dma_memory_increase_enable</code> .....	124
Table 3-159. Function <code>dma_memory_increase_disable</code> .....	125
Table 3-160. Function <code>dma_periph_increase_enable</code> .....	125
Table 3-161. Function <code>dma_periph_increase_disable</code> .....	126
Table 3-162. Function <code>dma_transfer_direction_config</code> .....	126
Table 3-163. Function <code>dma_flag_get</code> .....	127
Table 3-164. Function <code>dma_flag_clear</code> .....	128
Table 3-165. Function <code>dma_interrupt_flag_get</code> .....	129
Table 3-166. Function <code>dma_interrupt_flag_clear</code> .....	130
Table 3-167. Function <code>dma_interrupt_enable</code> .....	130
Table 3-168. Function <code>dma_interrupt_disable</code> .....	131
Table 3-169. Enum <code>IRQn_Type</code> .....	132
Table 3-170. ECLIC firmware function .....	134

Table 3-171. Function <code>eclic_global_interrupt_enable</code> .....	134
Table 3-172. Function <code>eclic_global_interrupt_disable</code> .....	134
Table 3-173. Function <code>eclic_priority_group_set</code> .....	135
Table 3-174. Function <code>eclic_irq_enable</code> .....	135
Table 3-175. Function <code>eclic_irq_disable</code> .....	136
Table 3-176. Function <code>eclic_system_reset</code> .....	137
Table 3-177. Function <code>eclic_send_event</code> .....	137
Table 3-178. EXMC Registers .....	138
Table 3-179. EXMC firmware function .....	138
Table 3-180. Structure <code>exmc_norsram_timing_parameter_struct</code> .....	138
Table 3-181. Structure <code>exmc_norsram_parameter_struct</code> .....	138
Table 3-182. Function <code>exmc_norsram_deinit</code> .....	139
Table 3-183. Function <code>exmc_norsram_struct_para_init</code> .....	139
Table 3-184. Function <code>exmc_norsram_init</code> .....	140
Table 3-185. Function <code>exmc_norsram_enable</code> .....	141
Table 3-186. Function <code>exmc_norsram_disable</code> .....	141
Table 3-187. EXTI Registers .....	142
Table 3-188. EXTI firmware function .....	142
Table 3-189. Enum <code>exti_line_enum</code> .....	143
Table 3-190. Enum <code>exti_mode_enum</code> .....	143
Table 3-191. Enum <code>exti_trig_type_enum</code> .....	143
Table 3-192. Function <code>exti_deinit</code> .....	144
Table 3-193. Function <code>exti_init</code> .....	144
Table 3-194. Function <code>exti_interrupt_enable</code> .....	145
Table 3-195. Function <code>exti_interrupt_disable</code> .....	145
Table 3-196. Function <code>exti_event_enable</code> .....	146
Table 3-197. Function <code>exti_event_disable</code> .....	146
Table 3-198. Function <code>exti_software_interrupt_enable</code> .....	147
Table 3-199. Function <code>exti_software_interrupt_disable</code> .....	147
Table 3-200. Function <code>exti_flag_get</code> .....	147
Table 3-201. Function <code>exti_flag_clear</code> .....	148
Table 3-202. Function <code>exti_interrupt_flag_get</code> .....	148
Table 3-203. Function <code>exti_interrupt_flag_clear</code> .....	149
Table 3-204. FMC Registers .....	150
Table 3-205. FMC firmware function .....	150
Table 3-206. Enum <code>fmc_state_enum</code> .....	151
Table 3-207. Enum <code>fmc_int_enum</code> .....	151
Table 3-208. Enum <code>fmc_flag_enum</code> .....	151
Table 3-209. Enum <code>fmc_interrupt_flag_enum</code> .....	151
Table 3-210. Function <code>fmc_wsctl_set</code> .....	152
Table 3-211. Function <code>fmc_unlock</code> .....	152
Table 3-212. Function <code>fmc_lock</code> .....	153
Table 3-213. Function <code>fmc_page_erase</code> .....	153
Table 3-214. Function <code>fmc_mass_erase</code> .....	154

Table 3-215. Function fmc_word_program .....	154
Table 3-216. Function fmc_word_program .....	155
Table 3-217. Function ob_unlock .....	156
Table 3-218. Function ob_lock .....	156
Table 3-219. Function ob_erase .....	157
Table 3-220. Function ob_write_protection_enable .....	158
Table 3-221. Function ob_security_protection_config .....	158
Table 3-222. Function ob_user_write .....	159
Table 3-223. Function ob_data_program .....	160
Table 3-224. Function ob_user_get .....	161
Table 3-225. Function ob_data_get .....	161
Table 3-226. Function ob_write_protection_get .....	162
Table 3-227. Function ob_spc_get .....	162
Table 3-228. Function fmc_interrupt_enable .....	163
Table 3-229. Function fmc_interrupt_disable .....	163
Table 3-230. Function fmc_flag_get .....	164
Table 3-231. Function fmc_flag_clear .....	164
Table 3-232. Function fmc_interrupt_flag_get .....	165
Table 3-233. Function fmc_interrupt_flag_clear .....	166
Table 3-234. Function fmc_state_get .....	166
Table 3-235. Function fmc_ready_wait .....	167
Table 3-236. FWDGT Registers .....	167
Table 3-237. FWDGT firmware function .....	168
Table 3-238. Function fwdgt_write_ensable .....	168
Table 3-239. Function fwdgt_write_disable .....	168
Table 3-240. Function fwdgt_enable .....	169
Table 3-241. Function fwdgt_counter_reload .....	169
Table 3-242. Function fwdgt_config .....	170
Table 3-243. Function fwdgt_flag_get .....	170
Table 3-244. GPIO Registers .....	171
Table 3-245. GPIO firmware function .....	172
Table 3-246. Function gpio_deinit .....	172
Table 3-247. Function gpio_afio_deinit .....	173
Table 3-248. Function gpio_init .....	173
Table 3-249. Function gpio_bit_set .....	174
Table 3-250. Function gpio_bit_reset .....	175
Table 3-251. Function gpio_bit_write .....	175
Table 3-252. Function gpio_port_write .....	176
Table 3-253. Function gpio_input_bit_get .....	177
Table 3-254. Function gpio_input_port_get .....	177
Table 3-255. Function gpio_output_bit_get .....	178
Table 3-256. Function gpio_output_port_get .....	179
Table 3-257. Function gpio_pin_remap_config .....	179
Table 3-258. Function gpio_exti_source_select .....	181



Table 3-259. Function gpio_event_output_config .....	181
Table 3-260. Function gpio_event_output_enable .....	182
Table 3-261. Function gpio_event_output_disable .....	182
Table 3-262. Function gpio_pin_lock .....	183
Table 3-263. I2C Registers .....	184
Table 3-264. I2C firmware function.....	184
Table 3-265. Function i2c_deinit .....	185
Table 3-266. Function i2c_clock_config .....	185
Table 3-267. Function i2c_mode_addr_config .....	186
Table 3-268. Function i2c_smbus_type_config .....	187
Table 3-269. Function i2c_ack_config .....	188
Table 3-270. Function i2c_ackpos_config .....	188
Table 3-271. Function i2c_master_addressing .....	189
Table 3-272. Function i2c_dualaddr_enable .....	189
Table 3-273. Function i2c_dualaddr_enable .....	190
Table 3-274. Function i2c_enable .....	190
Table 3-275. Function i2c_disable .....	191
Table 3-276. Function i2c_start_on_bus .....	191
Table 3-277. Function i2c_stop_on_bus .....	192
Table 3-278. Function i2c_data_transmit .....	192
Table 3-279. Function i2c_data_receive .....	193
Table 3-280. Function i2c_dma_config .....	193
Table 3-281. Function i2c_dma_last_transfer_config .....	194
Table 3-282. Function i2c_stretch_scl_low_config .....	195
Table 3-283. Function i2c_slave_response_to_gcall_config .....	195
Table 3-284. Function i2c_software_reset_config .....	196
Table 3-285. Function i2c_pec_config .....	196
Table 3-286. Function i2c_pec_transfer_config .....	197
Table 3-287. Function i2c_pec_value_get .....	198
Table 3-288. Function i2c_smbus_alert_config .....	198
Table 3-289. Function i2c_smbus_arb_config .....	199
Table 3-290. Function i2c_flag_get .....	199
Table 3-291. Function i2c_flag_clear .....	200
Table 3-292. Function i2c_interrupt_enable .....	201
Table 3-293. Function i2c_interrupt_disable .....	202
Table 3-294. Function i2c_interrupt_flag_get .....	202
Table 3-295. Function i2c_interrupt_flag_clear .....	203
Table 3-296. PMU Registers .....	205
Table 3-297. PMU firmware function .....	205
Table 3-298. Function pmu_deinit .....	205
Table 3-299. Function pmu_lvd_select .....	206
Table 3-300. Function pmu_lvd_disable .....	206
Table 3-301. Function pmu_to_sleepmode .....	207
Table 3-302. Function pmu_to_deepsleepmode .....	207

Table 3-303. Function pmu_to_standbymode .....	208
Table 3-304. Function pmu_wakeup_pin_enable .....	209
Table 3-305. Function pmu_wakeup_pin_disable .....	209
Table 3-306. Function pmu_backup_write_enable .....	210
Table 3-307. Function pmu_backup_write_disable .....	210
Table 3-308. Function pmu_flag_clear .....	211
Table 3-309. Function pmu_flag_get .....	211
Table 3-310. RCU Registers .....	212
Table 3-311. RCU firmware function .....	212
Table 3-312. Function rcu_deinit .....	213
Table 3-313. Function rcu_periph_clock_enable .....	214
Table 3-314. Function rcu_periph_clock_disable .....	215
Table 3-315. Function rcu_periph_clock_sleep_enable .....	216
Table 3-316. Function rcu_periph_clock_sleep_disable .....	216
Table 3-317. Function rcu_periph_reset_enable .....	217
Table 3-318. Function rcu_periph_reset_disable .....	218
Table 3-319. Function rcu_bkp_reset_enable .....	218
Table 3-320. Function rcu_bkp_reset_disable .....	219
Table 3-321. Function rcu_system_clock_source_config .....	219
Table 3-322. Function rcu_system_clock_source_get .....	220
Table 3-323. Function rcu_ahb_clock_config .....	221
Table 3-324. Function rcu_apb1_clock_config .....	221
Table 3-325. Function rcu_apb2_clock_config .....	222
Table 3-326. Function rcu_ckout0_config .....	222
Table 3-327. Function rcu_pll_config .....	223
Table 3-328. Function rcu_predv0_config .....	224
Table 3-329. Function rcu_predv1_config .....	225
Table 3-330. Function rcu_pll1_config .....	225
Table 3-331. Function rcu_pll2_config .....	226
Table 3-332. Function rcu_adc_clock_config .....	226
Table 3-333. Function rcu_usb_clock_config .....	227
Table 3-334. Function rcu_rtc_clock_config .....	228
Table 3-335. Function rcu_i2s1_clock_config .....	228
Table 3-336. Function rcu_i2s2_clock_config .....	229
Table 3-337. Function rcu_flag_get .....	229
Table 3-338. Function rcu_all_reset_flag_clear .....	230
Table 3-339. Function rcu_interrupt_flag_get .....	231
Table 3-340. Function rcu_interrupt_flag_clear .....	232
Table 3-341. Function rcu_interrupt_enable .....	233
Table 3-342. Function rcu_interrupt_disable .....	233
Table 3-343. Function rcu_osc_i_stab_wait .....	234
Table 3-344. Function rcu_osc_i_on .....	235
Table 3-345. Function rcu_osc_i_off .....	235
Table 3-346. Function rcu_osc_i_bypass_mode_enable .....	236

Table 3-347. Function rcu_osc_bypass_mode_disable .....	236
Table 3-348. Function rcu_hxtal_clock_monitor_enable .....	237
Table 3-349. Function rcu_hxtal_clock_monitor_disable .....	237
Table 3-350. Function rcu_irc8m_adjust_value_set.....	238
Table 3-351. Function rcu_deepsleep_voltage_set .....	238
Table 3-352. Function rcu_clock_freq_get.....	239
Table 3-353. RTC Registers .....	240
Table 3-354. RTC firmware function.....	240
Table 3-355. Function rtc_configuration_mode_enter .....	241
Table 3-356. Function rtc_configuration_mode_exit .....	241
Table 3-357. Function rtc_counter_set .....	242
Table 3-358. Function rtc_prescaler_set .....	242
Table 3-359. Function rtc_lwoff_wait .....	243
Table 3-360. Function rtc_register_sync_wait .....	243
Table 3-361. Function rtc_alarm_config.....	244
Table 3-362. Function rtc_counter_get.....	244
Table 3-363. Function rtc_divider_get .....	245
Table 3-364. Function rtc_flag_get.....	245
Table 3-365. Function rtc_flag_clear.....	246
Table 3-366. Function rtc_interrupt_flag_get .....	247
Table 3-367. Function rtc_interrupt_flag_clear .....	247
Table 3-368. Function rtc_interrupt_enable .....	248
Table 3-369. Function rtc_interrupt_disable .....	248
Table 3-370. SPI/I2S registers.....	249
Table 3-371. SPI/I2S firmware function.....	250
Table 3-372. Structure spi_parameter_struct.....	250
Table 3-373. Function spi_i2s_deinit .....	251
Table 3-374. Function spi_i2s_deinit .....	252
Table 3-375. Function spi_init .....	252
Table 3-376. Function spi_enable .....	253
Table 3-377. Function spi_disable .....	253
Table 3-378. Function i2s_init .....	254
Table 3-379. Function i2s_psc_config .....	255
Table 3-380. Function i2s_enable .....	256
Table 3-381. Function i2s_disable .....	257
Table 3-382. Function spi_nss_output_enable .....	257
Table 3-383. Function spi_nss_output_disable .....	258
Table 3-384. Function spi_nss_internal_high .....	258
Table 3-385. Function spi_nss_internal_low .....	259
Table 3-386. Function spi_dma_enable .....	259
Table 3-387. Function spi_dma_disable .....	260
Table 3-388. Function spi_i2s_data_frame_format_config .....	261
Table 3-389. Function spi_bidirectional_transfer_config.....	261
Table 3-390. Function spi_i2s_data_transmit.....	262

Table 3-391. Function spi_i2s_data_receive .....	262
Table 3-392. Function spi_i2s_format_error_clear.....	263
Table 3-393. Function spi_crc_polynomial_set .....	264
Table 3-394. Function spi_crc_polynomial_get .....	264
Table 3-395. Function spi_crc_on .....	265
Table 3-396. Function spi_crc_off .....	265
Table 3-397. Function spi_crc_next .....	266
Table 3-398. Function spi_crc_get .....	266
Table 3-399. Function spi_crc_error_clear .....	267
Table 3-400. Function spi_ti_mode_enable .....	267
Table 3-401. Function spi_ti_mode_disable .....	268
Table 3-402. Function spi_nssp_mode_enable.....	268
Table 3-403. Function spi_nssp_mode_disable.....	269
Table 3-404. Function spi_i2s_flag_get .....	269
Table 3-405. Function spi_i2s_interrupt_enable .....	270
Table 3-406. Function spi_i2s_interrupt_disable .....	271
Table 3-407. Function spi_i2s_interrupt_flag_get .....	272
Table 3-408. TIMERx Registers .....	273
Table 3-409. TIMERx firmware function.....	273
Table 3-410. Structure timer_parameter_struct .....	276
Table 3-411. Structure timer_break_parameter_struct .....	276
Table 3-412. Structure timer_oc_parameter_struct.....	276
Table 3-413. Structure timer_ic_parameter_struct.....	277
Table 3-414. Function timer_deinit.....	277
Table 3-415. Function timer_struct_para_init.....	278
Table 3-416. Function timer_init .....	278
Table 3-417. Function timer_enable .....	279
Table 3-418. Function timer_disable .....	279
Table 3-419. Function timer_auto_reload_shadow_enable .....	280
Table 3-420. Function timer_auto_reload_shadow_disable .....	280
Table 3-421. Function timer_update_event_enable .....	281
Table 3-422. Function timer_update_event_disable .....	281
Table 3-423. Function timer_counter_alignment .....	282
Table 3-424. Function timer_counter_up_direction .....	283
Table 3-425. Function timer_counter_down_direction .....	283
Table 3-426. Function timer_prescaler_config.....	284
Table 3-427. Function timer_repetition_value_config .....	285
Table 3-428. Function timer_autoreload_value_config .....	285
Table 3-429. Function timer_counter_value_config.....	286
Table 3-430. Function timer_counter_read .....	286
Table 3-431. Function timer_prescaler_read .....	287
Table 3-432. Function timer_single_pulse_mode_config .....	287
Table 3-433. Function timer_update_source_config.....	288
Table 3-434. Function timer_dma_enable .....	289



Table 3-435. Function timer_dma_disable .....	289
Table 3-436. Function timer_channel_dma_request_source_select.....	290
Table 3-437. Function timer_dma_transfer_config.....	291
Table 3-438. Function timer_event_software_generate.....	292
Table 3-439. Function timer_break_struct_para_init .....	293
Table 3-440. Function timer_break_config .....	294
Table 3-441. Function timer_break_enable .....	295
Table 3-442. Function timer_break_disable.....	295
Table 3-443. Function timer_automatic_output_enable .....	296
Table 3-444. Function timer_automatic_output_disable .....	296
Table 3-445. Function timer_primary_output_config.....	297
Table 3-446. Function timer_channel_control_shadow_config .....	298
Table 3-447. Function timer_channel_control_shadow_update_config.....	298
Table 3-448. Function timer_channel_output_struct_para_init .....	299
Table 3-449. Function timer_channel_output_config .....	299
Table 3-450. Function timer_channel_output_mode_config .....	300
Table 3-451. Function timer_channel_output_pulse_value_config.....	302
Table 3-452. Function timer_channel_output_shadow_config .....	302
Table 3-453. Function timer_channel_output_fast_config.....	303
Table 3-454. Function timer_channel_output_clear_config .....	304
Table 3-455. Function timer_channel_output_polarity_config.....	305
Table 3-456. Function timer_channel_complementary_output_polarity_config .....	306
Table 3-457. Function timer_channel_output_state_config .....	306
Table 3-458. Function timer_channel_complementary_output_state_config .....	307
Table 3-459. Function timer_channel_input_struct_para_init.....	308
Table 3-460. Function timer_input_capture_config.....	308
Table 3-461. Function timer_channel_input_capture_prescaler_config .....	309
Table 3-462. Function timer_channel_capture_value_register_read .....	310
Table 3-463. Function timer_input_pwm_capture_config.....	311
Table 3-464. Function timer_hall_mode_config.....	312
Table 3-465. Function timer_input_trigger_source_select .....	312
Table 3-466. Function timer_master_output_trigger_source_select .....	313
Table 3-467. Function timer_slave_mode_select .....	314
Table 3-468. Function timer_master_slave_mode_config .....	315
Table 3-469. Function timer_external_trigger_config.....	316
Table 3-470. Function timer_quadrature_decoder_mode_config.....	317
Table 3-471. Function timer_internal_clock_config .....	318
Table 3-472. Function timer_internal_trigger_as_external_clock_config .....	319
Table 3-473. Function timer_external_trigger_as_external_clock_config .....	319
Table 3-474. Function timer_external_clock_mode0_config .....	320
Table 3-475. Function timer_external_clock_mode1_config .....	321
Table 3-476. Function timer_external_clock_mode1_disable .....	322
Table 3-477. Function timer_interrupt_enable .....	323
Table 3-478. Function timer_interrupt_disable .....	323





Table 3-479. Function timer_interrupt_flag_get .....	324
Table 3-480. Function timer_interrupt_flag_clear .....	325
Table 3-481. Function timer_flag_get .....	326
Table 3-482. Function timer_flag_clear .....	327
Table 3-483. USART Registers .....	328
Table 3-484. USART firmware function .....	328
Table 3-485. Function usart_deinit .....	329
Table 3-486. Function usart_baudrate_set .....	330
Table 3-487. Function usart_parity_config .....	330
Table 3-488. Function usart_word_length_set .....	331
Table 3-489. Function usart_stop_bit_set .....	331
Table 3-490. Function usart_enable .....	332
Table 3-491. Function usart_disable .....	333
Table 3-492. Function usart_transmit_config .....	333
Table 3-493. Function usart_receive_config .....	334
Table 3-494. Function usart_data_transmit .....	335
Table 3-495. Function usart_data_receive .....	335
Table 3-496. Function usart_address_config .....	336
Table 3-497. Function usart_mute_mode_enable .....	336
Table 3-498. Function usart_mute_mode_disable .....	337
Table 3-499. Function usart_mute_mode_wakeup_config .....	337
Table 3-500. Function usart_lin_mode_enable .....	338
Table 3-501. Function usart_lin_mode_disable .....	339
Table 3-502. Function usart_lin_break_dection_length_config .....	339
Table 3-503. Function usart_send_break .....	340
Table 3-504. Function usart_halfduplex_enable .....	340
Table 3-505. Function usart_halfduplex_disable .....	341
Table 3-506. Function usart_synchronous_clock_enable .....	341
Table 3-507. Function usart_synchronous_clock_disable .....	342
Table 3-508. Function usart_synchronous_clock_config .....	342
Table 3-509. Function usart_guard_time_config .....	343
Table 3-510. Function usart_smartcard_mode_enable .....	344
Table 3-511. Function usart_smartcard_mode_disable .....	344
Table 3-512. Function usart_smartcard_mode_nack_enable .....	345
Table 3-513. Function usart_smartcard_mode_nack_disable .....	345
Table 3-514. Function usart_irda_mode_enable .....	346
Table 3-515. Function usart_irda_mode_disable .....	346
Table 3-516. Function usart_prescaler_config .....	347
Table 3-517. Function usart_irda_lowpower_config .....	347
Table 3-518. Function usart_hardware_flow_rts_config .....	348
Table 3-519. Function usart_hardware_flow_cts_config .....	349
Table 3-520. Function usart_dma_receive_config .....	349
Table 3-521. Function usart_dma_transmit_config .....	350
Table 3-522. Function usart_flag_get .....	351



Table 3-523. Function <code>usart_flag_clear</code> .....	352
Table 3-524. Function <code>usart_interrupt_enable</code> .....	352
Table 3-525. Function <code>usart_interrupt_disable</code> .....	353
Table 3-526. Function <code>usart_interrupt_flag_get</code> .....	354
Table 3-527. Function <code>usart_interrupt_flag_clear</code> .....	355
Table 3-528. WWDGT Registers .....	356
Table 3-529. WWDGT firmware function .....	356
Table 3-530. Function <code>wwdgt_deinit</code> .....	357
Table 3-531. Function <code>wwdgt_enable</code> .....	357
Table 3-532. Function <code>wwdgt_counter_update</code> .....	358
Table 3-533. Function <code>wwdgt_config</code> .....	358
Table 3-534. Function <code>wwdgt_interrupt_enable</code> .....	359
Table 3-535. Function <code>wwdgt_flag_get</code> .....	359
Table 3-536. Function <code>wwdgt_flag_clear</code> .....	360
Table 4-1. Revision history .....	361

## 1. Introduction

This manual introduces firmware library of GD32VF103 devices which are 32-bit microcontrollers based on the RISC-V processor.

The firmware library is a firmware function package, including program, data structure and macro definitions, all the performance features of peripherals of GD32VF103 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.

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
DAC	Digital-to-analog converter
DBG	Debug module
DMA	Direct memory access controller
EXMC	External memory controller
EXTI	Interrupt/event controller

Peripherals	Descriptions
FMC	Flash memory controller
FWDGT	Free watchdog timer
GPIO/AFIO	General-purpose and alternate-function I/Os
I2C	Inter-integrated circuit interface
PMU	Power management unit
RCU	Reset and clock unit
RTC	Real-time Clock
SPI/I2S	Serial peripheral interface/Inter-IC sound
TIMER	TIMER
USART	Universal synchronous/asynchronous receiver /transmitter
WWDGT	Window watchdog timer

### 1.1.2. Naming rules

The firmware library naming rules are shown as below:

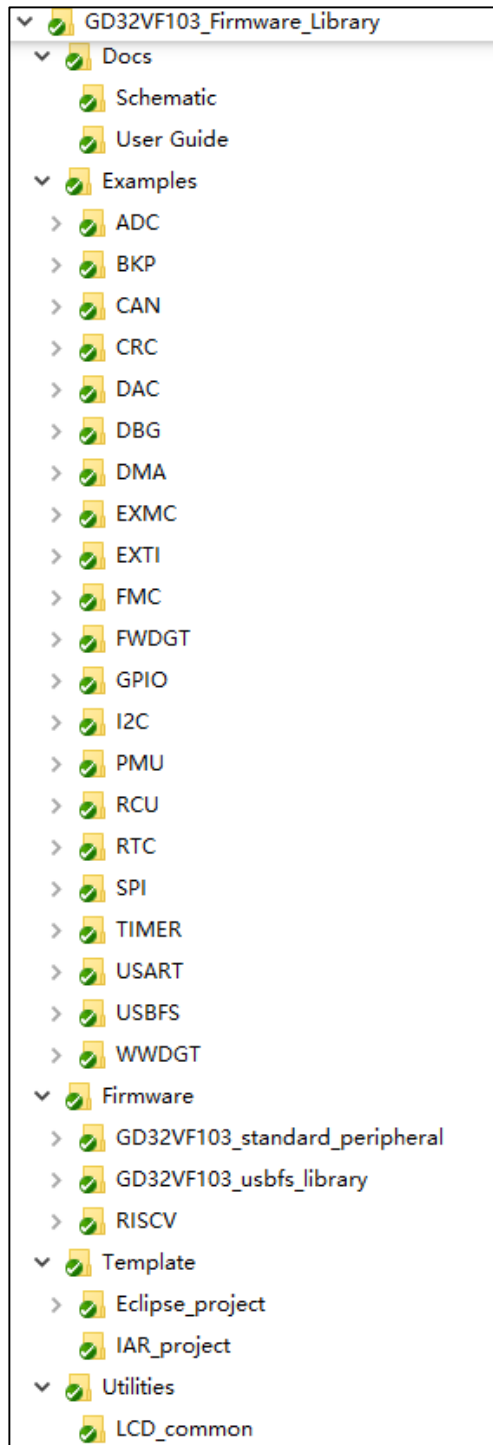
- The peripherals are shortened in XXX format, such as: ADC. More shorten information of peripherals refer to [Peripherals](#);
- The name of sourcefile and header file are started with “gd32vf103\_”, such as: gd32vf103\_adc.h;
- The constants used only in one file should be defined in the used file; 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

GD32VF103\_Firmware\_Library, the file structure is shown as below:

**Figure 2-1. File structure of firmware library of GD32VF103**



### 2.1.1. Docs Folder

The Docs folder contains the Firmware Library User Guide and development board schematic.

### 2.1.2. 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;
- `gd32vf103_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);
- `gd32vf103_it.c`: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- `gd32vf103_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.

### 2.1.3. Firmware Folder

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

- RISC-V subfolder:
  - `drivers` subfolder includes the RISC-V kernel support files, users need not modify this folder;
  - `env_eclipse` subfolder includes the startup file based on the RISC-V kernel processor, exception service program and link script files of Eclipse IDE, users need not modify this folder;
  - `env_IAR` subfolder includes the startup file based on the RISC-V kernel processor and exception service program of IAR, users need not modify this folder;
  - `stubs` subfolder includes definition of pile functions such `_write/_read` function, users need not modify this folder.
- GD32VF103\_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;
  - the global header file of GD32VF103 and system configuration file, users need not modify this folder.
- GD32VF103\_usbfs\_driver subfolder:

- Include subfolder includes all the header files of USB firmware library, users need not modify this folder;
- Source subfolder includes all the source files of USB firmware library, users need not modify this folder.

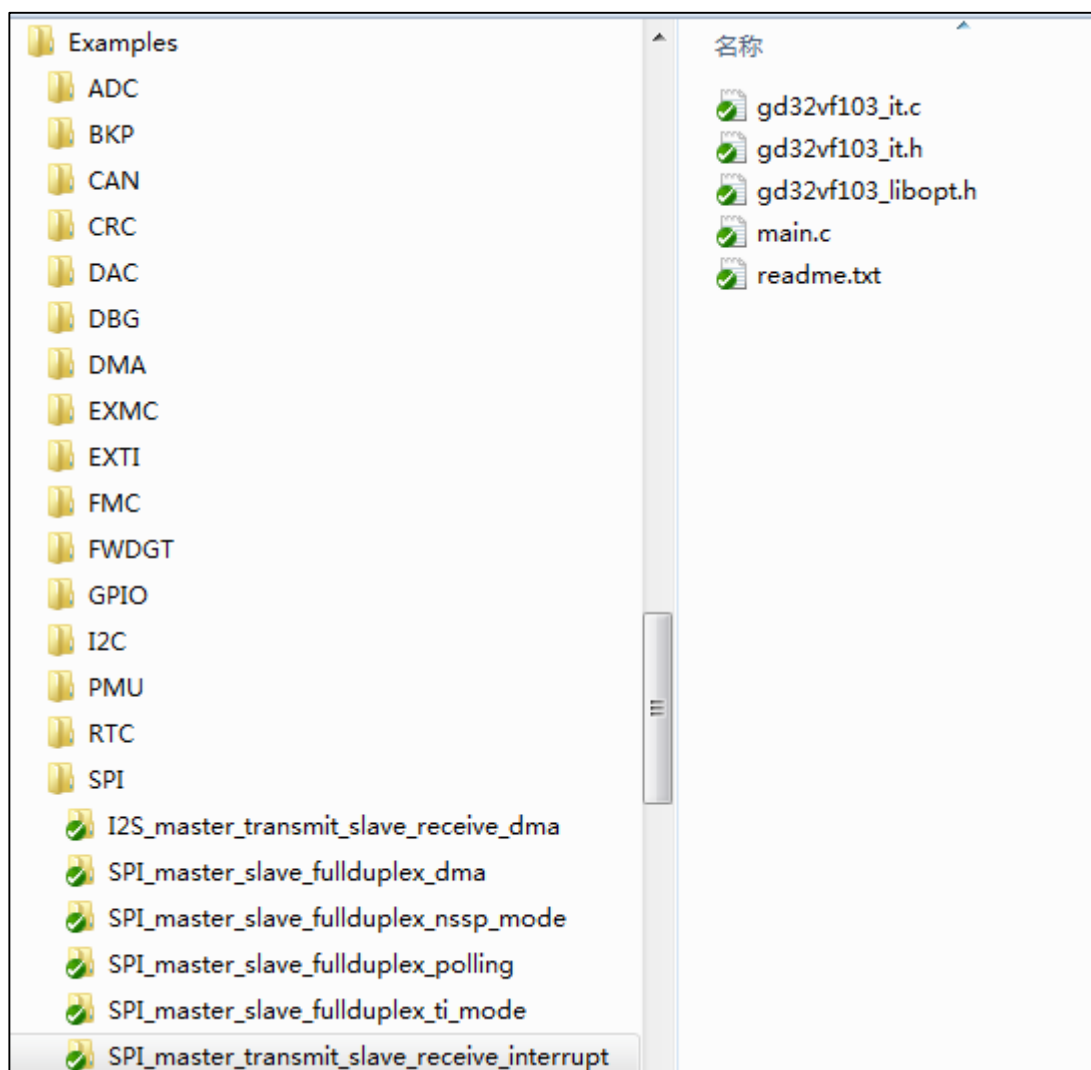
## 2.1.4. Template Folder

Template folder includes a simple demo of how to use LED (IAR\_project run in IAR, Eclipse\_project run in Eclipse). User can use the project template to compile the formware examples, the steps are shown as below:

### Select files

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

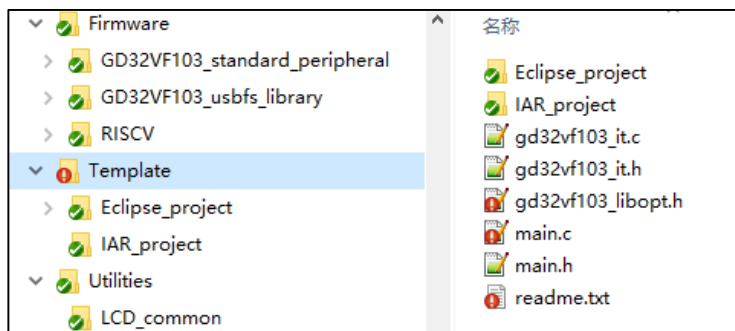
**Figure 2-2. Select peripheral example files**



## Copy files

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

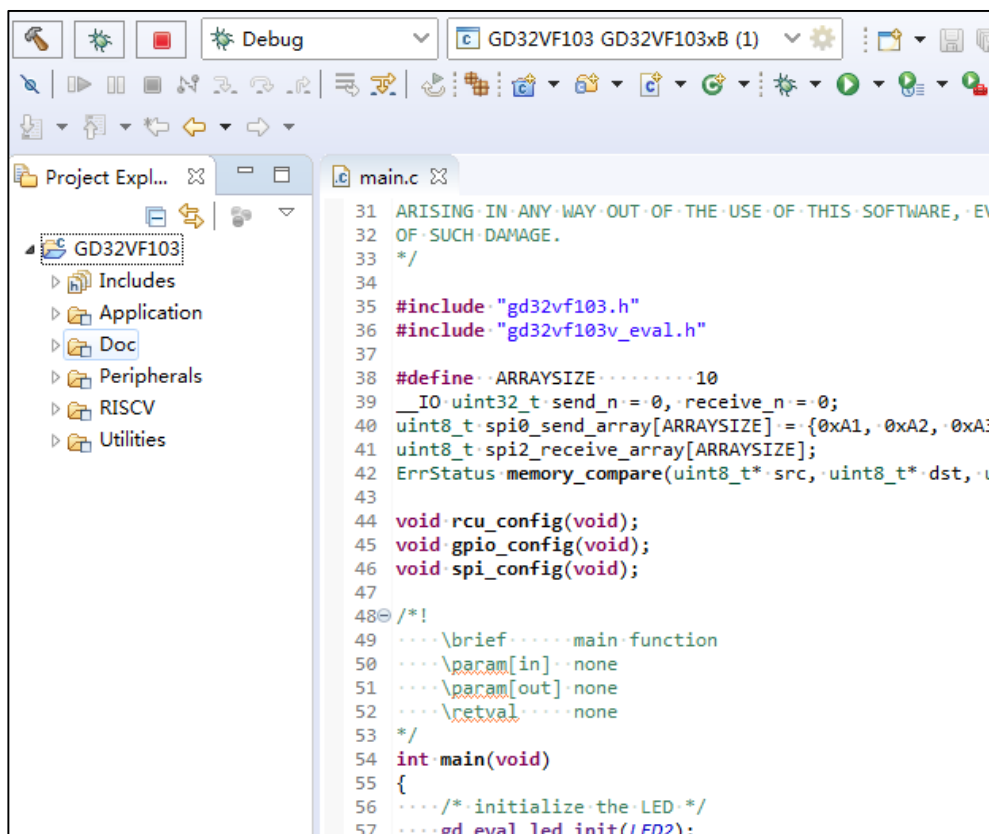
**Figure 2-3. Copy the peripheral example files**



## Open a project

GD provides Eclipse and IAR versions of the project. Depending on the software installed, user can open different projects. For example, open Eclipse and import project in Template folder, shown as below:

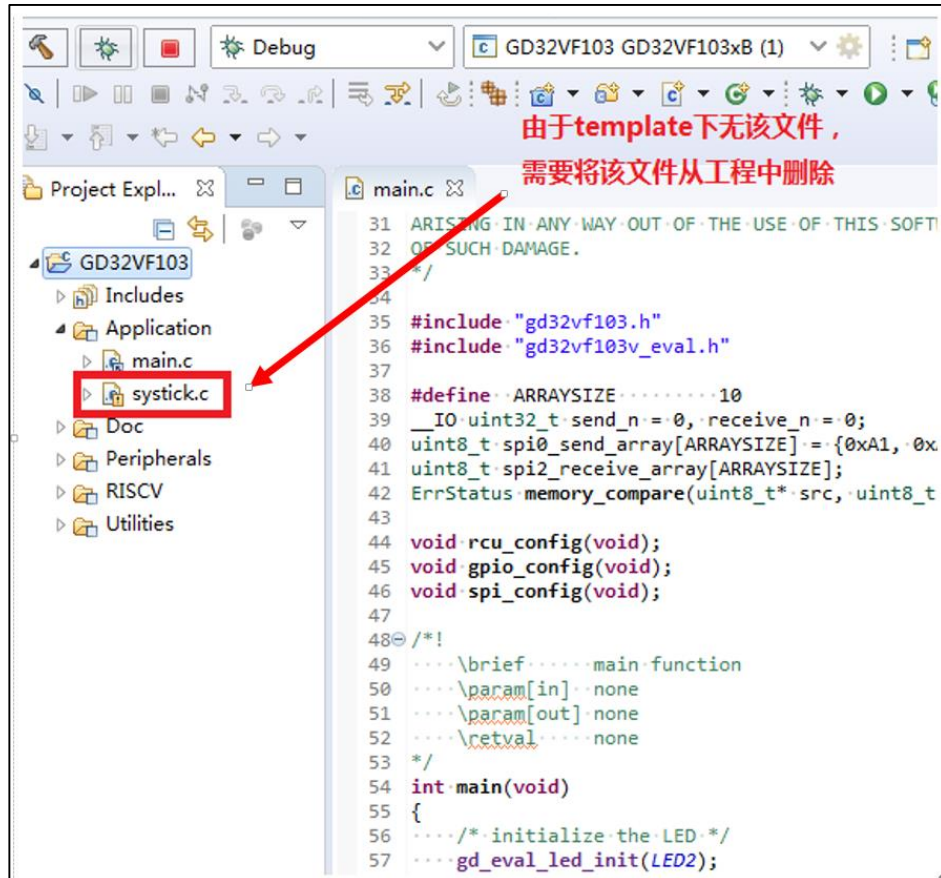
**Figure 2-4. Open the project file**





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:

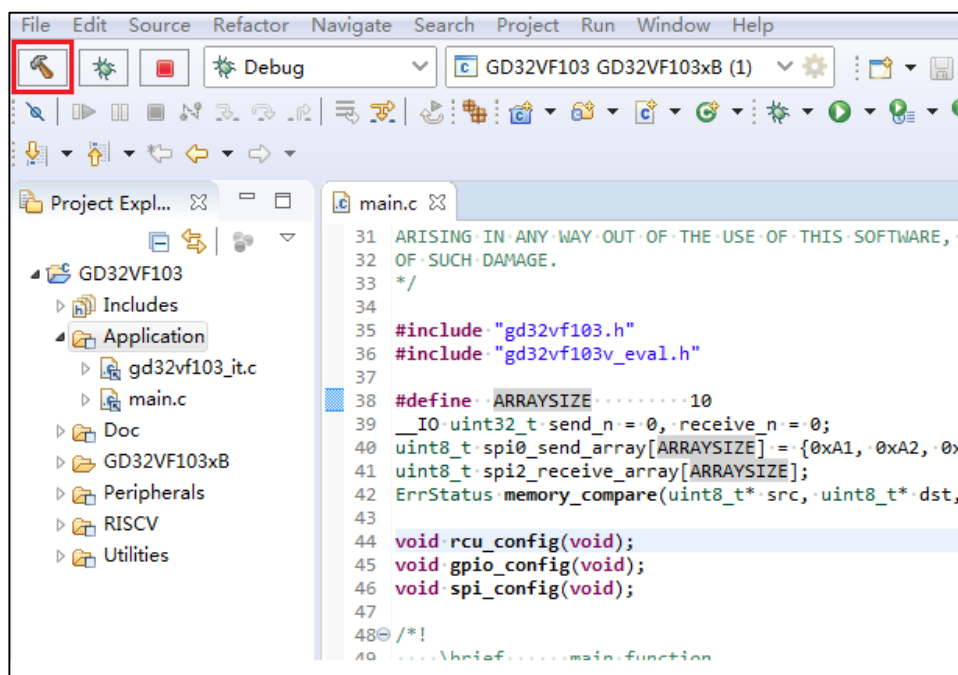
**Figure 2-5. Configure project files**



## 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. Compile operation is shown as below. Debug/download and other specific usage of IDE can refer to corresponding Eclipse software user guide.

**Figure 2-6. Compile**



## 2.1.5. Utilities Folder

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

- LCD\_Commom subfolder;
- gd32vf103\_eval.h and gd32vf103\_lcd\_eval.h are related header files of the evaluation board about running the firmware examples;
- gd32vf103\_eval.c and gd32vf103\_lcd\_eval.c are related source files of the evaluation board about running the firmware examples.

## 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
gd32vf103_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.
gd32vf103_it.h	Header file, including all the prototypes of interrupt service routines.
gd32vf103_it.c	Source files about interruptut service routines of peripherals. User can written

Files	Descriptions
	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.
gd32vf103_xxx.h	The header file of peripheral PPP, including functions about peripheral PPP, and the variables used for functions.
gd32vf103_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

Registers	Descriptions
ADC_CTL0	Control register 0
ADC_CTL1	Control register 1
ADC_SAMPT0	Sample time register 0
ADC_SAMPT1	Sample time register 1
ADC_IOFFx	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_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_enable	enable ADC interface
adc_disable	disable ADC interface
adc_calibration_enable	ADC calibration and reset calibration
adc_tempsensor_vrefint_enable	enable the temperature sensor and Vrefint channel
adc_tempsensor_vrefint_disable	disable the temperature sensor and Vrefint channel
adc_dma_mode_enable	enable DMA request
adc_dma_mode_disable	disable DMA request
adc_discontinuous_mode_config	configure ADC discontinuous mode
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_source_config	configure ADC external trigger source
adc_external_trigger_config	enable ADC external trigger
adc_software_trigger_enable	enable ADC software trigger
adc_regular_data_read	read ADC regular group data register

Function name	Function description
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_watchdog_single_channel_enable	configure ADC analog watchdog single channel
adc_watchdog_group_channel_enable	configure ADC analog watchdog group channel
adc_watchdog_disable	disable ADC analog watchdog
adc_watchdog_threshold_config	configure ADC analog watchdog threshold
adc_flag_get	get the ADC flag bits
adc_flag_clear	clear the ADC flag bits
adc_regular_software_startconv_flag_get	get the bit state of ADCx software start conversion
adc_inserted_software_startconv_flag_get	get the bit state of ADCx software inserted channel start conversion
adc_interrupt_flag_get	get the ADC interrupt bits
adc_interrupt_flag_clear	clear the ADC flag
adc_interrupt_enable	enable ADC interrupt
adc_interrupt_disable	disable ADC interrupt
adc_resolution_config	configure ADC resolution
adc_oversample_mode_config	configure ADC oversample mode
adc_oversample_mode_enable	enable ADC oversample mode
adc_oversample_mode_disable	disable ADC oversample mode

## adc\_deinit

The description of adc\_deinit is shown as below:

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

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

Example:

```
/* reset ADC0 */
```

```
adc_deinit(ADC0);
```

## adc\_mode\_config

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

**Table 3-5. 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 ADC sync mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mode</b>	ADC mode
ADC_MODE_FREE	all the ADCs work independently
ADC_DAUL_REGULAL _PARALLEL_INSERTE D_PARALLEL	ADC0 and ADC1 work in combined regular parallel + inserted parallel mode
ADC_DAUL_REGULAL _PARALLEL_INSERTE D_ROTATION	ADC0 and ADC1 work in combined regular parallel + trigger rotation mode
ADC_DAUL_INSERTE D_PARALLEL_REGUL AL_FOLLOWUP_FAST	ADC0 and ADC1 work in combined inserted parallel + follow-up fast mode
ADC_DAUL_INSERTE D_PARALLEL_REGUL AL_FOLLOWUP_SLO W	ADC0 and ADC1 work in combined inserted parallel + follow-up slow mode
ADC_DAUL_INSERTE D_PARALLEL	ADC0 and ADC1 work in inserted parallel mode only
ADC_DAUL_REGULAL _PARALLEL	ADC0 and ADC1 work in regular parallel mode only
ADC_DAUL_REGULAL _FOLLOWUP_FAST	ADC0 and ADC1 work in follow-up fast mode only
ADC_DAUL_REGULAL _FOLLOWUP_SLOW	ADC0 and ADC1 work in follow-up slow mode only
ADC_DAUL_INSERTE D_TRIGGER_ROTATI ON	ADC0 and ADC1 work in trigger rotation mode only
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* configure the ADC sync mode: regular parallel + inserted parallel mode */
```

```
adc_mode_config(ADC_DUAL_REGULAR_PARALLEL_INSERTED_PARALLEL);
```

### adc\_special\_function\_config

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

**Table 3-6. Function adc\_special\_function\_config**

<b>Function name</b>	adc_special_function_config
<b>Function prototype</b>	void adc_special_function_config(uint32_t adc_periph, uint32_t function, ControlStatus newvalue);
<b>Function descriptions</b>	enable or disable ADC special function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC periph
ADCx(x=0,1)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>function</b>	the function to config
ADC_SCAN_MODE	scan mode select
ADC_INSERTED_CHANNEL_AUTO	inserted channel group convert automatically
ADC_CONTINUOUS_MODE	continuous mode select
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
ENABLE	enable function
DISABLE	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### adc\_data\_alignment\_config

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



Table 3-7. Function `adc_data_alignment_config`

Function name	<code>adc_data_alignment_config</code>
Function prototype	<code>void adc_data_alignment_config(uint32_t adc_periph, uint32_t data_alignment);</code>
Function descriptions	configure ADCx data alignment
Precondition	-
The called functions	-
Input parameter{in}	
<code>adc_periph</code>	ADC periph
<code>ADCx(x=0,1)</code>	ADC peripheral selection
Input parameter{in}	
<code>data_alignment</code>	data alignment select
<code>ADC_DATAALIGN_RIGHT</code>	LSB alignment
<code>ADC_DATAALIGN_LEFT</code>	MSB alignment
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

### **adc\_enable**

The description of `adc_enable` is shown as below:

Table 3-8. Function `adc_enable`

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

Example:

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

### adc\_disable

The description of adc\_disable is shown as below:

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

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

Example:

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

### adc\_calibration\_enable

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

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

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

Example:

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

```
adc_calibration_enable(ADC0);
```

### adc\_tempsensor\_vrefint\_enable

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

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

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

Example:

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

```
adc_tempsensor_vrefint_enable();
```

### adc\_tempsensor\_vrefint\_disable

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

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

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

Example:

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

```
adc_tempsensor_vrefint_disable();
```

### adc\_dma\_mode\_enable

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

**Table 3-13. Function adc\_dma\_mode\_enable**

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

Example:

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

```
adc_dma_mode_enable(ADC0);
```

### adc\_dma\_mode\_disable

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

**Table 3-14. Function adc\_dma\_mode\_disable**

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

Example:

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

```
adc_dma_mode_disable(ADC0);
```

### adc\_discontinuous\_mode\_config

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

**Table 3-15. 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 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 periph
<i>ADCx(x=0,1)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_channel_group</b>	select the channel group
<i>ADC_REGULAR_CHANNEL</i>	regular channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
<i>ADC_CHANNEL_DISCONTINUOUS_DISABLE</i>	disable discontinuous mode of regular and inserted channel
<b>Input parameter{in}</b>	
<b>length</b>	number of conversions in discontinuous mode, the number can be 1..8 for regular channel, the number has no effect for inserted channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### adc\_channel\_length\_config

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

**Table 3-16. 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

	channel_group, uint32_t length);
<b>Function descriptions</b>	configure the length of regular channel group or inserted channel group
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC periph
<i>ADCx(x=0,1)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>channel_group</b>	select the channel group
<i>ADC_REGULAR_CHANNEL</i>	regular channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
<b>Input parameter{in}</b>	
<b>length</b>	the length of the channel, regular channel 1-16, inserted channel 1-4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### adc\_regular\_channel\_config

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

**Table 3-17. 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 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 periph
<i>ADCx(x=0,1)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>rank</b>	the regular group sequence rank, this parameter must be between 0 to 15
<b>Input parameter{in}</b>	
<b>channel</b>	the selected ADC channel
<i>ADC_CHANNEL_x</i>	ADC Channelx (x=0..17)(x=16 and x=17 are only for ADC0)

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

Example:

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

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

### adc\_inserted\_channel\_config

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

**Table 3-18. 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 channel, uint32_t sample_time);
<b>Function descriptions</b>	configure ADC inserted channel
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>adc_periph</b>	ADC periph
ADCx(x=0,1)	ADC peripheral selection
Input parameter{in}	
<b>rank</b>	the inserted group sequencer rank, this parameter must be between 0 to 3

Input parameter{in}	
<b>channel</b>	the selected ADC channel
<i>ADC_CHANNEL_x</i>	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_1 POINT5</i>	1.5 cycles
<i>ADC_SAMPLETIME_7 POINT5</i>	7.5 cycles
<i>ADC_SAMPLETIME_1 3POINT5</i>	13.5 cycles
<i>ADC_SAMPLETIME_2 8POINT5</i>	28.5 cycles
<i>ADC_SAMPLETIME_4 1POINT5</i>	41.5 cycles
<i>ADC_SAMPLETIME_5 5POINT5</i>	55.5 cycles
<i>ADC_SAMPLETIME_7 1POINT5</i>	71.5 cycles
<i>ADC_SAMPLETIME_2 39POINT5</i>	239.5 cycles
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

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

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

**Table 3-19. 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>	-
Input parameter{in}	
<b>adc_periph</b>	ADC periph



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

Example:

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

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

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

The description of `adc_external_trigger_source_config` is shown as below:

**Table 3-20. Function `adc_external_trigger_source_config`**

<b>Function name</b>	<code>adc_external_trigger_source_config</code>
<b>Function prototype</b>	<code>void adc_external_trigger_source_config(uint32_t adc_periph, uint8_t channel_group, uint32_t external_trigger_source);</code>
<b>Function descriptions</b>	configure ADC external trigger source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC periph
<i>ADCx(x=0,1)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>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>external_trigger_source</b>	regular or inserted group trigger source
<i>ADC0_1_EXTTRIGGER_REGULAR_T0_CH0</i>	TIMER0 CH0 event select for regular channel
<i>ADC0_1_EXTTRIGGER_REGULAR_T0_CH1</i>	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_TRGO	TIMER2 TRGO event select for regular channel
ADC0_1_EXTTRIG_REGULAR_T3_CH3	TIMER3 CH3 event select for regular channel
ADC0_1_EXTTRIG_REGULAR_EXTI_11	external interrupt line 11 for regular channel
ADC0_1_EXTTRIG_REGULAR_NONE	software trigger for regular channel
ADC0_1_EXTTRIG_INSERTED_T0_TRGO	TIMER0 TRGO event select for inserted channel
ADC0_1_EXTTRIG_INSERTED_T0_CH3	TIMER0 CH3 event select for inserted channel
ADC0_1_EXTTRIG_INSERTED_T1_TRGO	TIMER1 TRGO event select for inserted channel
ADC0_1_EXTTRIG_INSERTED_T1_CH0	TIMER1 CH0 event select for inserted channel
ADC0_1_EXTTRIG_INSERTED_T2_CH3	TIMER2 CH3 event select for inserted channel
ADC0_1_EXTTRIG_INSERTED_T3_TRGO	TIMER3 TRGO event select for inserted channel
ADC0_1_EXTTRIG_INSERTED_EXTI_15	external interrupt line 15 for inserted channel
ADC0_1_EXTTRIG_INSERTED_NONE	software trigger for inserted channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure 0 regular channel external trigger source */
```

```
adc_external_trigger_source_config(ADC0,ADC_REGULAR_CHANNEL,
ADC_EXTTRIG_REGULAR_T0_CH0);
```

### adc\_external\_trigger\_config

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

Table 3-21. Function `adc_external_trigger_config`

Function name	<code>adc_external_trigger_config</code>
Function prototype	<code>void adc_external_trigger_config(uint32_t adc_periph, uint8_t channel_group, ControlStatus newvalue);</code>
Function descriptions	configure ADC external trigger
Precondition	-
The called functions	-
Input parameter{in}	
<code>adc_periph</code>	ADC periph
<code>ADCx(x=0,1)</code>	ADC peripheral selection
Input parameter{in}	
<code>channel_group</code>	select the channel group
<code>ADC_REGULAR_CHANNEL</code>	regular channel group
<code>ADC_INSERTED_CHANNEL</code>	inserted channel group
Input parameter{in}	
<code>newvalue</code>	control value
<code>ENABLE</code>	enable function
<code>DISABLE</code>	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

### `adc_software_trigger_enable`

The description of `adc_software_trigger_enable` is shown as below:

Table 3-22. Function `adc_software_trigger_enable`

Function name	<code>adc_software_trigger_enable</code>
Function prototype	<code>void adc_software_trigger_enable(uint32_t adc_periph, uint8_t channel_group);</code>
Function descriptions	enable ADC software trigger
Precondition	-
The called functions	-
Input parameter{in}	
<code>adc_periph</code>	ADC periph
<code>ADCx(x=0,1)</code>	ADC peripheral selection

Input parameter{in}	
<b>channel_group</b>	select the channel group
<i>ADC_REGULAR_CHANNEL</i>	regular channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
Output parameter{out}	
-	-
Return value	
-	-

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 3-23. 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 periph
<i>ADCx(x=0,1)</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 3-24. Function `adc_inserted_data_read`

Function name	<code>adc_inserted_data_read</code>
Function prototype	<code>uint16_t adc_inserted_data_read(uint32_t adc_periph, uint8_t inserted_channel);</code>
Function descriptions	read ADC inserted group data register
Precondition	-
The called functions	-
Input parameter{in}	
<code>adc_periph</code>	ADC periph
<code>ADCx(x=0,1)</code>	ADC peripheral selection
Input parameter{in}	
<code>inserted_channel</code>	insert channel select
<code>ADC_INSERTED_CHANNEL_x</code>	inserted Channelx, x=0,1,2,3
Output parameter{out}	
-	-
Return value	
<code>uint16_t</code>	ADC conversion value (0-0xFFFF)

Example:

```
/* read ADC 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 3-25. Function `adc_sync_mode_convert_value_read`

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

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

**Table 3-26. 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 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 periph
<i>ADCx(x=0,1)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	the selected ADC channel
<i>ADC_CHANNEL_x</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 3-27. 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 channel_group);
<b>Function descriptions</b>	configure ADC analog watchdog group channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC periph
<i>ADCx(x=0,1)</i>	ADC peripheral selection

Input parameter{in}	
<b>channel_group</b>	the channel group use analog watchdog
<i>ADC_REGULAR_CHANNEL</i>	regular channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
<i>ADC_REGULAR_INSERTED_CHANNEL</i>	both regular and inserted group
Output parameter{out}	
-	-
Return value	
-	-

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 3-28. 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>	-
Input parameter{in}	
<b>adc_periph</b>	ADC periph
<i>ADCx(x=0,1)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
adc_watchdog_disable(ADC0);
```

### adc\_watchdog\_threshold\_config

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

Table 3-29. Function `adc_watchdog_threshold_config`

Function name	<code>adc_watchdog_threshold_config</code>
Function prototype	<code>void adc_watchdog_threshold_config(uint32_t adc_periph, uint16_t low_threshold, uint16_t high_threshold);</code>
Function descriptions	configure ADC analog watchdog threshold
Precondition	-
The called functions	-
Input parameter{in}	
<code>adc_periph</code>	ADC periph
<code>ADCx(x=0,1)</code>	ADC peripheral selection
Input parameter{in}	
<code>low_threshold</code>	analog watchdog low threshold, 0..4095
Input parameter{in}	
<code>high_threshold</code>	analog watchdog high threshold, 0..4095
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC analog watchdog threshold */
```

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

### `adc_flag_get`

The description of `adc_flag_get` is shown as below:

Table 3-30. Function `adc_flag_get`

Function name	<code>adc_flag_get</code>
Function prototype	<code>FlagStatus adc_flag_get(uint32_t adc_periph, uint32_t flag);</code>
Function descriptions	get the ADC flag bits
Precondition	-
The called functions	-
Input parameter{in}	
<code>adc_periph</code>	ADC periph
<code>ADCx(x=0,1)</code>	ADC peripheral selection
Input parameter{in}	
<code>flag</code>	the adc flag bits
<code>ADC_FLAG_WDE</code>	analog watchdog event flag
<code>ADC_FLAG_EOC</code>	end of group conversion flag
<code>ADC_FLAG_EOIC</code>	end of inserted group conversion flag
<code>ADC_FLAG_STIC</code>	start flag of inserted channel group
<code>ADC_FLAG_STRC</code>	start flag of regular channel group



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

Example:

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

```
FlagStatus flag_value;
```

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

### adc\_flag\_clear

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

**Table 3-31. Function adc\_flag\_clear**

Function name	adc_flag_clear
Function prototype	void adc_flag_clear(uint32_t adc_periph, uint32_t flag);
Function descriptions	clear the ADC flag bits
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC periph
ADCx(x=0,1)	ADC peripheral selection
Input parameter{in}	
flag	the adc flag bits
ADC_FLAG_WDE	analog watchdog event flag
ADC_FLAG_EOC	end of group conversion flag
ADC_FLAG_EOIC	end of inserted group conversion flag
ADC_FLAG_STIC	start flag of inserted channel group
ADC_FLAG_STRC	start flag of regular channel group
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

### adc\_regular\_software\_startconv\_flag\_get

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

Table 3-32. Function `adc_regular_software_startconv_flag_get`

<b>Function name</b>	<code>adc_regular_software_startconv_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus adc_regular_software_startconv_flag_get(uint32_t adc_periph);</code>
<b>Function descriptions</b>	get the bit state of ADCx software start conversion
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC periph
<i>ADCx(x=0,1)</i>	ADC peripheral selection
<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_regular_software_startconv_flag_get(ADC0,ADC_FLAG_WDE);
```

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

The description of `adc_inserted_software_startconv_flag_get` is shown as below:

Table 3-33. Function `adc_inserted_software_startconv_flag_get`

<b>Function name</b>	<code>adc_inserted_software_startconv_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus adc_inserted_software_startconv_flag_get(uint32_t adc_periph);</code>
<b>Function descriptions</b>	get the bit state of ADCx software inserted channel start conversion
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC periph
<i>ADCx(x=0,1)</i>	ADC peripheral selection
<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_inserted_software_startconv_flag_get(ADC0, ADC_FLAG_WDE);
```

## adc\_interrupt\_flag\_get

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

**Table 3-34. Function adc\_interrupt\_flag\_get**

<b>Function name</b>	adc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus adc_interrupt_flag_get(uint32_t adc_periph, uint32_t flag);
<b>Function descriptions</b>	get the ADC interrupt bits
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC periph
ADCx(x=0,1)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>flag</b>	the adc interrupt bits
ADC_INT_FLAG_WDE	analog watchdog interrupt
ADC_INT_FLAG_EOC	end of group conversion interrupt
ADC_INT_FLAG_EOIC	end of inserted group conversion interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

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

```
FlagStatus flag_value;
```

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

## adc\_interrupt\_flag\_clear

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

**Table 3-35. Function adc\_interrupt\_flag\_clear**

<b>Function name</b>	adc_interrupt_flag_clear
<b>Function prototype</b>	void adc_interrupt_flag_clear(uint32_t adc_periph, uint32_t flag);
<b>Function descriptions</b>	clear the ADC interrupt bits
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC periph
ADCx(x=0,1)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>flag</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_FLAG_WDE);
```

### adc\_interrupt\_enable

The description of `adc_interrupt_enable` is shown as below:

**Table 3-36. Function `adc_interrupt_enable`**

<b>Function name</b>	<code>adc_interrupt_enable</code>
<b>Function prototype</b>	<code>void adc_interrupt_enable(uint32_t adc_periph, uint32_t interrupt);</code>
<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 periph
<i>ADCx(x=0,1)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>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 3-37. Function `adc_interrupt_disable`

<b>Function name</b>	<code>adc_interrupt_disable</code>
<b>Function prototype</b>	<code>void adc_interrupt_disable(uint32_t adc_periph, uint32_t interrupt);</code>
<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 periph
<code>ADCx(x=0,1)</code>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>interrupt</b>	the adc interrupt
<code>ADC_INT_WDE</code>	analog watchdog interrupt
<code>ADC_INT_EOC</code>	end of group conversion interrupt
<code>ADC_INT_EOIC</code>	end of inserted group conversion interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### **adc\_resolution\_config**

The description of `adc_resolution_config` is shown as below:

Table 3-38. Function `adc_resolution_config`

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

<i>B</i>	
ADC_RESOLUTION_6	6-bit ADC resolution
<i>B</i>	
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### adc\_oversample\_mode\_config

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

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

<b>Function name</b>	adc_oversample_mode_config
<b>Function prototype</b>	void adc_oversample_mode_config(uint32_t adc_periph, uint32_t mode, uint16_t shift, uint8_t ratio);
<b>Function descriptions</b>	configure ADC oversample mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC periph
ADCx(x=0,1)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>mode</b>	ADC oversampling mode
ADC_OVERSAMPLING_ALL_CONVERT	all oversampled conversions for a channel are done consecutively after a trigger
ADC_OVERSAMPLING_ONE_CONVERT	each oversampled conversion for a channel needs a trigger
<b>Input parameter{in}</b>	
<b>shift</b>	ADC oversampling shift
ADC_OVERSAMPLING_SHIFT_NONE	no oversampling shift
ADC_OVERSAMPLING_SHIFT_1B	1-bit oversampling shift
ADC_OVERSAMPLING_SHIFT_2B	2-bit oversampling shift
ADC_OVERSAMPLING_SHIFT_3B	3-bit oversampling shift
ADC_OVERSAMPLING_SHIFT_4B	4-bit oversampling shift

ADC_OVERSAMPLING_SHIFT_4B	5-bit oversampling shift
ADC_OVERSAMPLING_SHIFT_5B	6-bit oversampling shift
ADC_OVERSAMPLING_SHIFT_6B	7-bit oversampling shift
ADC_OVERSAMPLING_SHIFT_7B	8-bit oversampling shift
ADC_OVERSAMPLING_SHIFT_8B	
<b>Input parameter{in}</b>	
ratio	ADC oversampling ratio
ADC_OVERSAMPLING_RATIO_MUL2	oversampling ratio multiple 2
ADC_OVERSAMPLING_RATIO_MUL4	oversampling ratio multiple 4
ADC_OVERSAMPLING_RATIO_MUL8	oversampling ratio multiple 8
ADC_OVERSAMPLING_RATIO_MUL16	oversampling ratio multiple 16
ADC_OVERSAMPLING_RATIO_MUL32	oversampling ratio multiple 32
ADC_OVERSAMPLING_RATIO_MUL64	oversampling ratio multiple 64
ADC_OVERSAMPLING_RATIO_MUL128	oversampling ratio multiple 128
ADC_OVERSAMPLING_RATIO_MUL256	oversampling ratio multiple 256
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
adc_oversample_mode_config(ADC0,ADC_OVERSAMPLING_ALL_CONVERT,
ADC_OVERSAMPLING_SHIFT_4B, ADC_OVERSAMPLING_RATIO_MUL16);
```

### adc\_oversample\_mode\_enable

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

**Table 3-40. Function adc\_oversample\_mode\_enable**

Function name	adc_oversample_mode_enable
---------------	----------------------------

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

Example:

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

### adc\_oversample\_mode\_disable

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

**Table 3-41. Function adc\_oversample\_mode\_disable**

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

Example:

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

## 3.3. BKP

The Backup registers are located in the Backup domain that remains powered-on by V<sub>BAT</sub> even if V<sub>DD</sub> power is shut down, they are forty-two 16-bit (84 bytes) registers for data



protection of user application data, and the wake-up action from Standby mode or system reset do not affect these registers. The BKP registers are listed in chapter [3.3.1](#), the BKP firmware functions are introduced in chapter [3.3.2](#).

### 3.3.1. Descriptions of Peripheral registers

BKP registers are listed in the table shown as below:

**Table 3-42. BKP Registers**

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

### 3.3.2. Descriptions of Peripheral functions

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

**Table 3-43. BKP firmware function**

Function name	Function description
bkp_deinit	reset BKP registers
bkp_data_write	write BKP data register
bkp_data_read	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
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_interrupt_enable	enable tamper interrupt
bkp_interrupt_disable	disable tamper interrupt
bkp_flag_get	get tamper flag state
bkp_flag_clear	clear tamper flag state
bkp_interrupt_flag_get	get tamper interrupt flag state
bkp_interrupt_flag_clear	clear tamper interrupt flag state

## Enum bkp\_data\_register\_enum

**Table 3-44. Enum bkp\_data\_register\_enum**

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

enum name	enum description
BKP_DATA_38	BKP data register 38
BKP_DATA_39	BKP data register 39
BKP_DATA_40	BKP data register 40
BKP_DATA_41	BKP data register 41

## bkp\_deinit

The description of bkp\_deinit is shown as below:

**Table 3-45. Function bkp\_deinit**

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

Example:

```
/* BKP deinitialize*/
```

```
bkp_deinit ();
```

## bkp\_data\_write

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

**Table 3-46. Function bkp\_data\_write**

<b>Function name</b>	bkp_data_write
<b>Function prototype</b>	void bkp_data_write(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 <a href="#">Table 3-44. Enum bkp_data_register_enum</a>
<i>BKP_DATA_x</i>	bkp data register number x, x = 0..41
<b>Input parameter{in}</b>	
<b>data</b>	the data to be write in BKP data register, 0 – 0xFF
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* write BKP data register 0 */
bkp_data_write(BKP_DATA_0, 0x55);
```

### bkp\_data\_read

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

**Table 3-47. Function bkp\_data\_read**

<b>Function name</b>	bkp_data_read
<b>Function prototype</b>	uint16_t bkp_data_read(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 <a href="#">Table 3-44. Enum bkp_data_register_enum</a>
<i>BKP_DATA_x</i>	bkp data register number x, x = 0..41
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	data of BKP data register, 0 – 0xFF

Example:

```
uint16_t bkp_data0 = 0;
/* write BKP data register 0 */
bkp_data0 = bkp_data_read (BKP_DATA_0);
```

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

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

**Table 3-48. 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>	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
bkp_rtc_calibration_output_enable();
```

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

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

**Table 3-49. Function bkp\_rtc\_calibration\_output\_disable**

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

Example:

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

```
bkp_rtc_calibration_output_disable ();
```

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

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

**Table 3-50. Function bkp\_rtc\_signal\_output\_enable**

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

-	-
Return value	
-	-

Example:

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

```
bkp_rtc_signal_output_enable ();
```

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

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

**Table 3-51. 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 3-52. 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
<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_SECOND_PULSE</i>	RTC second pulse is selected as the RTC output
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* selecte RTC second pulse as the RTC output */
```

```
bkp_rtc_output_select (RTC_OUTPUT_SECOND_PULSE);
```

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

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

**Table 3-53. 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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
bkp_rtc_calibration_value_set (0x30);
```

### bkp\_tamper\_detection\_enable

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

**Table 3-54. 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>	

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

Example:

```
/* enable tamper detection */
```

```
bkp_tamper_detection_enable ();
```

### bkp\_tamper\_detection\_disable

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

**Table 3-55. Function bkp\_tamper\_detection\_disable**

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

Example:

```
/* disable tamper detection */
```

```
bkp_tamper_detection_disable ();
```

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

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

**Table 3-56. Function bkp\_tamper\_active\_level\_set**

Function name	bkp_tamper_active_level_set
Function prototype	void bkp_tamper_active_level_set(uint16_t level);
Function descriptions	set tamper pin active level
Precondition	-
The called functions	-
Input parameter{in}	
level	tamper active level



TAMPER_PIN_ACTIVE_HIGH	the tamper pin is active high
TAMPER_PIN_ACTIVE_LOW	the tamper pin is active low
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
bkp_tamper_active_level_set (TAMPER_PIN_ACTIVE_HIGH);
```

### bkp\_interrupt\_enable

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

**Table 3-57. Function bkp\_interrupt\_enable**

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

Example:

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

```
bkp_interrupt_enable ();
```

### bkp\_interrupt\_disable

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

**Table 3-58. Function bkp\_interrupt\_disable**

Function name	bkp_interrupt_disable
Function prototype	void bkp_interrupt_disable(void);
Function descriptions	disable tamper interrupt
Precondition	-

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

Example:

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

```
bkp_interrupt_disable ();
```

### bkp\_flag\_get

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

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

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

Example:

```
FlagStatus bkp_flag = RESET;
```

```
/* get tamper flag state */
```

```
bkp_flag = bkp_flag_get ();
```

### bkp\_flag\_clear

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

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

Function name	bkp_flag_clear
Function prototype	void bkp_flag_clear(void);
Function descriptions	clear tamper flag state
Precondition	-

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

Example:

```
/* clear tamper flag state */
```

```
bkp_flag_clear ();
```

### bkp\_interrupt\_flag\_get

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

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

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

Example:

```
FlagStatus bkp_flag = RESET;
```

```
/* get tamper interrupt flag state */
```

```
bkp_flag = bkp_interrupt_flag_get ();
```

### bkp\_interrupt\_flag\_clear

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

**Table 3-62. Function bkp\_interrupt\_flag\_clear**

Function name	bkp_interrupt_flag_clear
Function prototype	void bkp_interrupt_flag_clear(void);
Function descriptions	clear tamper interrupt flag state
Precondition	-

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

Example:

```
/* clear tamper interrupt flag state */
```

```
bkp_interrupt_flag_clear ();
```

## 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 3-63. 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
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

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

### 3.4.2. Descriptions of Peripheral functions

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

**Table 3-64. 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
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 3-65. Structure can\_parameter\_struct**

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

## Structure can\_transmit\_message\_struct

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

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

## Structure can\_receive\_message\_struct

**Table 3-67. Structure can\_receive\_message\_struct**

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

## Structure can\_filter\_parameter\_struct

**Table 3-68. Structure can\_filter\_parameter\_struct**

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

## can\_deinit

The description of can\_deinit is shown as below:

**Table 3-69. Function can\_deinit**

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

Example:

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

```
can_deinit (CAN0);
```

## can\_struct\_para\_init

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

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

Function name	can_struct_para_init
Function prototype	void can_struct_para_init(can_struct_type_enum type, void* p_struct)
Function descriptions	initialize CAN parameter struct with a default value

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>type</b>	CAN peripheral
<i>CAN_INIT_STRUCT</i>	CAN initialize parameters struct
<i>CAN_FILTER_STRUCT</i>	CAN filter parameters struct
<i>CAN_TX_MESSAGE_STRUCT</i>	CAN transmit message struct
<i>CAN_RX_MESSAGE_STRUCT</i>	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 3-71. Function can\_init**

<b>Function name</b>	can_init
<b>Function prototype</b>	ErrStatus can_init(uint32_t can_periph, can_parameter_struct* can_parameter_init);
<b>Function descriptions</b>	initialize CAN
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>can_parameter_init</b>	CAN parameter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-65. Structure can_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS / ERROR

Example:



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

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

### can\_filter\_init

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

**Table 3-72. 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>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_filter_parameter_init</b>	CAN filter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-68. Structure can_filter_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### can1\_filter\_start\_bank

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

**Table 3-73. 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 3-74. Function can\_debug\_freeze\_enable**

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

Example:

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

```
can_debug_freeze_enable (CAN0);
```

### can\_debug\_freeze\_disable

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

**Table 3-75. Function can\_debug\_freeze\_disable**

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

Example:

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

```
can_debug_freeze_disable (CAN0);
```

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

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

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

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

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 3-77. Function can\_time\_trigger\_mode\_disable**

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

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 3-78. Function can\_message\_transmit**

<b>Function name</b>	can_message_transmit
<b>Function prototype</b>	uint8_t can_message_transmit(uint32_t can_periph, can_transmit_message_struct* transmit_message);
<b>Function descriptions</b>	transmit CAN message
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>transmit_message</b>	CAN transmit message struct, the structure members can refer to members of the structure <a href="#">Table 3-66. Structure can_transmit_message_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	0x00-0x03

Example:

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

```
uint8_t transmit_mailbox = 0;
```

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

## can\_transmit\_states

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

**Table 3-79. Function can\_transmit\_states**

<b>Function name</b>	can_transmit_states
<b>Function prototype</b>	can_transmit_state_enum can_transmit_states(uint32_t can_periph, uint8_t mailbox_number);
<b>Function descriptions</b>	get CAN transmit state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

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

Example:

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

```
uint8_t transmit_state = 0;
```

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

### can\_transmission\_stop

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

**Table 3-80. Function can\_transmission\_stop**

<b>Function name</b>	can_transmission_stop
<b>Function prototype</b>	void can_transmission_stop(uint32_t can_periph, uint8_t mailbox_number);
<b>Function descriptions</b>	stop CAN transmission
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>mailbox_number</b>	Mailbox number
<i>CAN_MAILBOXx</i>	CAN_MAILBOXx(x=0,1,2)
<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 3-81. Function can\_message\_receive**

<b>Function name</b>	can_message_receive
<b>Function prototype</b>	void can_message_receive(uint32_t can_periph, uint8_t fifo_number, can_receive_message_struct* receive_message);
<b>Function descriptions</b>	CAN receive message
<b>Precondition</b>	can_struct_para_init()
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
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>Input parameter{in}</b>	
<b>receive_message</b>	CAN message receive struct, the structure members can refer to members of the structure <a href="#">Table 3-67. Structure can_receive_message_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

## can\_fifo\_release

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

**Table 3-82. 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
<b>Input parameter{in}</b>	
<b>fifo_number</b>	FIFO number

<i>CAN_FIFOx</i>	CAN_FIFOx(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

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 3-83. Function can\_receive\_message\_length\_get**

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

Example:

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

### can\_working\_mode\_set

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

**Table 3-84. Function can\_working\_mode\_set**

Function name	can_working_mode_set
---------------	----------------------

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

Example:

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

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

### can\_wakeup

The description of can\_wakeup is shown as below:

**Table 3-85. Function can\_wakeup**

<b>Function name</b>	can_wakeup
<b>Function prototype</b>	ErrStatus can_wakeup(uint32_t can_periph);
<b>Function descriptions</b>	wake up CAN
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
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 3-86. Function can\_error\_get**

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

Example:

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

```
can_error_get (CAN0);
```

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

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

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

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

Example:

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

can\_receive\_error\_number\_get (CAN0);

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

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

**Table 3-88. Function can\_transmit\_error\_number\_get**

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

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 3-89. Function can\_interrupt\_enable**

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

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

Example:

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

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

### can\_interrupt\_disable

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

**Table 3-90. Function can\_interrupt\_disable**

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

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

Example:

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

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

### can\_flag\_get

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

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

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

Return value	
FlagStatus	SET / RESET

Example:

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

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

### can\_flag\_clear

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

**Table 3-92. Function can\_flag\_clear**

<b>Function name</b>	can_flag_clear
<b>Function prototype</b>	void can_flag_clear(uint32_t can_periph, can_flag_enum flag);
<b>Function descriptions</b>	clear CAN flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>flag</b>	CAN flags
CAN_FLAG_MTE2	mailbox 2 transmit error
CAN_FLAG_MTE1	mailbox 1 transmit error
CAN_FLAG_MTE0	mailbox 0 transmit error
CAN_FLAG_MTF2	mailbox 2 transmit finished
CAN_FLAG_MTF1	mailbox 1 transmit finished
CAN_FLAG_MTF0	mailbox 0 transmit finished
CAN_FLAG_RFO0	receive FIFO0 overfull
CAN_FLAG_RFF0	receive FIFO0 full
CAN_FLAG_RFO1	receive FIFO1 overfull
CAN_FLAG_RFF1	receive FIFO1 full
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

## can\_interrupt\_flag\_get

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

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

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

Example:

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

## can\_interrupt\_flag\_clear

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

**Table 3-94. 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
<b>CANx(x=0,1)</b>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>flag</b>	CAN interrupt flags
<b>CAN_INT_FLAG_SLPIF</b>	status change interrupt flag of sleep working mode entering
<b>CAN_INT_FLAG_WUIF</b>	status change interrupt flag of wakeup from sleep working mode
<b>CAN_INT_FLAG_ERRIF</b>	error interrupt flag
<b>CAN_INT_FLAG_MTF2</b>	mailbox 2 transmit finished interrupt flag
<b>CAN_INT_FLAG_MTF1</b>	mailbox 1 transmit finished interrupt flag
<b>CAN_INT_FLAG_MTF0</b>	mailbox 0 transmit finished interrupt flag
<b>CAN_INT_FLAG_RFO0</b>	receive FIFO0 overfull interrupt flag
<b>CAN_INT_FLAG_RFF0</b>	receive FIFO0 full interrupt flag
<b>CAN_INT_FLAG_RFO1</b>	receive FIFO1 overfull interrupt flag
<b>CAN_INT_FLAG_RFF1</b>	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 3-95. CRC Registers**

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

### 3.5.2. Descriptions of Peripheral functions

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

**Table 3-96. 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 3-97. 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 3-98. Function `crc_data_register_reset`

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

Example:

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

### `crc_data_register_read`

The description of `crc_data_register_read` is shown as below:

Table 3-99. Function `crc_data_register_read`

Function name	<code>crc_data_register_read</code>
Function prototype	<code>uint32_t crc_data_register_read(void);</code>
Function descriptions	read the value of the data register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
<code>uint32_t</code>	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 3-100. Function `crc_free_data_register_read`

Function name	<code>crc_free_data_register_read</code>
Function prototype	<code>uint8_t crc_free_data_register_read(void);</code>
Function descriptions	read the value of the free data register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
<code>uint8_t</code>	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 3-101. Function `crc_free_data_register_write`

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

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 3-102. Function `crc_single_data_calculate`

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

Example:

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

### `crc_block_data_calculate`

The description of `crc_block_data_calculate` is shown as below:

Table 3-103. Function `crc_block_data_calculate`

Function name	<code>crc_block_data_calculate</code>
Function prototype	<code>uint32_t crc_block_data_calculate(uint32_t array[], uint32_t size);</code>
Function descriptions	calculate the CRC value of an array of 32-bit values
Precondition	-
The called functions	-
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. DAC

The Digital-to-analog converter converts 12-bit digital data to a voltage on the external pins. The DAC registers are listed in chapter [3.6.1](#) the DAC firmware functions are introduced in chapter [3.6.2](#).

### 3.6.1. Peripheral register description

DAC registers are listed in the table shown as below:

**Table 3-104. DAC Registers**

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

### 3.6.2. Descriptions of Peripheral functions

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

**Table 3-105. DAC firmware functions**

Function name	Function description
dac_deinit	deinitialize DAC
dac_enable	enable DAC

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

## dac\_deinit

The description of `dac_deinit` is shown as below:

**Table 3-106. Function `dac_deinit`**

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

Example:

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

```
dac_deinit(DAC0);
```

## **dac\_enable**

The description of `dac_enable` is shown as below:

**Table 3-107. Function `dac_enable`**

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

Example:

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

## **dac\_disable**

The description of `dac_disable` is shown as below:

**Table 3-108. Function `dac_disable`**

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

-	-
Return value	
-	-

Example:

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

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

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

The description of `dac_dma_enable` is shown as below:

**Table 3-109. Function `dac_dma_enable`**

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

Example:

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

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

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

The description of `dac_dma_disable` is shown as below:

**Table 3-110. Function `dac_dma_disable`**

<b>Function name</b>	<code>dac_dma_disable</code>
<b>Function prototype</b>	<code>void dac_dma_disable(uint32_t dac_periph, uint8_t dac_out);</code>
<b>Function descriptions</b>	disable DAC DMA function
<b>Precondition</b>	-
<b>The called functions</b>	-

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

Example:

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

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

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

The description of `dac_output_buffer_enable` is shown as below:

**Table 3-111. Function `dac_output_buffer_enable`**

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

Example:

```
/* enable DAC0_OUT0 output buffer */
```

```
dac_output_buffer_enable(DAC0, DAC_OUT0);
```

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

The description of `dac_output_buffer_disable` is shown as below:



Table 3-112. Function `dac_output_buffer_disable`

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

Example:

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

### `dac_output_value_get`

The description of `dac_output_value_get` is shown as below:

Table 3-113. Function `dac_output_value_get`

Function name	<code>dac_output_value_get</code>
Function prototype	<code>uint16_t dac_output_value_get(uint32_t dac_periph, uint8_t dac_out);</code>
Function descriptions	get DAC output value
Precondition	-
The called functions	-
Input parameter{in}	
<code>dac_periph</code>	DAC peripheral
<code>DACx</code>	DAC peripheral selection(x = 0)
Input parameter{in}	
<code>dac_out</code>	DAC output
<code>DAC_OUTx</code>	DAC output channel selection(x = 0,1)
Output parameter{out}	
-	-
Return value	
<code>uint16_t</code>	DAC output data (0~4095)

Example:

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

```
uint16_t data = 0;
```

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

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

The description of `dac_data_set` is shown as below:

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

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

Example:

```
/* set DAC0_OUT0 data holding register value */
```

```
dac_data_set(DAC0, DAC_OUT0, DAC_ALIGN_8B_R, 0xFF);
```

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

The description of `dac_trigger_enable` is shown as below:

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

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

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

Example:

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

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

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

**Table 3-116. Function dac\_trigger\_disable**

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

Example:

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

```

dac_trigger_disable(DAC0, DAC_OUT0);

```

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

The description of `dac_trigger_source_config` is shown as below:

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

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

Example:

```

/* configure DAC0_OUT0 trigger source */

```

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

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

The description of `dac_software_trigger_enable` is shown as below:

**Table 3-118. Function `dac_software_trigger_enable`**

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

Example:

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

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

The description of `dac_wave_mode_config` is shown as below:

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

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

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

Example:

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

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

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

The description of `dac_lfsr_noise_config` is shown as below:

**Table 3-120. Function `dac_lfsr_noise_config`**

<b>Function name</b>	<code>dac_lfsr_noise_config</code>
<b>Function prototype</b>	<code>void dac_lfsr_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t unmask_bits);</code>
<b>Function descriptions</b>	configure DAC LFSR noise mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
<b>Input parameter{in}</b>	
<b>dac_out</b>	DAC output
<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
<b>Input parameter{in}</b>	
<b>unmask_bits</b>	LFSR noise unmask bits
<i>DAC_LFSR_BIT0</i>	unmask the LFSR bit0
<i>DAC_LFSR_BITSx_0</i>	unmask the LFSR bits [x:0] (x = 1,2,3..11)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

The description of `dac_triangle_noise_config` is shown as below:

**Table 3-121. Function `dac_triangle_noise_config`**

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

Example:

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

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

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

The description of `dac_concurrent_enable` is shown as below:

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

<b>Function name</b>	<code>dac_concurrent_enable</code>
<b>Function prototype</b>	<code>void dac_concurrent_enable(uint32_t dac_periph);</code>
<b>Function descriptions</b>	enable DAC concurrent mode
<b>Precondition</b>	-
<b>The called functions</b>	-

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

Example:

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

```
dac_concurrent_enable(DAC0);
```

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

The description of `dac_concurrent_disable` is shown as below:

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

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

Example:

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

```
dac_concurrent_disable(DAC0);
```

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

The description of `dac_concurrent_software_trigger_enable` is shown as below:

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

<b>Function name</b>	<code>dac_concurrent_software_trigger_enable</code>
<b>Function prototype</b>	<code>void dac_concurrent_software_trigger_enable(uint32_t dac_periph);</code>
<b>Function descriptions</b>	enable DAC concurrent software trigger
<b>Precondition</b>	-



The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
DACx	DAC peripheral selection (x = 0)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

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

**Table 3-125. Function dac\_concurrent\_output\_buffer\_enable**

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

Example:

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

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

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

**Table 3-126. Function dac\_concurrent\_output\_buffer\_disable**

Function name	dac_concurrent_output_buffer_disable
Function prototype	void dac_concurrent_output_buffer_disable(uint32_t dac_periph);
Function descriptions	disable DAC concurrent buffer function

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DAC0 concurrent buffer function */
```

```
dac_concurrent_output_buffer_disable(DAC0);
```

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

The description of `dac_concurrent_data_set` is shown as below:

**Table 3-127. Function `dac_concurrent_data_set`**

<b>Function name</b>	<code>dac_concurrent_data_set</code>
<b>Function prototype</b>	<code>void dac_concurrent_data_set(uint32_t dac_periph, uint32_t dac_align, uint16_t data0, uint16_t data1);</code>
<b>Function descriptions</b>	set DAC concurrent mode data holding register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b>dac_align</b>	DAC data alignment mode
<i>DAC_ALIGN_12B_R</i>	12-bit right-aligned data
<i>DAC_ALIGN_12B_L</i>	12-bit left-aligned data
<i>DAC_ALIGN_8B_R</i>	8-bit right-aligned data
<b>Input parameter{in}</b>	
<b>data0</b>	DACx_OUT0 data to be loaded (0~4095)
<b>Input parameter{in}</b>	
<b>data1</b>	DACx_OUT1 data to be loaded (0~4095)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

## 3.7. DBG

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

### 3.7.1. Descriptions of Peripheral registers

DBG registers are listed in the table shown as below:

**Table 3-128. DBG Registers**

Registers	Descriptions
DBG_ID	DBG ID code register
DBG_CTL0	DBG control register0
DBG_CTL1	DBG control register1

### 3.7.2. Descriptions of Peripheral functions

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

**Table 3-129. DBG firmware function**

Function name	Function description
dbg_deinit	reset DBG register
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

#### Enum dbg\_periph\_enum

**Table 3-130. 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_TIMER2_HOLD	hold TIMER2 counter when core is halted
DBG_TIMER5_HOLD	hold TIMER5 counter when core is halted
DBG_TIMER13_HOLD	hold TIMER13 counter when core is halted
DBG_TIMER14_HOLD	hold TIMER14 counter when core is halted
DBG_TIMER15_HOLD	hold TIMER15 counter when core is halted

DBG_TIMER16_HOLD	hold TIMER16 counter when core is halted
DBG_I2C0_HOLD	hold I2C0 smbus when core is halted
DBG_I2C1_HOLD	hold I2C1 smbus when core is halted
DBG_RTC_HOLD	hold RTC counter when core is halted

## dbg\_deinit

The description of dbg\_deinit is shown as below:

**Table 3-131. 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 3-132. 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 3-133. 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
DBG_LOW_POWER_SLEEP	keep debugger connection during sleep mode
DBG_LOW_POWER_DEEPSLEEP	keep debugger connection during deepsleep mode
DBG_LOW_POWER_STANDBY	keep debugger connection during standby mode
<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 3-134. Function dbg\_low\_power\_disable**

<b>Function name</b>	dbg_low_power_disable
<b>Function prototype</b>	void dbg_low_power_disable(uint32_t dbg_low_power);
<b>Function descriptions</b>	Disable low power behavior when the mcu is in debug mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dbg_low_power</b>	low power mode

<i>DBG_LOW_POWER_SLEEP</i>	keep debugger connection during sleep mode
<i>DBG_LOW_POWER_DEEPSLEEP</i>	keep debugger connection during deepsleep mode
<i>DBG_LOW_POWER_STANDBY</i>	keep debugger connection during standby mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

### dbg\_periph\_enable

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

**Table 3-135. Function dbg\_periph\_enable**

<b>Function name</b>	dbg_periph_enable
<b>Function prototype</b>	void dbg_periph_enable(dbg_periph_enum dbg_periph);
<b>Function descriptions</b>	Enable peripheral behavior when the mcu is in debug mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dbg_periph</b>	Peripheral refer to <a href="#">Table 3-130. 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_TIMERx_HOLD</i>	x=0,2,5,13,14,15,16, hold TIMEx counter when core is halted
<i>DBG_I2Cx_HOLD</i>	x=0,1, hold I2Cx smbus when core is halted
<i>DBG_RTC_HOLD</i>	hold RTC 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 3-136. Function dbg\_periph\_disable**

<b>Function name</b>	dbg_periph_disable
<b>Function prototype</b>	void dbg_periph_disable(dbg_periph_enum dbg_periph);
<b>Function descriptions</b>	Disable peripheral behavior when the mcu is in debug mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dbg_periph</b>	peripheral refer to <a href="#">Table 3-130. 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_TIMERx_HOLD</i>	x=0,2,5,13,14,15,16, hold TIMERx counter when core is halted
<i>DBG_I2Cx_HOLD</i>	x=0,1, hold I2Cx smbus when core is halted
<i>DBG_RTC_HOLD</i>	hold RTC 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);
```

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

### 3.8.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

**Table 3-137. DMA Registers**

Registers	Descriptions
DMA_INTF	Interrupt flag register
DMA_INTC	Interrupt flag clear register
DMA_CHxCTL (x=0..6)	Channel x control register

Registers	Descriptions
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

### 3.8.2. Descriptions of Peripheral functions

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

**Table 3-138. DMA firmware function**

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



## Structure dma\_parameter\_struct

**Table 3-139. 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 3-140. Function dma\_deinit**

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

Example:

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

## dma\_struct\_para\_init

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

Table 3-141. Function dma\_struct\_para\_init

Function name	dma_struct_para_init
Function prototype	void dma_struct_para_init(dma_parameter_struct* init_struct);
Function descriptions	initialize the parameters of DMA struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
init_struct	the initialization data needed to initialize DMA channel
Output parameter{out}	
-	-
Return value	
-	-

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 3-142. Function dma\_init

Function name	dma_init
Function prototype	void dma_init(uint32_t dma_periph, dma_channel_enum channelx, dma_parameter_struct* init_struct);
Function descriptions	initialize DMA channel
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
DMAx(x=0, 1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..6)	DMA channel selection, DMA0: DMA_CHx(x=0..6), DMA1: DMA_CHx(x=0..4)
Input parameter{in}	
init_struct	Structure for initialization, the structure members can refer to <a href="#">Table 3-139. Structure dma_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DMA0 channel0 initialize */
dma_parameter_struct 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 3-143. 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(x=0..6)</i>	DMA channel selection, DMA0: DMA_CHx(x=0..6), DMA1: DMA_CHx(x=0..4)
<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 3-144. Function dma\_circulation\_disable**

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

Example:

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

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

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

**Table 3-145. Function dma\_memory\_to\_memory\_enable**

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

	DMA_CHx(x=0..4)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

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

**Table 3-146. Function dma\_memory\_to\_memory\_disable**

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

Example:

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

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

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

**Table 3-147. Function dma\_channel\_enable**

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

	channelx);
<b>Function descriptions</b>	enable DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..6)</i>	DMA channel selection, DMA0: DMA_CHx(x=0..6), DMA1: DMA_CHx(x=0..4)
<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 3-148. Function dma\_channel\_disable**

<b>Function name</b>	dma_channel_disable
<b>Function prototype</b>	void dma_channel_disable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	disable DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..6)</i>	DMA channel selection, DMA0: DMA_CHx(x=0..6), DMA1: DMA_CHx(x=0..4)
<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 3-149. 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(x=0..6)</i>	DMA channel selection, DMA0: DMA_CHx(x=0..6), DMA1: DMA_CHx(x=0..4)
<b>Input parameter{in}</b>	
<b>address</b>	peripheral base address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA0 channel0 periph address */
```

```
#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 3-150. 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>	-

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

Example:

```
/* configure DMA0 channel0 memory address */
```

```
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 3-151. 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>	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..6)</i>	DMA channel selection, DMA0: DMA_CHx(x=0..6), DMA1: DMA_CHx(x=0..4)
Input parameter{in}	
<b>number</b>	data transfer number(0x0-0xFFFF)
Output parameter{out}	
-	-



Return value	
-	-

Example:

```
/* configure DMA0 channel0 transfer number */

#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 3-152. Function dma\_transfer\_number\_get**

Function name	dma_transfer_number_get
Function prototype	uint32_t dma_transfer_number_get(uint32_t dma_periph, dma_channel_enum channelx);
Function descriptions	get the number of remaining data to be transferred by the DMA
Precondition	-
The called functions	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..6)</i>	DMA channel selection, DMA0: DMA_CHx(x=0..6), DMA1: DMA_CHx(x=0..4)
Output parameter{out}	
-	-
Return value	
<b>uint32_t</b>	DMA data transmission remaining quantity (0x0-0xFFFF)

Example:

```
/* get DMA0 channel0 transfer number */

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 3-153. Function dma\_priority\_config**

Function name	dma_priority_config
Function prototype	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(x=0..6)</i>	DMA channel selection, DMA0: DMA_CHx(x=0..6), DMA1: DMA_CHx(x=0..4)
<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:

```
/* configure DMA0 channel0 priority */
```

```
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 3-154. 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(x=0..6)</i>	DMA channel selection, DMA0: DMA_CHx(x=0..6), DMA1: DMA_CHx(x=0..4)
<b>Input parameter{in}</b>	
<b>mwidth</b>	transfer data width of memory
<i>DMA_MEMORY_WIDT H_8BIT</i>	transfer data width of memory is 8-bit
<i>DMA_MEMORY_WIDT H_16BIT</i>	transfer data width of memory is 16-bit
<i>DMA_MEMORY_WIDT H_32BIT</i>	transfer data width of memory is 32-bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA0 channel0 memory width */
```

```
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 3-155. 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 width of peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..6)</i>	DMA channel selection, DMA0: DMA_CHx(x=0..6), DMA1: DMA_CHx(x=0..4)
<b>Input parameter{in}</b>	
<b>pwidth</b>	transfer data width of peripheral
<i>DMA_PERIPHERAL_W IDTH_8BIT</i>	transfer data width of peripheral is 8-bit
<i>DMA_PERIPHERAL_W</i>	transfer data width of peripheral is 16-bit

<i>IDTH_16BIT</i>	
<i>DMA_PERIPHERAL_WIDTH_32BIT</i>	transfer data width of peripheral is 32-bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA0 channel0 periph width */
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 3-156. 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(x=0..6)</i>	DMA channel selection, DMA0: DMA_CHx(x=0..6), DMA1: DMA_CHx(x=0..4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMA0 channel0 memory increase */
dma_memory_increase_enable(DMA0, DMA_CH0);
```

### **dma\_memory\_increase\_disable**

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

Table 3-157. Function dma\_memory\_increase\_disable

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

Example:

```
/* disable DMA0 channel0 memory increase */
```

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

### dma\_periph\_increase\_enable

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

Table 3-158. Function dma\_periph\_increase\_enable

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

Return value	
-	-

Example:

```
/* enable DMA0 channel0 periph increase */
```

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

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

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

**Table 3-159. Function dma\_periph\_increase\_disable**

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

Example:

```
/* disable DMA0 channel0 periph increase */
```

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

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

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

**Table 3-160. Function dma\_transfer\_direction\_config**

Function name	dma_transfer_direction_config
Function prototype	void dma_transfer_direction_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t direction);
Function descriptions	configure the direction of data transfer on the channel

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

```
/* configure DMA0 channel0 transfer direction */
```

```
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 3-161. 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(x=0..6)</i>	DMA channel selection, DMA0: DMA_CHx(x=0..6), DMA1: DMA_CHx(x=0..4)
<b>Input parameter{in}</b>	

<b>flag</b>	specify get which flag
<i>DMA_FLAG_G</i>	global interrupt flag of channel
<i>DMA_FLAG_FTF</i>	full transfer finish flag of channel
<i>DMA_FLAG_HTF</i>	half transfer finish flag of channel
<i>DMA_FLAG_ERR</i>	error flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get DMA0 channel0 flag */
```

```
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 3-162. Function dma\_flag\_clear**

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

```
/* clear DMA0 channel0 flag */
```

```
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 3-163. Function dma\_interrupt\_flag\_get**

<b>Function name</b>	dma_interrupt_flag_get
<b>Function prototype</b>	FlagStatus dma_interrupt_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
<b>Function descriptions</b>	check DMA flag and interrupt enable bit is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
DMAx(x=0,1)	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
DMA_CHx(x=0..6)	DMA channel selection, DMA0: DMA_CHx(x=0..6), DMA1: DMA_CHx(x=0..4)
<b>Input parameter{in}</b>	
<b>flag</b>	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_ERR	error interrupt flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get DMA interrupt_flag */
```

```
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 3-164. Function dma\_interrupt\_flag\_clear

Function name	dma_interrupt_flag_clear
Function prototype	void dma_interrupt_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
Function descriptions	clear DMA a channel flag
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
DMAx(x=0,1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..6)	DMA channel selection, DMA0: DMA_CHx(x=0..6), DMA1: DMA_CHx(x=0..4)
Input parameter{in}	
flag	specify get which flag
DMA_INT_FLAG_G	global interrupt flag of channel
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_ERR	error interrupt flag of channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* get DMA interrupt_flag */
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 3-165. Function dma\_interrupt\_enable

Function name	dma_interrupt_enable
Function prototype	void dma_interrupt_enable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
Function descriptions	enable DMA interrupt
Precondition	-
The called functions	-
Input parameter{in}	

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

```
/* enable DMA0 channel0 interrupt */
```

```
dma_interrupt_enable(DMA0, DMA_CH0, DMA_INT_FTF);
```

### **dma\_interrupt\_disable**

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

**Table 3-166. Function dma\_interrupt\_disable**

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

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
dma_interrupt_disable(DMA0, DMA_CH0, DMA_INT_FTF);
```

## 3.9. ECLIC

RISC-V integrates the Enhancement Core-Local Interrupt Controller (ECLIC) for efficient interrupts processing. ECLIC is designed to provide low-latency, vectored, pre-emptive interrupts for RISC-V systems. The ECLIC firmware functions are introduced in chapter [3.9.1](#).

### 3.9.1. Descriptions of Peripheral functions

#### Enum IRQn\_Type

Table3-167. Enum IRQn\_Type

Member name	Function description
WWDGT_IRQn	WWDGT interrupt
LVD_IRQn	LVD from EXTI interrupt
TAMPER_IRQn	Tamper interrupt
RTC_IRQn	RTC global interrupt
FMC_IRQn	FMC global interrupt
RCU_CTC_IRQn	RCU global interrupt
EXTI0_IRQn	EXTI line0 interrupt
EXTI1_IRQn	EXTI line1 interrupt
EXTI2_IRQn	EXTI line2 interrupt
EXTI3_IRQn	EXTI line3 interrupt
EXTI4_IRQn	EXTI line4 interrupt
DMA0_Channel0_IRQn	DMA0 channel0 global interrupt
DMA0_Channel1_IRQn	DMA0 channel1 global interrupt
DMA0_Channel2_IRQn	DMA0 channel2 global interrupt
DMA0_Channel3_IRQn	DMA0 channel3 global interrupt
DMA0_Channel4_IRQn	DMA0 channel4 global interrupt
DMA0_Channel5_IRQn	DMA0 channel5 global interrupt
DMA0_Channel6_IRQn	DMA0 channel6 global interrupt
ADC0_1_IRQn	ADC0 and ADC1 global interrupts
CAN0_TX_IRQn	CAN0 TX interrupt

Member name	Function description
CAN0_RX0_IRQn	CAN0 RX0 interrupt
CAN0_RX1_IRQn	CAN0 RX1 interrupt
CAN0_EWMC_IRQn	CAN0 EWMC interrupt
EXTI5_9_IRQn	EXTI line[9:5] interrupt
TIMER0_BRK_IRQn	TIMER0 break interrupt
TIMER0_UP_IRQn	TIMER0 update interrupt
TIMER0_TRG_CMT_IRQn	TIMER0 trigger and channel commutation interrupts
TIMER0_Channel_IRQn	TIMER0 channel capture compare interrupt
TIMER1_IRQn	TIMER1 global interrupt
TIMER2_IRQn	TIMER2 global interrupt
TIMER3_IRQn	TIMER3 global interrupt
I2C0_EV_IRQn	I2C0 event interrupt
I2C0_ER_IRQn	I2C0 error interrupt
I2C1_EV_IRQn	I2C1 event interrupt
I2C1_ER_IRQn	I2C1 error interrupt
SPI0_IRQn	SPI0 global interrupt
SPI1_IRQn	SPI1 global interrupt
USART0_IRQn	USART0 global interrupt
USART1_IRQn	USART1 global interrupt
USART2_IRQn	USART2 global interrupt
EXTI10_15_IRQn	EXTI line[15:10] interrupts
RTC_Alarm_IRQn	RTC alarm from EXTI interrupt
USBFS_WKUP_IRQn	USBFS wakeup from EXTI interrupt
TIMER4_IRQn	TIMER4 global interrupt
SPI2_IRQn	SPI2 global interrupt
UART3_IRQn	UART3 global interrupt
UART4_IRQn	UART4 global interrupt
TIMER5_IRQn	TIMER5 global interrupt
TIMER6_IRQn	TIMER6 global interrupt
DMA1_Channel0_IRQn	DMA1 channel0 global interrupt
DMA1_Channel1_IRQn	DMA1 channel1 global interrupt
DMA1_Channel2_IRQn	DMA1 channel2 global interrupt
DMA1_Channel3_IRQn	DMA1 channel3 global interrupt
DMA1_Channel4_IRQn	DMA1 channel4 global interrupt
CAN1_TX_IRQn	CAN1 TX interrupt
CAN1_RX0_IRQn	CAN1 RX0 interrupt
CAN1_RX1_IRQn	CAN1 RX1 interrupt
CAN1_EWMC_IRQn	CAN1 EWMC interrupt
USBFS_IRQn	USBFS global interrupt

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

Table 3-168. ECLIC firmware function

Function name	Function description
eclic_global_interrupt_enable	enable the global interrupt
eclic_global_interrupt_disable	disable the global interrupt
eclic_priority_group_set	set the priority group
eclic_irq_enable	enable the interrupt request
eclic_irq_disable	disable the interrupt request
eclic_system_reset	reset system
eclic_send_event	send event(SEV)

### eclic\_global\_interrupt\_enable

The description of eclic\_global\_interrupt\_enable is shown as below:

Table 3-169. Function eclic\_global\_interrupt\_enable

Function name	eclic_global_interrupt_enable
Function prototype	void eclic_global_interrupt_enable(void);
Function descriptions	enable the global interrupt
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the global interrupt */
eclic_global_interrupt_enable();
```

### eclic\_global\_interrupt\_disable

The description of eclic\_global\_interrupt\_disable is shown as below:

Table 3-170. Function eclic\_global\_interrupt\_disable

Function name	eclic_global_interrupt_disable
Function prototype	void eclic_global_interrupt_disable(void);
Function descriptions	disable the global interrupt
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* disable the global interrupt */
eclic_global_interrupt_disable();
```

### eclic\_priority\_group\_set

The description of eclic\_priority\_group\_set is shown as below:

**Table 3-171. Function eclic\_priority\_group\_set**

Function name	eclic_priority_group_set
Function prototype	void eclic_priority_group_set(uint32_t prigroup);
Function descriptions	set the priority group
Precondition	-
The called functions	-
Input parameter{in}	
prigroup	specify the priority group
ECLIC_PRIGROUP_LEV EL0_Prio4	0 bits for level 4 bits for priority
ECLIC_PRIGROUP_LEV EL1_Prio3	1 bits for level 3 bits for priority
ECLIC_PRIGROUP_LEV EL2_Prio2	2 bits for level 2 bits for priority
ECLIC_PRIGROUP_LEV EL3_Prio1	3 bits for level 1 bits for priority
ECLIC_PRIGROUP_LEV EL4_Prio0	4 bits for level 0 bits for priority
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the priority group */
eclic_priority_group_set(ECLIC_PRIGROUP_LEVEL0_Prio4);
```

### eclic\_irq\_enable

The description of eclic\_irq\_enable is shown as below:

**Table 3-172. Function eclic\_irq\_enable**

Function name	eclic_irq_enable
Function prototype	void eclic_irq_enable(uint32_t source, uint8_t level, uint8_t priority);

<b>Function descriptions</b>	enable ECLIC interrupt request
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	interrupt request, detailed in <a href="#">Enum IRQn_Type</a>
<b>Input parameter{in}</b>	
<b>level</b>	the level needed to set (maximum is 15, refer to the priority group)
<b>Input parameter{in}</b>	
<b>priority</b>	the priority needed to set (maximum is 15, refer to the priority group)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable and set key EXTI interrupt to the specified priority */
eclic_global_interrupt_enable();
eclic_priority_group_set(ECLIC_PRIGROUP_LEVEL3_PRIO1);
eclic_irq_enable(EXTI10_15_IRQn, 1, 1);
```

### eclic\_irq\_disable

The description of eclic\_irq\_disable is shown as below:

**Table 3-173. Function eclic\_irq\_disable**

<b>Function name</b>	eclic_irq_disable
<b>Function prototype</b>	void eclic_irq_disable(uint32_t source);
<b>Function descriptions</b>	disable ECLIC interrupt request
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	interrupt request, detailed in <a href="#">Enum IRQn_Type</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the interrupt request */
eclic_irq_disable(EXTI10_15_IRQn);
```

### eclic\_system\_reset

The description of eclic\_system\_reset is shown as below:



Table 3-174. Function `eclic_system_reset`

Function name	<code>eclic_system_reset</code>
Function prototype	<code>void eclic_system_reset(void);</code>
Function descriptions	reset system
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset system */
eclic_system_reset();
```

### `eclic_send_event`

The description of `eclic_send_event` is shown as below:

Table 3-175. Function `eclic_send_event`

Function name	<code>eclic_send_event</code>
Function prototype	<code>void eclic_send_event(void);</code>
Function descriptions	send event(SEV)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* send event(SEV) */
eclic_send_event();
```

## 3.10. EXMC

The external memory controller EXMC, is used as a translator for MCU to access a variety of external memory. The EXMC registers are listed in chapter [3.10.1](#), the EXMC firmware functions are introduced in chapter [3.10.2](#).

### 3.10.1. Descriptions of Peripheral registers

EXMC registers are listed in the table shown as below:

**Table 3-176. EXMC Registers**

Registers	Descriptions
EXMC_SNCTLx (x=0)	SRAM/NOR Flash control registers
EXMC_SNTCFGx (x=0)	SRAM/NOR Flash timing configuration registers

### 3.10.2. Descriptions of Peripheral functions

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

**Table 3-177. EXMC firmware function**

Function name	Function description
exmc_norsram_deinit	deinitialize EXMC NOR/SRAM region
exmc_norsram_struct_para_init	initialize the struct exmc_norsram_parameter_struct
exmc_norsram_init	initialize EXMC NOR/SRAM region
exmc_norsram_enable	enable EXMC NOR/PSRAM bank region
exmc_norsram_disable	disable EXMC NOR/PSRAM bank region

#### Structure exmc\_norsram\_timing\_parameter\_struct

**Table 3-178. Structure exmc\_norsram\_timing\_parameter\_struct**

Member name	Function description
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 3-179. Structure exmc\_norsram\_parameter\_struct**

Member name	Function description
norsram_region	select the region of EXMC NOR/SRAM bank
asyn_wait	enable or disable the asynchronous wait function
nwait_signal	enable or disable the NWAIT signal while in synchronous bust mode
memory_write	enable or disable the write operation
nwait_polarity	specifies the polarity of NWAIT signal from memory
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

## exmc\_norsram\_deinit

The description of exmc\_norsram\_deinit is shown as below:

**Table 3-180. Function exmc\_norsram\_deinit**

<b>Function name</b>	exmc_norsram_deinit
<b>Function prototype</b>	void exmc_norsram_deinit(uint32_t norsram_region);
<b>Function descriptions</b>	deinitialize EXMC NOR/SRAM region
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>norsram_region</b>	select the region of bank0
EXMC_BANK0_NORSRAM_REGIONx(x=0)	region x of bank0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* deinitialize EXMC NOR/SRAM region 0 */
exmc_norsram_deinit(EXMC_BANK0_NORSRAM_REGION0);
```

## exmc\_norsram\_struct\_para\_init

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

**Table 3-181. Function exmc\_norsram\_struct\_para\_init**

<b>Function name</b>	exmc_norsram_struct_para_init
<b>Function prototype</b>	void exmc_norsram_struct_para_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);
<b>Function descriptions</b>	initialize the struct exmc_norsram_parameter_struct
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_norsram_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-179. Structure exmc_norsram_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the struct nor_init_struct */
```

```
exmc_norsram_parameter_struct nor_init_struct;
exmc_norsram_struct_para_init(&nor_init_struct);
```

## exmc\_norsram\_init

The description of exmc\_norsram\_init is shown as below:

**Table 3-182. Function exmc\_norsram\_init**

<b>Function name</b>	exmc_norsram_init
<b>Function prototype</b>	void exmc_norsram_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);
<b>Function descriptions</b>	initialize EXMC NOR/SRAM region
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_norsram_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-179. Structure exmc_norsram_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize EXMC NOR/SRAM region */
exmc_norsram_parameter_struct lcd_init_struct;
exmc_norsram_timing_parameter_struct lcd_timing_init_struct;
lcd_timing_init_struct.bus_latency = 2;
lcd_timing_init_struct.asyn_data_setuptime = 10;
lcd_timing_init_struct.asyn_address_holdtime = 2;
lcd_timing_init_struct.asyn_address_setuptime = 5;

lcd_init_struct.norsram_region = EXMC_BANK0_NORSRAM_REGION0;
lcd_init_struct.asyn_wait = DISABLE;
lcd_init_struct.nwait_signal = DISABLE;
lcd_init_struct.memory_write = ENABLE;
lcd_init_struct.nwait_polarity = EXMC_NWAIT_POLARITY_LOW;
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 = ENABLE;
lcd_init_struct.read_write_timing = &lcd_timing_init_struct;

exmc_norsram_init(&lcd_init_struct);
```

## exmc\_norsram\_enable

The description of exmc\_norsram\_enable is shown as below:

**Table 3-183. Function exmc\_norsram\_enable**

<b>Function name</b>	exmc_norsram_enable
<b>Function prototype</b>	void exmc_norsram_enable(uint32_t norsram_region);
<b>Function descriptions</b>	enable EXMC NOR/PSRAM bank region
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>norsram_region</b>	specifie the region of NOR/PSRAM bank
EXMC_BANK0_NORS RAM_REGIONx(x=0)	region x of bank0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable region 0 of bank0 */
exmc_norsram_enable(EXMC_BANK0_NORSRAM_REGION0);
```

## exmc\_norsram\_disable

The description of exmc\_norsram\_disable is shown as below:

**Table 3-184. Function exmc\_norsram\_disable**

<b>Function name</b>	exmc_norsram_disable
<b>Function prototype</b>	void exmc_norsram_disable(uint32_t norsram_region);
<b>Function descriptions</b>	disable EXMC NOR/PSRAM bank region
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>norsram_region</b>	specifie the region of NOR/PSRAM bank
EXMC_BANK0_NORS RAM_REGIONx(x=0)	region x of bank0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable region 0 of bank0 */
```

```
exmc_norsram_disable(EXMC_BANK0_NORSRAM_REGION0);
```

## 3.11. EXTI

EXTI is the interrupt/event controller in the MCU. It contains up to 19 independent edge detectors and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter [3.11.1](#), the EXTI firmware functions are introduced in chapter [3.11.2](#).

### 3.11.1. Descriptions of Peripheral registers

EXTI registers are listed in the table shown as below:

**Table 3-185. EXTI Registers**

Registers	Descriptions
EXTI_INTEN	interrupt enable register
EXTI_EVEN	event enable register
EXTI_RTEN	rising edge trigger enable register
EXTI_FTEN	falling edge trigger enable register
EXTI_SWIEV	software interrupt event register
EXTI_PD	pending register

### 3.11.2. Descriptions of Peripheral functions

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

**Table 3-186. EXTI firmware function**

Function name	Function description
exti_deinit	deinitialize the EXTI
exti_init	initialize the EXTI line x
exti_interrupt_enable	enable the interrupts from EXTI line x
exti_event_enable	enable the events from EXTI line x
exti_interrupt_disable	disable EXTI line x interrupt
exti_event_disable	disable EXTI line x event
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_software_interrupt_enable	enable EXTI line x software interrupt
exti_software_interrupt_disable	disable EXTI line x software interrupt

## Enum exti\_line\_enum

**Table 3-187. Enum exti\_line\_enum**

Member name	Function description
EXTI_0	EXTI line 0
EXTI_1	EXTI line 1
EXTI_2	EXTI line 2
EXTI_3	EXTI line 3
EXTI_4	EXTI line 4
EXTI_5	EXTI line 5
EXTI_6	EXTI line 6
EXTI_7	EXTI line 7
EXTI_8	EXTI line 8
EXTI_9	EXTI line 9
EXTI_10	EXTI line 10
EXTI_11	EXTI line 11
EXTI_12	EXTI line 12
EXTI_13	EXTI line 13
EXTI_14	EXTI line 14
EXTI_15	EXTI line 15
EXTI_16	EXTI line 16
EXTI_17	EXTI line 17
EXTI_18	EXTI line 18

## Enum exti\_mode\_enum

**Table 3-188. Enum exti\_mode\_enum**

Member name	Function description
EXTI_INTERRUPT	EXTI interrupt mode
EXTI_EVENT	EXTI event mode

## Enum exti\_trig\_type\_enum

**Table 3-189. Enum exti\_trig\_type\_enum**

Member name	Function description
EXTI_TRIG_RISING	EXTI rising edge trigger
EXTI_TRIG_FALLING	EXTI falling edge trigger
EXTI_TRIG_BOTH	EXTI rising and falling edge trigger
EXTI_TRIG_NONE	without rising edge or falling edge trigger

## exti\_deinit

The description of exti\_deinit is shown as below:

Table 3-190. Function exti\_deinit

Function name	exti_deinit
Function prototype	void exti_deinit(void);
Function descriptions	deinitialize the EXTI
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize the EXTI */
exti_deinit();
```

### exti\_init

The description of exti\_init is shown as below:

Table 3-191. Function exti\_init

Function name	exti_init
Function prototype	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
Function descriptions	initialize the EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-187. Enum exti_line_enum</a>
Input parameter{in}	
mode	EXTI mode <a href="#">Table 3-188. Enum exti_mode_enum</a>
Input parameter{in}	
trig_type	trigger type <a href="#">Table 3-189. Enum exti_trig_type_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure EXTI_0 */
exti_init(EXTI_0, EXTI_INTERRUPT, EXTI_TRIG_BOTH);
```



## exti\_interrupt\_enable

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

**Table 3-192. Function exti\_interrupt\_enable**

<b>Function name</b>	exti_interrupt_enable
<b>Function prototype</b>	void exti_interrupt_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable the interrupts from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-187. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the interrupts from EXTI line 0 */
exti_interrupt_enable(EXTI_0);
```

## exti\_interrupt\_disable

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

**Table 3-193. Function exti\_interrupt\_disable**

<b>Function name</b>	exti_interrupt_disable
<b>Function prototype</b>	void exti_interrupt_disable(exti_line_enum linex);
<b>Function descriptions</b>	disable the interrupts from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-187. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the interrupts from EXTI line 0 */
exti_interrupt_disable(EXTI_0);
```

## exti\_event\_enable

The description of exti\_event\_enable is shown as below:

**Table 3-194. Function exti\_event\_enable**

<b>Function name</b>	exti_event_enable
<b>Function prototype</b>	void exti_event_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable the events from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-187. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the events from EXTI line 0 */
exti_event_enable(EXTI_0);
```

## exti\_event\_disable

The description of exti\_event\_disable is shown as below:

**Table 3-195. Function exti\_event\_disable**

<b>Function name</b>	exti_event_disable
<b>Function prototype</b>	void exti_event_disable(exti_line_enum linex);
<b>Function descriptions</b>	disable the events from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-187. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the events from EXTI line 0 */
exti_event_disable(EXTI_0);
```

## exti\_software\_interrupt\_enable

The description of exti\_software\_interrupt\_enable is shown as below:

Table 3-196. Function exti\_software\_interrupt\_enable

Function name	exti_software_interrupt_enable
Function prototype	void exti_software_interrupt_enable(exti_line_enum linex);
Function descriptions	enable the software interrupt event from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-187. Enum exti_line_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable EXTI line 0 software interrupt */
exti_software_interrupt_enable(EXTI_0);
```

### exti\_software\_interrupt\_disable

The description of exti\_software\_interrupt\_disable is shown as below:

Table 3-197. Function exti\_software\_interrupt\_disable

Function name	exti_software_interrupt_disable
Function prototype	void exti_software_interrupt_disable(exti_line_enum linex);
Function descriptions	disable the software interrupt event from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-187. Enum exti_line_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable EXTI line 0 software interrupt */
exti_software_interrupt_disable(EXTI_0);
```

### exti\_flag\_get

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

Table 3-198. Function exti\_flag\_get

Function name	exti_flag_get
---------------	---------------

<b>Function prototype</b>	FlagStatus exti_flag_get(exti_line_enum linex);
<b>Function descriptions</b>	get EXTI line x interrupt pending flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-187. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get EXTI line 0 flag status */
FlagStatus state = exti_flag_get(EXTI_0);
```

## exti\_flag\_clear

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

**Table 3-199. Function exti\_flag\_clear**

<b>Function name</b>	exti_flag_clear
<b>Function prototype</b>	void exti_flag_clear(exti_line_enum linex);
<b>Function descriptions</b>	clear EXTI line x interrupt pending flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-187. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear EXTI line 0 flag status */
exti_flag_clear(EXTI_0);
```

## exti\_interrupt\_flag\_get

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

**Table 3-200. Function exti\_interrupt\_flag\_get**

<b>Function name</b>	exti_interrupt_flag_get
<b>Function prototype</b>	FlagStatus exti_interrupt_flag_get(exti_line_enum linex);
<b>Function descriptions</b>	get EXTI line x interrupt pending flag

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-187. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get EXTI line 0 interrupt flag status */
FlagStatus state = exti_interrupt_flag_get(EXTI_0);
```

### exti\_interrupt\_flag\_clear

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

**Table 3-201. Function exti\_interrupt\_flag\_clear**

<b>Function name</b>	exti_interrupt_flag_clear
<b>Function prototype</b>	void exti_interrupt_flag_clear(exti_line_enum linex);
<b>Function descriptions</b>	clear EXTI line x interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-187. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear EXTI line 0 interrupt flag status */
exti_interrupt_flag_clear(EXTI_0);
```

## 3.12. FMC

There is flash controller and option byte for GD32VF103 series. The FMC registers are listed in chapter [3.12.1](#). The FMC firmware functions are introduced in chapter [3.12.2](#).

### 3.12.1. Descriptions of Peripheral registers

FMC registers are listed in the table shown as below:

**Table 3-202. FMC Registers**

Registers	Descriptions
FMC_WS	FMC wait state register
FMC_KEY	FMC unlock key register
FMC_OBKEY	FMC option bytes unlock key register
FMC_STAT	FMC status register
FMC_CTL	FMC control register
FMC_ADDR	FMC address register
FMC_OBSTAT	FMC option bytes status register
FMC_WP	FMC write protection register
FMC_PID	FMC product ID register

### 3.12.2. Descriptions of Peripheral functions

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

**Table 3-203. FMC firmware function**

Function name	Function description
fmc_wscont_set	set the FMC wait state counter
fmc_unlock	unlock the main FMC operation
fmc_lock	lock the main FMC operation
fmc_page_erase	FMC erase page
fmc_mass_erase	FMC erase whole chip
fmc_word_program	FMC program a word at the corresponding address
fmc_halfword_program	FMC program a half word at the corresponding address
ob_unlock	unlock the option byte operation
ob_lock	lock the option byte operation
ob_erase	erase the FMC option byte
ob_write_protection_enable	enable write protection
ob_security_protection_config	configure security protection
ob_user_write	program the FMC user option byte
ob_data_program	program the FMC data option byte
ob_user_get	get OB_USER in register FMC_OBSTAT
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 FMC option byte security protection state
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
fmc_interrupt_flag_get	get FMC interrupt flag state
fmc_interrupt_flag_clear	clear FMC interrupt flag state
fmc_state_get	return the FMC state

Function name	Function description
fmc_ready_wait	check FMC ready or not

### Enum fmc\_state\_enum

**Table 3-204. Enum fmc\_state\_enum**

enum name	enum description
FMC_READY	the operation has been completed
FMC_BUSY	the operation is in progress
FMC_PGERR	program error
FMC_WPERR	erase/program protection error
FMC_TOERR	timeout error

### Enum fmc\_int\_enum

**Table 3-205. Enum fmc\_int\_enum**

enum name	enum description
FMC_INT_END	enable FMC end of program interrupt
FMC_INT_ERR	enable FMC error interrupt

### Enum fmc\_flag\_enum

**Table 3-206. Enum fmc\_flag\_enum**

enum name	enum description
FMC_FLAG_BUSY	FMC busy flag
FMC_FLAG_PGER R	FMC operation error flag bit
FMC_FLAG_WPER R	FMC erase/program protection error flag bit
FMC_FLAG_END	FMC end of operation flag bit
FMC_FLAG_OBER R	FMC option bytes read error flag

### Enum fmc\_interrupt\_flag\_enum

**Table 3-207. Enum fmc\_interrupt\_flag\_enum**

enum name	enum description
FMC_INT_FLAG_P GERR	FMC operation error interrupt flag bit
FMC_INT_FLAG_W PERR	FMC erase/program protection error interrupt flag bit
FMC_INT_FLAG_E ND	FMC end of operation interrupt flag bit

## fmc\_wscnt\_set

The description of fmc\_wscnt\_set is shown as below:

**Table 3-208. Function fmc\_wscnt\_set**

<b>Function name</b>	fmc_wscnt_set
<b>Function prototype</b>	void fmc_wscnt_set(uint32_t wscnt);
<b>Function descriptions</b>	set the wait state counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wscnt</b>	wait state counter value
WS_WSCNT_0	FMC 0 wait state
WS_WSCNT_1	FMC 1 wait state
WS_WSCNT_2	FMC 2 wait state
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the wait state counter value */
fmc_wscnt_set (WS_WSCNT_1);
```

## fmc\_unlock

The description of fmc\_unlock is shown as below:

**Table 3-209. Function fmc\_unlock**

<b>Function name</b>	fmc_unlock
<b>Function prototype</b>	void fmc_unlock (void);
<b>Function descriptions</b>	unlock the main FMC operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* unlock the main FMC operation */
```



```
fmc_unlock ( );
```

## fmc\_lock

The description of fmc\_lock is shown as below:

**Table 3-210. Function fmc\_lock**

<b>Function name</b>	fmc_lock
<b>Function prototype</b>	void fmc_lock(void);
<b>Function descriptions</b>	lock the main FMC operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* lock the main FMC operation */
```

```
fmc_lock( );
```

## fmc\_page\_erase

The description of fmc\_page\_erase is shown as below:

**Table 3-211. 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>	FMC erase page
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>page_address</b>	the page address to be erased
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-204. Enum fmc_state_enum</a>

Example:

```
/* erase page */
```

```
fmc_state_enum state;
```

```
fmc_unlock( );

state = fmc_page_erase (0x08004000);

fmc_lock( );
```

### fmc\_mass\_erase

The description of fmc\_mass\_erase is shown as below:

**Table 3-212. Function fmc\_mass\_erase**

<b>Function name</b>	fmc_mass_erase
<b>Function prototype</b>	fmc_state_enum fmc_mass_erase(void );
<b>Function descriptions</b>	FMC erase whole chip
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-204. Enum fmc_state_enum</a>

Example:

```
/* erase whole chip */

fmc_state_enum state;

fmc_unlock( );

state = fmc_mass_erase( );

fmc_lock( );
```

### fmc\_word\_program

The description of fmc\_word\_program is shown as below:

**Table 3-213. 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>	FMC program a word at the corresponding address
<b>Precondition</b>	fmc_unlock/fmc_page_erase
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>address</b>	address to program
<b>Input parameter{in}</b>	

<b>data</b>	word to program
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-204. Enum fmc_state_enum</a>

Example:

```

/* program a word at the corresponding address */

fmc_state_enum state;

fmc_unlock( );

fmc_page_erase(0x08004000);

state = fmc_word_program( 0x08004000,0xaabbccdd);

fmc_lock( );

```

### fmc\_halfword\_program

The description of fmc\_halfword\_program is shown as below:

**Table 3-214. Function fmc\_word\_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>	FMC program a half word at the corresponding address
<b>Precondition</b>	fmc_unlock/fmc_page_erase
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>address</b>	address to program
<b>Input parameter{in}</b>	
<b>data</b>	half word to program
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-204. Enum fmc_state_enum</a>

Example:

```

/* program a half word at the corresponding address */

fmc_state_enum state;

fmc_unlock( );

fmc_page_erase(0x08004000);

```

```
state = fmc_halfword_program( 0x08004000,0xaabb);
```

```
fmc_lock( );
```

## ob\_unlock

The description of ob\_unlock is shown as below:

**Table 3-215. 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( );
```

```
fmc_lock( );
```

## ob\_lock

The description of ob\_lock is shown as below:

**Table 3-216. 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_unlock( );
```

```
ob_lock( );
```

```
fmc_lock( );
```

### ob\_erase

The description of ob\_erase is shown as below:

**Table 3-217. Function ob\_erase**

<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>	
fmc_state_enum	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-204. Enum fmc_state_enum</a>

Example:

```
/* erase the option byte */
```

```
fmc_state_enum fmc_state;
```

```
fmc_unlock( );
```

```
ob_unlock( );
```

```
fmc_state = ob_erase ( );
```

```
ob_lock( );
```

```
fmc_lock( );
```

### ob\_write\_protection\_enable

The description of ob\_write\_protection\_enable is shown as below:

Table 3-218. Function `ob_write_protection_enable`

<b>Function name</b>	<code>ob_write_protection_enable</code>
<b>Function prototype</b>	<code>fmc_state_enum ob_write_protection_enable(uint32_t ob_wp);</code>
<b>Function descriptions</b>	enable write protection
<b>Precondition</b>	<code>ob_unlock</code>
<b>The called functions</b>	<code>fmc_ready_wait</code>
<b>Input parameter{in}</b>	
<b>ob_wp</b>	specify sector to be write protected, set the bit to 1 if you want to protect the corresponding pages. meanwhile, sector macro could used to set specific sector write protected.
<code>OB_WP_x</code>	write protect specify sector(x = 0...31)
<code>OB_WP_ALL</code>	write protect all sector
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-204. Enum fmc_state_enum</a>

Example:

```

/* enable write protection */

fmc_state_enum fmc_state;

fmc_unlock( );

ob_unlock( );

fmc_state = ob_write_protection_enable(OB_WP_10);

ob_lock( );

fmc_lock( );

```

## ob\_security\_protection\_config

The description of `ob_security_protection_config` is shown as below:

Table 3-219. Function `ob_security_protection_config`

<b>Function name</b>	<code>ob_security_protection_config</code>
<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>
<b>Input parameter{in}</b>	
<b>ob_spc</b>	specify security protection
<code>FMC_NSPC</code>	no security protection
<code>FMC_USPC</code>	under security protection

Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-204. Enum fmc_state_enum</a>

Example:

```
/* enable security protection */

fmc_state_enum fmc_state;

fmc_unlock( );

ob_unlock( );

fmc_state = ob_security_protection_config(FMC_USPC);

ob_lock( );

fmc_lock( );
```

### ob\_user\_write

The description of ob\_user\_write is shown as below:

**Table 3-220. Function ob\_user\_write**

Function name	ob_user_write
Function prototype	fmc_state_enum ob_user_write(uint8_t ob_fwdgt, uint8_t ob_deepsleep, uint8_t ob_stdby, uint8_t ob_boot);
Function descriptions	program the FMC user option byte
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
ob_fwdgt	option byte watchdog value
OB_FWDGT_SW	software free watchdog
OB_FWDGT_HW	hardware free watchdog
Input parameter{in}	
ob_deepsleep	option byte deepsleep reset value
OB_DEEPSLEEP_NRS T	no reset when entering deepsleep mode
OB_DEEPSLEEP_RST	generate a reset instead of entering deepsleep mode
Input parameter{in}	
ob_stdby	option byte standby reset value
OB_STDBY_NRST	no reset when entering standby mode
OB_STDBY_RST	generate a reset instead of entering standby mode
Input parameter{in}	
ob_boot	specifies the option byte boot bank value

<i>OB_BOOT_B0</i>	boot from bank0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-204. Enum fmc_state_enum</a>

Example:

```

/* program the FMC user option byte */

fmc_state_enum fmc_state;

fmc_unlock( );

ob_unlock( );

fmc_state = ob_user_write(OB_FWDGT_SW, OB_DEEPSLEEP_NRST, OB_STDBY_NRST,
OB_BOOT_B0);

ob_lock( );

fmc_lock( );

```

### ob\_data\_program

The description of ob\_data\_program is shown as below:

**Table 3-221. 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 the FMC data option byte
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>address</b>	the option bytes address to be programmed
<b>Input parameter{in}</b>	
<b>data</b>	the byte to be programmed
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-204. Enum fmc_state_enum</a>

Example:

```

/* program option bytes data */

fmc_state_enum fmc_state;

```



```
fmc_unlock( );

ob_unlock( );

fmc_state = ob_data_program(0x1ffff804, 0x55);

ob_lock( );

fmc_lock( );
```

### ob\_user\_get

The description of ob\_user\_get is shown as below:

**Table 3-222. 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 OB_USER in register FMC_OBSTAT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint8_t	the FMC user option byte values(0x00 – 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 3-223. Function ob\_data\_get**

<b>Function name</b>	ob_data_get
<b>Function prototype</b>	uint16_t ob_data_get(void);
<b>Function descriptions</b>	get OB_DATA in register FMC_OBSTAT
<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 3-224. 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 3-225. 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 FMC option byte security protection state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the FMC option byte security protection level */
```

```
FlagStatus spc_stat = ob_spc_get( );
```

### fmc\_interrupt\_enable

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

**Table 3-226. Function fmc\_interrupt\_enable**

<b>Function name</b>	fmc_interrupt_enable
<b>Function prototype</b>	void fmc_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable FMC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the FMC interrupt source
<i>FMC_INT_END</i>	enable FMC end of program interrupt
<i>FMC_INT_ERR</i>	enable FMC error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable FMC interrupt */
```

```
fmc_interrupt_enable(FMC_INT_END);
```

### fmc\_interrupt\_disable

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

**Table 3-227. Function fmc\_interrupt\_disable**

<b>Function name</b>	fmc_interrupt_disable
<b>Function prototype</b>	void fmc_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable FMC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the FMC interrupt source
<i>FMC_INT_END</i>	FMC end of program interrupt
<i>FMC_INT_ERR</i>	FMC error interrupt
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* disable FMC interrupt */
```

```
fmc_interrupt_disable(FMC_INT_END);
```

### fmc\_flag\_get

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

**Table 3-228. Function fmc\_flag\_get**

Function name	fmc_flag_get
Function prototype	FlagStatus fmc_flag_get(uint32_t flag);
Function descriptions	check FMC flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	check FMC flag
FMC_FLAG_BUSY	FMC busy flag bit
FMC_FLAG_PGERR	FMC programming error flag
FMC_FLAG_WPERR	FMC write protection error flag
FMC_FLAG_END	FMC end of programming flag
FMC_FLAG_OBERR	FMC option byte read error flag bit
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get FMC flag */
```

```
FlagStatus flag = fmc_flag_get(FMC_FLAG_END);
```

### fmc\_flag\_clear

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

**Table 3-229. Function fmc\_flag\_clear**

Function name	fmc_flag_clear
Function prototype	void fmc_flag_clear(uint32_t flag);
Function descriptions	clear FMC flag
Precondition	-
The called functions	-

Input parameter{in}	
<b>flag</b>	clear FMC flag
<i>FMC_FLAG_PGERR</i>	FMC operation error flag
<i>FMC_FLAG_WPERR</i>	FMC erase/program protection error flag
<i>FMC_FLAG_END</i>	FMC end of operation flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear FMC flag */
```

```
fmc_flag_clear(FMC_FLAG_END);
```

### fmc\_interrupt\_flag\_get

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

**Table 3-230. 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>	-
Input parameter{in}	
<b>flag</b>	FMC interrupt flags
<i>FMC_INT_FLAG_PGE</i> <i>RR</i>	FMC operation error flag
<i>FMC_INT_FLAG_WPE</i> <i>RR</i>	FMC erase/program protection error flag
<i>FMC_INT_FLAG_END</i>	FMC end of operation flag
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get FMC flag */
```

```
FlagStatus flag = fmc_interrupt_flag_get (FMC_INT_FLAG_PGERR);
```

### fmc\_interrupt\_flag\_clear

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

Table 3-231. Function `fmc_interrupt_flag_clear`

Function name	<code>fmc_interrupt_flag_clear</code>
Function prototype	<code>void fmc_interrupt_flag_clear(fmc_interrupt_flag_enum flag);</code>
Function descriptions	clear FMC interrupt flag state
Precondition	-
The called functions	-
Input parameter{in}	
flag	FMC interrupt flags
<code>FMC_INT_FLAG_PGE</code> <code>RR</code>	FMC operation error flag
<code>FMC_INT_FLAG_WPE</code> <code>RR</code>	FMC erase/program protection error flag
<code>FMC_INT_FLAG_END</code>	FMC end of operation flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear FMC flag */
```

```
fmc_interrupt_flag_get (FMC_INT_FLAG_PGERR);
```

### `fmc_state_get`

The description of `fmc_state_get` is shown as below:

Table 3-232. Function `fmc_state_get`

Function name	<code>fmc_state_get</code>
Function prototype	<code>fmc_state_enum fmc_state_get(void);</code>
Function descriptions	get the FMC state
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
<code>fmc_state_enum</code>	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-204. Enum <code>fmc_state</code> enum</a>

Example:

```
/* get the FMC state */
```

```
fmc_state_enum state = fmc_state_get( );
```

## fmc\_ready\_wait

The description of fmc\_ready\_wait is shown as below:

**Table 3-233. Function fmc\_ready\_wait**

<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>	timeout count
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-204. Enum fmc_state_enum</a>

Example:

```
/* check whether FMC is ready or not */
```

```
fmc_state_enum state = fmc_ready_wait (0x00001000 );
```

## 3.13. FWDGT

The free watchdog timer (FWDGT) is a hardware timing circuitry that can be used to detect system failures due to software malfunctions. It's suitable for the situation that requires an independent environment and lower timing accuracy. The FWDGT registers are listed in chapter [3.13.1](#) the FWDGT firmware functions are introduced in chapter [3.13.2](#).

### 3.13.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

**Table 3-234. FWDGT Registers**

Registers	Descriptions
FWDGT_CTL	Control register
FWDGT_PSC	Prescaler register
FWDGT_RLD	Reload register
FWDGT_STAT	Status register

### 3.13.2. Descriptions of Peripheral functions

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

Table 3-235. 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 FWDGT 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 3-236. Function fwdgt\_write\_enable

Function name	fwdgt_write_enable
Function prototype	void fwdgt_write_enable(void);
Function descriptions	enable write access to FWDGT_PSC and FWDGT_RLD
Precondition	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable write access to FWDGT_PSC and FWDGT_RLD */
```

```
fwdgt_write_enable ( );
```

### fwdgt\_write\_disable

The description of fwdgt\_write\_disable is shown as below:

Table 3-237. Function fwdgt\_write\_disable

Function name	fwdgt_write_disable
Function prototype	void fwdgt_write_disable(void);
Function descriptions	disable write access to FWDGT_PSC and FWDGT_RLD
Precondition	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-



Example:

```
/* disable write access to FWDGT_PSC and FWDGT_RLD */
```

```
fwdgt_write_disable ( );
```

### **fwdgt\_enable**

The description of fwdgt\_enable is shown as below:

**Table 3-238. Function fwdgt\_enable**

<b>Function name</b>	fwdgt_enable
<b>Function prototype</b>	void fwdgt_enable(void);
<b>Function descriptions</b>	start the FWDGT counter
<b>Precondition</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 3-239. 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>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 3-240. 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>Input parameter{in}</b>	
<b>reload_value</b>	specify reload value(0x0000 - 0x0FFF)
<b>Input parameter{in}</b>	
<b>prescaler_div</b>	FWDGT prescaler value
<i>FWDGT_PSC_DIV4</i>	FWDGT prescaler set to 4
<i>FWDGT_PSC_DIV8</i>	FWDGT prescaler set to 8
<i>FWDGT_PSC_DIV16</i>	FWDGT prescaler set to 16
<i>FWDGT_PSC_DIV32</i>	FWDGT prescaler set to 32
<i>FWDGT_PSC_DIV64</i>	FWDGT prescaler set to 64
<i>FWDGT_PSC_DIV128</i>	FWDGT prescaler set to 128
<i>FWDGT_PSC_DIV256</i>	FWDGT prescaler set to 256
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* confiure FWDGT counter clock: 40KHz(IRC40K) / 64 = 0.625 KHz */
```

```
fwdgt_config(2*500, FWDGT_PSC_DIV64);
```

## fwdgt\_flag\_get

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

**Table 3-241. Function fwdgt\_flag\_get**

<b>Function name</b>	fwdgt_flag_get
<b>Function prototype</b>	FlagStatus fwdgt_flag_get(uint16_t flag);
<b>Function descriptions</b>	get flag state of FWDGT
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flag to get
<i>FWDGT_FLAG_PUD</i>	a write operation to FWDGT_PSC register is on going
<i>FWDGT_FLAG_RUD</i>	a write operation to FWDGT_RLD register is on going
<b>Output parameter{out}</b>	
-	-

Return value	
FlagStatus	SET or RESET

Example:

```

/* test if a prescaler value update is on going */

FlagStatus status;

status = fwdgt_flag_get (FWDGT_FLAG_PUD);

if(status == RESET)

{

...

}else

{

...

}

```

## 3.14. GPIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter [3.14.1](#), the GPIO firmware functions are introduced in chapter [3.14.2](#).

### 3.14.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

**Table 3-242. GPIO Registers**

Registers	Descriptions
GPIOx_CTL0	GPIO port control register 0
GPIOx_CTL1	GPIO port control register 1
GPIOx_ISTAT	GPIO port input status register
GPIOx_OCTL	GPIO port output control register
GPIOx_BOP	GPIO port bit operate register
GPIOx_BC	GPIO port bit clear register
GPIOx_LOCK	GPIO port configuration lock register
AFIO_EC	AFIO event control register
AFIO_PCF0	AFIO port configuration register 0
AFIO_EXTISS0	AFIO port EXTI sources selection register 0
AFIO_EXTISS1	AFIO port EXTI sources selection register 1
AFIO_EXTISS2	AFIO port EXTI sources selection register 2
AFIO_EXTISS3	AFIO port EXTI sources selection register 3

Registers	Descriptions
AFIO_PCF1	AFIO port configuration register 1

### 3.14.2. Descriptions of Peripheral functions

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

**Table 3-243. 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 bit
gpio_bit_reset	reset GPIO pin bit
gpio_bit_write	write data to the specified GPIO pin
gpio_port_write	write data to the specified GPIO port
gpio_input_bit_get	get GPIO pin input status
gpio_input_port_get	get GPIO port input status
gpio_output_bit_get	get GPIO pin output status
gpio_output_port_get	get GPIO port output status
gpio_pin_remap_config	configure GPIO pin remap
gpio_exti_source_select	select GPIO pin exti sources
gpio_event_output_config	configure GPIO pin event output
gpio_event_output_enable	enable GPIO pin event output
gpio_event_output_disable	disable GPIO pin event output
gpio_pin_lock	lock GPIO pin bit

#### gpio\_deinit

The description of gpio\_deinit is shown as below:

**Table 3-244. Function gpio\_deinit**

<b>Function name</b>	gpio_deinit
<b>Function prototype</b>	void gpio_deinit(uint32_t gpio_periph);
<b>Function descriptions</b>	reset GPIO port
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,E)
<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 3-245. Function gpio\_afio\_deinit**

<b>Function name</b>	gpio_afio_deinit
<b>Function prototype</b>	void gpio_afio_deinit(void);
<b>Function descriptions</b>	reset alternate function I/O(AFIO)
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset alternate function */

gpio_afio_deinit();
```

### gpio\_init

The description of gpio\_init is shown as below:

**Table 3-246. Function gpio\_init**

<b>Function name</b>	gpio_init
<b>Function prototype</b>	void gpio_init(uint32_t gpio_periph, uint32_t mode, uint32_t speed, uint32_t pin);
<b>Function descriptions</b>	GPIO parameter initialization
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,E)
<b>Input parameter{in}</b>	
<b>mode</b>	gpio pin mode
<i>GPIO_MODE_AIN</i>	analog input mode
<i>GPIO_MODE_IN_FLO</i>	floating input mode

<i>ATING</i>	
<i>GPIO_MODE_IPD</i>	pull-down input mode
<i>GPIO_MODE_IPU</i>	pull-up input mode
<i>GPIO_MODE_OUT_OD</i>	GPIO output with open-drain
<i>GPIO_MODE_OUT_PP</i>	GPIO output with push-pull
<i>GPIO_MODE_AF_OD</i>	AFIO output with open-drain
<i>GPIO_MODE_AF_PP</i>	AFIO output with push-pull
<b>Input parameter{in}</b>	
<b>speed</b>	gpio output max speed value
<i>GPIO_OSPEED_10MHZ</i>	output max speed 10MHz
<i>GPIO_OSPEED_2MHZ</i>	output max speed 2MHz
<i>GPIO_OSPEED_50MHZ</i>	output max speed 50MHz
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config PA0 as 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 3-247. Function gpio\_bit\_set**

<b>Function name</b>	gpio_bit_set
<b>Function prototype</b>	void gpio_bit_set(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	set GPIO pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,E)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin

<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set PA0*/
```

```
gpio_bit_set(GPIOA, GPIO_PIN_0);
```

### gpio\_bit\_reset

The description of gpio\_bit\_reset is shown as below:

**Table 3-248. Function gpio\_bit\_reset**

<b>Function name</b>	gpio_bit_reset
<b>Function prototype</b>	void gpio_bit_reset(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	reset GPIO pin bit
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,E)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset PA0*/
```

```
gpio_bit_set(GPIOA, GPIO_PIN_0);
```

### gpio\_bit\_write

The description of gpio\_bit\_write is shown as below:

**Table 3-249. Function gpio\_bit\_write**

<b>Function name</b>	gpio_bit_write
----------------------	----------------

<b>Function prototype</b>	void gpio_bit_write(uint32_t gpio_periph,uint32_t pin,bit_status bit_value);
<b>Function descriptions</b>	write data to the specified GPIO pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,E)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Input parameter{in}</b>	
<b>bit_value</b>	SET or RESET
<i>RESET</i>	clear the port pin
<i>SET</i>	set the port pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* write 1 to PA0 */
```

```
gpio_bit_write(GPIOA, GPIO_PIN_0, SET);
```

## gpio\_port\_write

The description of gpio\_port\_write is shown as below:

**Table 3-250. Function gpio\_port\_write**

<b>Function name</b>	gpio_port_write
<b>Function prototype</b>	void gpio_port_write(uint32_t gpio_periph,uint16_t data);
<b>Function descriptions</b>	write data to the specified GPIO port
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,E)
<b>Input parameter{in}</b>	
<b>data</b>	specify the value to be written to the port output data register
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-



Example:

```
/*write 1010 0101 to Port A */
gpio_port_write (GPIOA, 0xA5);
```

## gpio\_input\_bit\_get

The description of gpio\_input\_bit\_get is shown as below:

**Table 3-251. Function gpio\_input\_bit\_get**

<b>Function name</b>	gpio_input_bit_get
<b>Function prototype</b>	FlagStatus gpio_input_bit_get(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	get GPIO pin input status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,E)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get status of PA0 */
FlagStatus bit_state;
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 3-252. Function gpio\_input\_port\_get**

<b>Function name</b>	gpio_input_port_get
<b>Function prototype</b>	uint16_t gpio_input_port_get(uint32_t gpio_periph);
<b>Function descriptions</b>	get GPIO port input status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,E)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	0x00-0xFF

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 3-253. Function gpio\_output\_bit\_get**

<b>Function name</b>	gpio_output_bit_get
<b>Function prototype</b>	FlagStatus gpio_output_bit_get(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	get GPIO pin output status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,E)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get output status of PA0 */

FlagStatus bit_state;

bit_state = gpio_output_bit_get(GPIOA, GPIO_PIN_0);
```

### gpio\_output\_port\_get

The description of gpio\_output\_port\_get is shown as below:

Table 3-254. 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>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,E)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	0x00-0xFF

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 3-255. Function `gpio_pin_remap_config`

<b>Function name</b>	<code>gpio_pin_remap_config</code>
<b>Function prototype</b>	<code>void gpio_pin_remap_config(uint32_t gpio_remap, ControlStatus newvalue);</code>
<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
<i>GPIO_SPI0_REMAP</i>	SPI0 remapping
<i>GPIO_I2C0_REMAP</i>	I2C0 remapping
<i>GPIO_USART0_REMAP</i>	USART0 remapping
<i>GPIO_USART1_REMAP</i>	USART1 remapping
<i>GPIO_USART2_PARTIAL_REMAP</i>	USART2 partial remapping
<i>GPIO_USART2_FULL_REMAP</i>	USART2 full remapping
<i>GPIO_TIMER0_PARTIAL_REMAP</i>	TIMER0 partial remapping

<i>AL_REMAP</i>	
<i>GPIO_TIMER0_FULL_REMAP</i>	TIMER0 full remapping
<i>GPIO_TIMER1_PARTIAL_REMAP0</i>	TIMER1 partial remapping
<i>GPIO_TIMER1_PARTIAL_REMAP1</i>	TIMER1 partial remapping
<i>GPIO_TIMER1_FULL_REMAP</i>	TIMER1 full remapping
<i>GPIO_TIMER2_PARTIAL_REMAP</i>	TIMER2 partial remapping
<i>GPIO_TIMER2_FULL_REMAP</i>	TIMER2 full remapping
<i>GPIO_TIMER3_REMAP</i>	TIMER3 remapping
<i>GPIO_CAN0_PARTIAL_REMAP</i>	CAN0 partial remapping
<i>GPIO_CAN0_FULL_REMAP</i>	CAN0 full remapping
<i>GPIO_PD01_REMAP</i>	PD01 remapping
<i>GPIO_TIMER4CH3_INTERNAL_REMAP</i>	TIMER4 channel3 internal remapping
<i>GPIO_CAN1_REMAP</i>	CAN1 remapping
<i>GPIO_SWJ_NONJTRST_REMAP</i>	JTAG-DP, but without NJTRST
<i>GPIO_SWJ_DISABLE_REMAP</i>	JTAG-DP disabled
<i>GPIO_SPI2_REMAP</i>	SPI2 remapping
<i>GPIO_TIMER1_IT1_REMAP</i>	TIMER1 internal trigger 1 remapping
<i>GPIO_EXMC_NADV_REMAP</i>	EXMC_NADV connect/disconnect
<b>Input parameter{in}</b>	
<b>newvalue</b>	ENABLE or DISABLE
<i>ENABLE</i>	<i>enable</i>
<i>DISABLE</i>	<i>disable</i>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*enable SPI0 remapping*/
```

```
gpio_pin_remap_config (GPIO_SPI0_REMAP, ENABLE);
```

## gpio\_exti\_source\_select

The description of gpio\_exti\_source\_select is shown as below:

**Table 3-256. Function gpio\_exti\_source\_select**

<b>Function name</b>	gpio_exti_source_select
<b>Function prototype</b>	void gpio_exti_source_select(uint8_t gpio_outputport,uint8_t gpio_outputpin);
<b>Function descriptions</b>	select GPIO pin exti sources
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_outputport</b>	gpio event output port
<b>GPIO_PORT_SOURCE_GPIOx</b>	output port source (x=A..E)
<b>Input parameter{in}</b>	
<b>gpio_outputpin</b>	gpio event output pin
<b>GPIO_PIN_SOURCE_x</b>	Pin number(x=0..15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config PA0 as EXTI source*/
```

```
gpio_exti_source_select(GPIO_PORT_SOURCE_GPIOA, GPIO_PIN_SOURCE_0);
```

## gpio\_event\_output\_config

The description of gpio\_event\_output\_config is shown as below:

**Table 3-257. Function gpio\_event\_output\_config**

<b>Function name</b>	gpio_event_output_config
<b>Function prototype</b>	void gpio_event_output_config(uint8_t gpio_outputport,uint8_t gpio_outputpin);
<b>Function descriptions</b>	configure GPIO pin event output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_outputport</b>	gpio event output port
<b>GPIO_EVENT_PORT_GPIOx</b>	event output port x (x=A..E)

Input parameter{in}	
<b>gpio_outputpin</b>	gpio event output pin
<i>GPIO_EVENT_PIN_x</i>	Pin number (x=0..15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* Config PA0 as the output of event */
```

```
gpio_event_output_config(GPIO_EVENT_PORT_GPIOA, GPIO_EVENT_PIN_0);
```

### gpio\_event\_output\_enable

The description of gpio\_event\_output\_enable is shown as below:

**Table 3-258. 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>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable GPIO pin event output */
```

```
gpio_event_output_enable();
```

### gpio\_event\_output\_disable

The description of gpio\_event\_output\_disable is shown as below:

**Table 3-259. 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>	-

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

Example:

```
/* disable GPIO pin event output */
```

```
gpio_event_output_disable();
```

### gpio\_pin\_lock

The description of gpio\_pin\_lock is shown as below:

**Table 3-260. Function gpio\_pin\_lock**

Function name	gpio_pin_lock
Function prototype	void gpio_pin_lock(uint32_t gpio_periph,uint32_t pin);
Function descriptions	lock GPIO pin
Precondition	-
The called functions	-
Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	<i>GPIOx(x = A,B,C,D,E)</i>
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock PA0 */
```

```
gpio_pin_lock(GPIOA, GPIO_PIN_0);
```

## 3.15. I2C

The I2C (inter-integrated circuit) module provides an I2C interface which is an industry standard two-line serial interface for MCU to communicate with external I2C interface. The I2C registers are listed in chapter [3.15.1](#), the I2C firmware functions are introduced in chapter

### 3.15.2.

#### 3.15.1. Descriptions of Peripheral registers

I2C registers are listed in the table shown as below:

**Table 3-261. I2C Registers**

Registers	Descriptions
I2C_CTL0	Control register 0
I2C_CTL1	Control register 1
I2C_SADDR0	Slave address register 0
I2C_SADDR1	Slave address register 1
I2C_DATA	Transfer buffer register
I2C_STAT0	Transfer status register 0
I2C_STAT1	Transfer status register 1
I2C_CKCFG	Clock configure register
I2C_RT	Rise time register
I2C_FMPCFG	Fast mode plus configure register

#### 3.15.2. Descriptions of Peripheral functions

I2C firmware functions are listed in the table shown as below:

**Table 3-262. 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 ACK position
i2c_master_addressing	master send slave address
i2c_dualaddr_enable	enable dual-address mode
i2c_dualaddr_disable	disable dual-address mode
i2c_enable	enable I2C
i2c_disable	disable I2C
i2c_start_on_bus	generate a START condition on I2C bus
i2c_stop_on_bus	generate a STOP condition on I2C bus
i2c_data_transmit	I2C transmit data function
i2c_data_receive	I2C receive data function
i2c_dma_config	I2C DMA mode enable
i2c_dma_last_transfer_config	configure whether next DMA EOT is DMA last transfer or not
i2c_stretch_scl_low_config	whether to stretch SCL low when data is not ready in slave



Function name	Function description
	mode
i2c_slave_response_to_gcall_config	whether or not to response to a general call
i2c_software_reset_config	configure software reset of I2C
i2c_pec_config	configure I2C PEC calculation
i2c_pec_transfer_config	configure whether to transfer PEC value
i2c_pec_value_get	packet error checking value
i2c_smbus_alert_config	configure I2C alert through SMBA pin
i2c_smbus_arp_config	configure I2C ARP protocol in SMBus
i2c_flag_get	get I2C flag status
i2c_flag_clear	clear I2C flag status
i2c_interrupt_enable	enable I2C interrupt
i2c_interrupt_disable	disable I2C interrupt
i2c_interrupt_flag_get	get I2C interrupt flag status
i2c_interrupt_flag_clear	clear I2C interrupt flag status

## i2c\_deinit

The description of i2c\_deinit is shown as below:

**Table 3-263. Function i2c\_deinit**

Function name	i2c_deinit
Function prototype	void i2c_deinit(uint32_t i2c_periph);
Function descriptions	reset I2C
Precondition	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset I2C0 */
```

```
i2c_deinit (I2C0);
```

## i2c\_clock\_config

The description of i2c\_clock\_config is shown as below:

**Table 3-264. Function i2c\_clock\_config**

Function name	i2c_clock_config
Function prototype	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>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>clkspeed</b>	i2c clock speed
<b>Input parameter{in}</b>	
<b>dutycyc</b>	duty cycle in fast mode
<i>I2C_DTCY_2</i>	T_low/T_high=2
<i>I2C_DTCY_16_9</i>	T_low/T_high=16/9
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure I2C0 clock speed as 100KHz*/
```

```
i2c_clock_config(I2C0, 100000, I2C_DTCY_2);
```

### i2c\_mode\_addr\_config

The description of i2c\_mode\_addr\_config is shown as below:

**Table 3-265. 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>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_ENABLER</i>	I2C mode
<i>I2C_SMBUSMODE_ENABLED</i>	SMBus mode
<b>Input parameter{in}</b>	
<b>addformat</b>	7bits or 10bits
<i>I2C_ADDFORMAT_7BI</i>	7bits

<i>TS</i>	
<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 3-266. 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>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>type</b>	Device or host
<i>I2C_SMBUS_DEVICE</i>	device
<i>I2C_SMBUS_HOST</i>	host
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config I2C0 as SMBUS host type*/
```

```
i2c_smbus_type_config (I2C0, I2C_SMBUS_HOST);
```

### i2c\_ack\_config

The description of i2c\_ack\_config is shown as below:

Table 3-267. Function `i2c_ack_config`

<b>Function name</b>	<code>i2c_ack_config</code>
<b>Function prototype</b>	<code>void i2c_ack_config(uint32_t i2c_periph, uint32_t ack);</code>
<b>Function descriptions</b>	whether or not to send an ACK
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>ack</b>	whether or not to send an ACK
<i>I2C_ACK_ENABLE</i>	ACK will be sent
<i>I2C_ACK_DISABLE</i>	ACK will not be sent
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 will send ACK */
```

```
i2c_ack_config (I2C0, I2C_ACK_ENABLE);
```

### `i2c_ackpos_config`

The description of `i2c_ackpos_config` is shown as below:

Table 3-268. Function `i2c_ackpos_config`

<b>Function name</b>	<code>i2c_ackpos_config</code>
<b>Function prototype</b>	<code>void i2c_ackpos_config(uint32_t i2c_periph, uint32_t pos);</code>
<b>Function descriptions</b>	I2C POAP position configure
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>pos</b>	ACK position
<i>I2C_ACKPOS_CURRENT</i>	whether to send ACK or not for the current
<i>I2C_ACKPOS_NEXT</i>	whether to send ACK or not for the next byte
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* The ACK of I2C0 is send for the current frame*/
```

```
i2c_ackpos_config (I2C0, I2C_ACKPOS_CURRENT);
```

### i2c\_master\_addressing

The description of i2c\_master\_addressing is shown as below:

**Table 3-269. Function i2c\_master\_addressing**

<b>Function name</b>	i2c_master_addressing
<b>Function prototype</b>	void i2c_master_addressing(uint32_t i2c_periph, uint32_t addr, uint32_t trandirection);
<b>Function descriptions</b>	master sends slave address
<b>Precondition</b>	-
<b>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 3-270. 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 addr)
<b>Function descriptions</b>	dual-address mode enable
<b>Precondition</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>	second address in dual-address mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 dual-address*/
```

```
i2c_dualaddr_enable (I2C0, 0x80);
```

### **i2c\_dualaddr\_disable**

The description of i2c\_dualaddr\_disable is shown as below:

**Table 3-271. Function i2c\_dualaddr\_enable**

<b>Function name</b>	i2c_dualaddr_disable
<b>Function prototype</b>	void i2c_dualaddr_disable(uint32_t i2c_periph)
<b>Function descriptions</b>	dual-address mode disable
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C0 dual-address*/
```

```
i2c_dualaddr_disable (I2C0);
```

### **i2c\_enable**

The description of i2c\_enable is shown as below:

**Table 3-272. 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>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 3-273. 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>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C0 */
```

```
i2c_disable (I2C0);
```

### i2c\_start\_on\_bus

The description of i2c\_start\_on\_bus is shown as below:

**Table 3-274. 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>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral

<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 send a start condition to I2C bus */
```

```
i2c_start_on_bus (I2C0);
```

### i2c\_stop\_on\_bus

The description of i2c\_stop\_on\_bus is shown as below:

**Table 3-275. 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>	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

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 3-276. 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>	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)



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

Example:

```
/* I2C0 transmit data */
```

```
i2c_data_transmit (I2C0, 0x80);
```

### i2c\_data\_receive

The description of i2c\_data\_receive is shown as below:

**Table 3-277. Function i2c\_data\_receive**

Function name	i2c_data_receive
Function prototype	uint8_t i2c_data_receive(uint32_t i2c_periph);
Function descriptions	I2C receive data function
Precondition	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
uint8_t	0x00..0xFF

Example:

```
/* I2C0 receive data */
```

```
uint8_t i2c_receiver;
```

```
i2c_receiver = i2c_data_receive(I2C0);
```

### i2c\_dma\_config

The description of i2c\_dma\_config is shown as below:

**Table 3-278. Function i2c\_dma\_config**

Function name	i2c_dma_config
Function prototype	void i2c_dma_config(uint32_t i2c_periph, uint32_t dmastate);
Function descriptions	enable I2C DMA mode
Precondition	-
Input parameter{in}	

<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>dmastate</b>	on or off
<i>I2C_DMA_ON</i>	DMA mode enable
<i>I2C_DMA_OFF</i>	DMA mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 DMA mode enable */
```

```
i2c_dma_config(I2C0, I2C_DMA_ON);
```

### i2c\_dma\_last\_transfer\_config

The description of i2c\_dma\_last\_transfer\_config is shown as below:

**Table 3-279. Function i2c\_dma\_last\_transfer\_config**

<b>Function name</b>	i2c_dma_last_transfer_config
<b>Function prototype</b>	void i2c_dma_last_transfer_config(uint32_t i2c_periph, uint32_t dmalast);
<b>Function descriptions</b>	configure whether next DMA EOT is DMA last transfer or not
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>dmalast</b>	next DMA EOT is the last transfer or not
<i>I2C_DMALST_ON</i>	next DMA EOT is the last transfer
<i>I2C_DMALST_OFF</i>	next DMA EOT is not the last transfer
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* next DMA EOT is the last transfer */
```

```
i2c_dma_last_transfer_config (I2C0, I2C_DMALST_ON);
```

### i2c\_stretch\_scl\_low\_config

The description of i2c\_stretch\_scl\_low\_config is shown as below:

Table 3-280. Function `i2c_stretch_scl_low_config`

<b>Function name</b>	<code>i2c_stretch_scl_low_config</code>
<b>Function prototype</b>	<code>void i2c_stretch_scl_low_config(uint32_t i2c_periph, uint32_t stretchpara);</code>
<b>Function descriptions</b>	whether to stretch SCL low when data is not ready in slave mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>stretchpara</b>	SCL stretching enable or disable
<i>I2C_SCLSTRETCH_ENABLE</i>	SCL stretching is enabled
<i>I2C_SCLSTRETCH_DISABLE</i>	SCL stretching is disabled
<b>Output parameter{out}</b>	
-	-
<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 3-281. Function `i2c_slave_response_to_gcall_config`

<b>Function name</b>	<code>i2c_slave_response_to_gcall_config</code>
<b>Function prototype</b>	<code>void i2c_slave_response_to_gcall_config(uint32_t i2c_periph, uint32_t gcallpara);</code>
<b>Function descriptions</b>	whether or not to response to a general call
<b>Precondition</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 3-282. 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>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>sreset</b>	reset or not
<i>I2C_SRESET_SET</i>	I2C is under reset
<i>I2C_SRESET_RESET</i>	I2C is not under reset
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* software reset I2C0 */
```

```
i2c_software_reset_config (I2C0, I2C_SRESET_SET);
```

### i2c\_pec\_config

The description of i2c\_pec\_config is shown as below:

**Table 3-283. Function i2c\_pec\_config**

<b>Function name</b>	i2c_pec_config
<b>Function prototype</b>	void i2c_pec_config(uint32_t i2c_periph, uint32_t pecstate);
<b>Function descriptions</b>	configure I2C PEC calculation
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)

Input parameter{in}	
<b>pecstate</b>	on or off
<i>I2C_PEC_ENABLE</i>	PEC calculation on
<i>I2C_PEC_DISABLE</i>	PEC calculation off
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* Enable I2C PEC calculation */
```

```
i2c_pec_config(I2C0, I2C_PEC_ENABLE);
```

### i2c\_pec\_transfer\_config

The description of i2c\_pec\_transfer\_config is shown as below:

**Table 3-284. Function i2c\_pec\_transfer\_config**

<b>Function name</b>	i2c_pec_transfer_config
<b>Function prototype</b>	void i2c_pec_transfer_config(uint32_t i2c_periph, uint32_t pecpara);
<b>Function descriptions</b>	configure whether to transfer PEC value
<b>Precondition</b>	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
<b>pecpara</b>	Transfer PEC or not
<i>I2C_PECTRANS_ENABLE</i>	transfer PEC
<i>I2C_PECTRANS_DISABLE</i>	not transfer PEC
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 transfer PEC */
```

```
i2c_pec_transfer_config(I2C0, I2C_PECTRANS_ENABLE);
```

### i2c\_pec\_value\_get

The description of i2c\_pec\_value\_get is shown as below:

Table 3-285. Function `i2c_pec_value_get`

Function name	<code>i2c_pec_value_get</code>
Function prototype	<code>uint8_t i2c_pec_value_get(uint32_t i2c_periph);</code>
Function descriptions	get packet error checking value
Precondition	-
Input parameter{in}	
<code>i2c_periph</code>	I2C peripheral
<code>I2Cx</code>	(x=0,1)
Output parameter{out}	
-	-
Return value	
<code>uint8_t</code>	PEC value

Example:

```
/* I2C0 get packet error checking value */
```

```
uint8_t pec_value;
```

```
pec_value = i2c_pec_value_get (I2C0);
```

### `i2c_smbus_alert_config`

The description of `i2c_smbus_alert_config` is shown as below:

Table 3-286. Function `i2c_smbus_alert_config`

Function name	<code>i2c_smbus_alert_config</code>
Function prototype	<code>void i2c_smbus_alert_config(uint32_t i2c_periph, uint32_t smbuspara);</code>
Function descriptions	configure I2C alert through SMBA pin
Precondition	-
Input parameter{in}	
<code>i2c_periph</code>	I2C peripheral
<code>I2Cx</code>	(x=0,1)
Input parameter{in}	
<code>smbuspara</code>	issue alert through SMBA pin or not
<code>I2C_SALTSEND_ENABLE</code>	issue alert through SMBA pin
<code>I2C_SALTSEND_DISABLE</code>	not issue alert through SMBA pin
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 issue alert through SMBA pin enable */
```

```
i2c_smbus_alert_config(I2C0, I2C_SALTSEND_ENABLE);
```

## i2c\_smbus\_arp\_config

The description of i2c\_smbus\_arp\_config is shown as below:

**Table 3-287. Function i2c\_smbus\_arp\_config**

<b>Function name</b>	i2c_smbus_arp_config
<b>Function prototype</b>	void i2c_smbus_arp_config(uint32_t i2c_periph, uint32_t arpstate);
<b>Function descriptions</b>	configure I2C ARP protocol in SMBus
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>arpstate</b>	ARP protocol in SMBus switch
<i>I2C_ARP_ENABLE</i>	enable ARP
<i>I2C_ARP_DISABLE</i>	disable ARP
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 ARP protocol in SMBus switch */
```

```
i2c_smbus_arp_config(I2C0, I2C_ARP_ENABLE);
```

## i2c\_flag\_get

The description of i2c\_flag\_get is shown as below:

**Table 3-288. Function i2c\_flag\_get**

<b>Function name</b>	i2c_flag_get
<b>Function prototype</b>	FlagStatus i2c_flag_get(uint32_t i2c_periph, i2c_flag_enum flag)
<b>Function descriptions</b>	get I2C flag status
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>I2C_FLAG_SBSSEND</i>	start condition send out
<i>I2C_FLAG_ADDSEND</i>	address is sent in master mode or received and matches in slave mode
<i>I2C_FLAG_BTC</i>	byte transmission finishes

<i>I2C_FLAG_ADD10SEND</i>	header of 10-bit address is sent in master mode
<i>I2C_FLAG_STPDET</i>	stop condition detected in slave mode
<i>I2C_FLAG_RBNE</i>	I2C_DATA is not Empty during receiving
<i>I2C_FLAG_TBE</i>	I2C_DATA is empty during transmitting
<i>I2C_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus
<i>I2C_FLAG_LOSTARB</i>	arbitration lost in master mode
<i>I2C_FLAG_AERR</i>	acknowledge error
<i>I2C_FLAG_OUERR</i>	overflow or 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_TR</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
<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 3-289. Function i2c\_flag\_clear**

<b>Function name</b>	i2c_flag_clear
<b>Function prototype</b>	void i2c_flag_clear(uint32_t i2c_periph, i2c_flag_enum flag)
<b>Function descriptions</b>	clear I2C flag status
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral



<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>flag</b>	flag type
<i>I2C_FLAG_SMBALT</i>	SMBus Alert status
<i>I2C_FLAG_SMBTO</i>	timeout signal in SMBus mode
<i>I2C_FLAG_PECERR</i>	PEC error when receiving data
<i>I2C_FLAG_OUERR</i>	over-run or under-run situation occurs in slave mode
<i>I2C_FLAG_AERR</i>	acknowledge error
<i>I2C_FLAG_LOSTARB</i>	arbitration lost in master mode
<i>I2C_FLAG_BERR</i>	a bus error
<i>I2C_FLAG_ADDSEND</i>	cleared by reading I2C_STAT0 and reading I2C_STAT1
<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 3-290. Function i2c\_interrupt\_enable**

<b>Function name</b>	i2c_interrupt_enable
<b>Function prototype</b>	void i2c_interrupt_enable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);
<b>Function descriptions</b>	enable I2C interrupt
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>inttype</b>	interrupt type
<i>I2C_INT_ERR</i>	error interrupt
<i>I2C_INT_EV</i>	event interrupt
<i>I2C_INT_BUF</i>	buffer interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 event interrupt */
```

```
i2c_interrupt_enable (I2C0, I2C_INT_EV);
```

### i2c\_interrupt\_disable

The description of i2c\_interrupt\_disable is shown as below:

**Table 3-291. Function i2c\_interrupt\_disable**

<b>Function name</b>	i2c_interrupt_disable
<b>Function prototype</b>	void i2c_interrupt_disable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);
<b>Function descriptions</b>	disable I2C interrupt
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>inttype</b>	interrupt type
<i>I2C_INT_ERR</i>	error interrupt
<i>I2C_INT_EV</i>	event interrupt
<i>I2C_INT_BUF</i>	buffer interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C0 event interrupt */
```

```
i2c_interrupt_disable (I2C0, I2C_INT_EV);
```

### i2c\_interrupt\_flag\_get

The description of i2c\_interrupt\_flag\_get is shown as below:

**Table 3-292. Function i2c\_interrupt\_flag\_get**

<b>Function name</b>	i2c_interrupt_flag_get
<b>Function prototype</b>	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag)
<b>Function descriptions</b>	get I2C interrupt flag status
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)

Input parameter{in}	
<b>int_flag</b>	interrupt flag
<i>I2C_INT_FLAG_SBSE ND</i>	start condition sent out in master mode interrupt flag
<i>I2C_INT_FLAG_ADDS END</i>	address is sent in master mode or received and matches in slave mode interrupt flag
<i>I2C_INT_FLAG_BTC</i>	byte transmission finishes
<i>I2C_INT_FLAG_ADD10 SEND</i>	header of 10-bit address is sent in master mode interrupt flag
<i>I2C_INT_FLAG_STPD ET</i>	stop condition detected in slave mode interrupt flag
<i>I2C_INT_FLAG_RBNE</i>	I2C_DATA is not Empty during receiving interrupt flag
<i>I2C_INT_FLAG_TBE</i>	I2C_DATA is empty during transmitting interrupt flag
<i>I2C_INT_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag
<i>I2C_INT_FLAG_LOSTA RB</i>	arbitration lost in master mode interrupt flag
<i>I2C_INT_FLAG_AERR</i>	acknowledge error interrupt flag
<i>I2C_INT_FLAG_OUER R</i>	over-run or under-run situation occurs in slave mode interrupt flag
<i>I2C_INT_FLAG_PECE RR</i>	PEC error when receiving data interrupt flag
<i>I2C_INT_FLAG_SMBT O</i>	timeout signal in SMBus mode interrupt flag
<i>I2C_INT_FLAG_SMBA LT</i>	SMBus Alert status interrupt flag
Output parameter{out}	
-	-
Return value	
<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 3-293. Function i2c\_interrupt\_flag\_clear**

<b>Function name</b>	i2c_interrupt_flag_clear
<b>Function prototype</b>	void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum

	int_flag);
<b>Function descriptions</b>	clear I2C interrupt flag status
<b>Precondition</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_ADDS END</i>	address is sent in master mode or received and matches in slave mode interrupt flag
<i>I2C_INT_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag
<i>I2C_INT_FLAG_LOSTA RB</i>	arbitration lost in master mode interrupt flag
<i>I2C_INT_FLAG_AERR</i>	acknowledge error interrupt flag
<i>I2C_INT_FLAG_OUER R</i>	over-run or under-run situation occurs in slave mode interrupt flag
<i>I2C_INT_FLAG_PEC RR</i>	PEC error when receiving data interrupt flag
<i>I2C_INT_FLAG_SMBT O</i>	timeout signal in SMBus mode interrupt flag
<i>I2C_INT_FLAG_SMBA LT</i>	SMBus Alert status interrupt flag
<i>I2C_INT_FLAG_TFF</i>	txframe fall interrupt flag
<i>I2C_INT_FLAG_TFR</i>	txframe rise interrupt flag
<i>I2C_INT_FLAG_RFF</i>	rxframe fall interrupt flag
<i>I2C_INT_FLAG_RFR</i>	rxframe rise interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the acknowledge error interrupt flag */
```

```
i2c_interrupt_flag_clear (I2C0, I2C_INT_FLAG_AERR);
```

## 3.16. 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 [3.16.1](#), the PMU firmware functions are introduced in chapter [3.16.2](#).

### 3.16.1. Descriptions of Peripheral registers

PMU registers are listed in the table shown as below:

**Table 3-294. PMU Registers**

Registers	Descriptions
PMU_CTL	PMU control register
PMU_CS	PMU control and status register

### 3.16.2. Descriptions of Peripheral functions

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

**Table 3-295. PMU firmware function**

Function name	Function description
pmu_deinit	reset PMU register
pmu_lvd_select	select low voltage detector threshold
pmu_lvd_disable	disable PMU lvd
pmu_to_sleepmode	PMU work in sleep mode
pmu_to_deepsleepmode	PMU work at deepsleep mode
pmu_to_standbymode	pmu work at standby mode
pmu_wakeup_pin_enable	enable PMU wakeup pin
pmu_wakeup_pin_disable	disable PMU 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 3-296. Function pmu\_deinit**

Function name	pmu_deinit
Function prototype	void pmu_deinit(void);
Function descriptions	reset PMU register
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset PMU */
```

```
pmu_deinit ();
```

## pmu\_lvd\_select

The description of pmu\_lvd\_select is shown as below:

**Table 3-297. Function pmu\_lvd\_select**

Function name	pmu_lvd_select
Function prototype	void pmu_lvd_select(uint32_t lvd_t_n);
Function descriptions	select low voltage detector threshold
Precondition	-
The called functions	-
Input parameter{in}	
lvd_t_n	voltage threshold value
PMU_LVDT_0	voltage threshold is 2.2V
PMU_LVDT_1	voltage threshold is 2.3V
PMU_LVDT_2	voltage threshold is 2.4V
PMU_LVDT_3	voltage threshold is 2.5V
PMU_LVDT_4	voltage threshold is 2.6V
PMU_LVDT_5	voltage threshold is 2.7V
PMU_LVDT_6	voltage threshold is 2.8V
PMU_LVDT_7	voltage threshold is 2.9V
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select low voltage detector threshold as 2.9V */
```

```
pmu_lvd_select ();
```

## pmu\_lvd\_disable

The description of pmu\_lvd\_disable is shown as below:

**Table 3-298. Function pmu\_lvd\_disable**

Function name	pmu_lvd_disable
Function prototype	void pmu_lvd_disable (void);
Function descriptions	disable PMU lvd
Precondition	-
The called functions	-

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

Example:

```
/* disable PMU lvd */
pmu_lvd_disable ();
```

### pmu\_to\_sleepmode

The description of pmu\_to\_sleepmode is shown as below:

**Table 3-299. Function pmu\_to\_sleepmode**

Function name	pmu_to_sleepmode
Function prototype	void pmu_to_sleepmode(uint8_t sleepmodecmd);
Function descriptions	PMU work at sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
sleepmodecmd	command to enter sleep mode
WFI_CMD	use WFI command
WFE_CMD	use WFE command
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* PMU work at sleep mode */
pmu_to_sleepmode (WFI_CMD);
```

### pmu\_to\_deepsleepmode

The description of pmu\_to\_deepsleepmode is shown as below:

**Table 3-300. Function pmu\_to\_deepsleepmode**

Function name	pmu_to_deepsleepmode
Function prototype	void pmu_to_deepsleepmode(uint32_t ldo,uint8_t deepsleepmodecmd);
Function descriptions	PMU work at deepsleep mode
Precondition	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ldo</b>	ldo work mode
<i>PMU_LDO_NORMAL</i>	LDO operates normally when pmu enter deepsleep mode
<i>PMU_LDO_LOWPOWER</i>	LDO work at low power mode when pmu enter deepsleep mode
<b>Input parameter{in}</b>	
<b>deepsleepmodecmd</b>	command to enter deepsleep mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
<b>Output parameter{out}</b>	
-	-
<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 3-301. 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
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work at standby mode */
```

```
pmu_to_standby (WFI_CMD);
```



## pmu\_wakeup\_pin\_enable

The description of pmu\_wakeup\_pin\_enable is shown as below:

**Table 3-302. 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 3-303. Function pmu\_wakeup\_pin\_disable**

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

Example:

```
/* disable wakeup pin */
pmu_wakeup_pin_disable ();
```

## pmu\_backup\_write\_enable

The description of pmu\_backup\_write\_enable is shown as below:

**Table 3-304. 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 3-305. Function pmu\_backup\_write\_disable**

<b>Function name</b>	pmu_backup_write_disable
<b>Function prototype</b>	void pmu_backup_write_disable (void);
<b>Function descriptions</b>	disable backup domain write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable backup domain write */
pmu_backup_write_disable ();
```

## pmu\_flag\_clear

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

**Table 3-306. Function pmu\_flag\_clear**

<b>Function name</b>	pmu_flag_clear
<b>Function prototype</b>	void pmu_flag_clear(uint32_t flag_clear);
<b>Function descriptions</b>	clear flag bit
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag_clear</b>	flag
PMU_FLAG_RESET_WAKEUP	reset wakeup flag
PMU_FLAG_RESET_STANDBY	reset standby flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear flag bit */
```

```
pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

## pmu\_flag\_get

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

**Table 3-307. Function pmu\_flag\_get**

<b>Function name</b>	pmu_flag_get
<b>Function prototype</b>	FlagStatus pmu_flag_get(uint32_t flag);
<b>Function descriptions</b>	get flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flag
PMU_FLAG_WAKEUP	wakeup flag
PMU_FLAG_STANDBY	standby flag
PMU_FLAG_LVD	lvd flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

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

```
FlagStatus status;
```

```
status = pmu_flag_get (PMU_FLAG_WAKEUP);
```

## 3.17. 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 [3.17.1](#), the RCU firmware functions are introduced in chapter [3.17.2](#).

### 3.17.1. Descriptions of Peripheral registers

**Table 3-308. RCU Registers**

Registers	Descriptions
RCU_CTL	Control register
RCU_CFG0	Clock configuration register 0
RCU_INT	Clock interrupt register
RCU_APB2RST	APB2 reset register
RCU_APB1RST	APB1 reset register
RCU_AHBEN	AHB enable register
RCU_APB2EN	APB2 enable register
RCU_APB1EN	APB1 enable register
RCU_BDCTL	Backup domain control register
RCU_RSTSCK	Reset source/clock register
RCU_AHBRST	AHB reset register
RCU_CFG1	Clock configuration register 1
RCU_DSV	Deep-sleep mode voltage register

### 3.17.2. Descriptions of Peripheral functions

**Table 3-309. 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

Function name	Function description
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_predv0_config	configure the PREDV0 division factor and clock source
rcu_predv1_config	configure the PREDV1 division factor
rcu_pll1_config	configure the PLL1 clock
rcu_pll2_config	configure the PLL2 clock
rcu_adc_clock_config	configure the ADC division 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_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_osci_stab_wait	wait for oscillator stabilization flags is SET or oscillator startup is timeout
rcu_osci_on	turn on the oscillator
rcu_osci_off	turn off the oscillator
rcu_osci_bypass_mode_enable	enable the oscillator bypass mode
rcu_osci_bypass_mode_disable	disable the oscillator bypass mode
rcu_hxtal_clock_monitor_enable	enable the HXTAL clock monitor
rcu_hxtal_clock_monitor_disable	disable the HXTAL clock monitor
rcu_irc8m_adjust_value_set	set the IRC8M adjust value
rcu_deepsleep_voltage_set	set the deep-sleep mode voltage value
rcu_clock_freq_get	get the system clock, bus clock frequency

## rcu\_deinit

The description of rcu\_deinit is shown as below:

**Table 3-310. Function rcu\_deinit**

Function name	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>	-
<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 3-311. 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
<i>RCU_GPIOx</i>	GPIO ports clock (x=A,B,C,D,E)
<i>RCU_AF</i>	alternate function clock
<i>RCU_CRC</i>	CRC clock
<i>RCU_DMAx</i>	DMAx clock (x=0,1)
<i>RCU_USBFS</i>	USBFS clock
<i>RCU_EXMC</i>	EXMC clock
<i>RCU_TIMERx</i>	TIMERx clock (x=0,1,2,3,4,5,6)
<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)
<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)
<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 3-312. 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)
<i>RCU_AF</i>	alternate function clock
<i>RCU_CRC</i>	CRC clock
<i>RCU_DMAx</i>	DMAx clock (x=0,1)
<i>RCU_USBFS</i>	USBFS clock
<i>RCU_EXMC</i>	EXMC clock
<i>RCU_TIMERx</i>	TIMERx clock (x=0,1,2,3,4,5,6)
<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)
<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)
<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)
<i>RCU_BKPI</i>	BKP interface clock
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* disable the USART0 clock */
```

```
rcu_periph_clock_disable(RCU_USART0);
```

### rcu\_periph\_clock\_sleep\_enable

The description of rcu\_periph\_clock\_sleep\_enable is shown as below:

**Table 3-313. Function rcu\_periph\_clock\_sleep\_enable**

Function name	rcu_periph_clock_sleep_enable
Function prototype	void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);
Function descriptions	enable the peripherals clock when in sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to rcu_periph_sleep_enum
<i>RCU_FMC_SLP</i>	FMC clock
<i>RCU_SRAM_SLP</i>	SRAM clock
Output parameter{out}	
-	-
Return value	
-	-

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 3-314. Function rcu\_periph\_clock\_sleep\_disable**

Function name	rcu_periph_clock_sleep_disable
Function prototype	void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph);
Function descriptions	disable the peripherals clock when in sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to rcu_periph_sleep_enum
<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 3-315. 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)
<i>RCU_AFRST</i>	reset alternate function clock
<i>RCU_USBFSRST</i>	reset USBFS clock
<i>RCU_TIMERxRST</i>	reset TIMERx clock (x=0,1,2,3,4,5,6)
<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)
<i>RCU_PMURST</i>	reset PMU clock
<i>RCU_DACRST</i>	reset DAC clock
<i>RCU_ADCxRST</i>	reset ADCx clock (x=0,1)
<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 3-316. 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 rcu_periph_reset_enum
<i>RCU_GPIOxRST</i>	reset GPIO ports clock (x=A,B,C,D,E)
<i>RCU_AFRST</i>	reset alternate function clock
<i>RCU_USBFSRST</i>	reset USBFS clock
<i>RCU_TIMERxRST</i>	reset TIMERx clock (x=0,1,2,3,4,5,6)
<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)
<i>RCU_PMURST</i>	reset PMU clock
<i>RCU_DACRST</i>	reset DAC clock
<i>RCU_ADCxRST</i>	reset ADCx clock (x=0,1)
<i>RCU_BKPIRST</i>	reset BKPI clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 reset */
```

```
rcu_periph_reset_disable(RCU_SPI0RST);
```

### rcu\_bkp\_reset\_enable

The description of rcu\_bkp\_reset\_enable is shown as below:

**Table 3-317. Function rcu\_bkp\_reset\_enable**

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

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

Example:

```
/* reset the BKP domain */
```

```
rcu_bkp_reset_enable();
```

### rcu\_bkp\_reset\_disable

The description of rcu\_bkp\_reset\_disable is shown as below:

**Table 3-318. Function rcu\_bkp\_reset\_disable**

<b>Function name</b>	rcu_bkp_reset_disable
<b>Function prototype</b>	void rcu_bkp_reset_disable(void);
<b>Function descriptions</b>	disable the BKP domain reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the BKP domain reset */
```

```
rcu_bkp_reset_disable();
```

### rcu\_system\_clock\_source\_config

The description of rcu\_system\_clock\_source\_config is shown as below:

**Table 3-319. Function rcu\_system\_clock\_source\_config**

<b>Function name</b>	rcu_system_clock_source_config
<b>Function prototype</b>	void rcu_system_clock_source_config(uint32_t ck_sys);
<b>Function descriptions</b>	configure the system clock source

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_sys</b>	system clock source select
<i>RCU_CKSYSSRC_IRC8M</i>	select CK_IRC8M as the CK_SYS source
<i>RCU_CKSYSSRC_HXTAL</i>	select CK_HXTAL as the CK_SYS source
<i>RCU_CKSYSSRC_PLL</i>	select CK_PLL as the CK_SYS source
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the CK_HXTAL as the CK_SYS source */
rcu_system_clock_source_config(RCU_CKSYSSRC_HXTAL);
```

### rcu\_system\_clock\_source\_get

The description of rcu\_system\_clock\_source\_get is shown as below:

**Table 3-320. 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 3-321. Function rcu\_ahb\_clock\_config

<b>Function name</b>	rcu_ahb_clock_config
<b>Function prototype</b>	void rcu_ahb_clock_config(uint32_t ck_ahb);
<b>Function descriptions</b>	configure the AHB clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_ahb</b>	AHB clock prescaler selection
<i>RCU_AHB_CKSYS_DIVx</i>	select CK_SYS / x, (x=1, 2, 4, 8, 16, 64, 128, 256, 512)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure CK_SYS/128 */
```

```
rcu_ahb_clock_config(RCU_AHB_CKSYS_DIV128);
```

### rcu\_apb1\_clock\_config

The description of rcu\_apb1\_clock\_config is shown as below:

Table 3-322. 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_DIV4</i>	select CK_AHB/4 as CK_APB1
<i>RCU_APB1_CKAHB_DIV8</i>	select CK_AHB/8 as CK_APB1
<i>RCU_APB1_CKAHB_DIV16</i>	select CK_AHB/16 as CK_APB1
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* configure CK_AHB/16 as CK_APB1 */
rcu_apb1_clock_config(RCU_APB1_CKAHB_DIV16);
```

### rcu\_apb2\_clock\_config

The description of rcu\_apb2\_clock\_config is shown as below:

**Table 3-323. Function rcu\_apb2\_clock\_config**

Function name	rcu_apb2_clock_config
Function prototype	void rcu_apb2_clock_config(uint32_t ck_apb2);
Function descriptions	configure the APB2 clock prescaler selection
Precondition	-
The called functions	-
Input parameter{in}	
ck_apb2	APB2 clock prescaler selection
RCU_APB2_CKAHB_DIV1	select CK_AHB as CK_APB2
RCU_APB2_CKAHB_DIV2	select CK_AHB/2 as CK_APB2
RCU_APB2_CKAHB_DIV4	select CK_AHB/4 as CK_APB2
RCU_APB2_CKAHB_DIV8	select CK_AHB/8 as CK_APB2
RCU_APB2_CKAHB_DIV16	select CK_AHB/16 as CK_APB2
Output parameter{out}	
-	-
Return value	
-	-

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 3-324. Function rcu\_ckout0\_config**

Function name	rcu_ckout0_config
---------------	-------------------

<b>Function prototype</b>	void rcu_ckout0_config(uint32_t ckout0_src);
<b>Function descriptions</b>	configure the CK_OUT0 clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ckout0_src</b>	CK_OUT0 clock source selection
<i>RCU_CKOUT0SRC_NONE</i>	no clock selected
<i>RCU_CKOUT0SRC_CKSYS</i>	select system clock CK_SYS
<i>RCU_CKOUT0SRC_IRC8M</i>	select high speed 8M internal oscillator clock
<i>RCU_CKOUT0SRC_HXTAL</i>	select HXTAL
<i>RCU_CKOUT0SRC_CKPLL_DIV2</i>	select (CK_PLL / 2) clock
<i>RCU_CKOUT0SRC_CKPLL1</i>	select CK_PLL1 clock
<i>RCU_CKOUT0SRC_CKPLL2_DIV2</i>	select (CK_PLL2 / 2) clock
<i>RCU_CKOUT0SRC_CKPLL2</i>	select CK_PLL2 clock
<i>RCU_CKOUT0SRC_EXT1</i>	select EXT1 clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the HXTAL as CK_OUT0 clock source */
```

```
rcu_ckout0_config(RCU_CKOUT0SRC_HXTAL);
```

## rcu\_pll\_config

The description of rcu\_pll\_config is shown as below:

**Table 3-325. 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>	-

Input parameter{in}	
<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</i>	HXTAL is selected as source clock of PLL
Input parameter{in}	
<b>pll_mul</b>	PLL clock multiplication factor
<i>RCU_PLL_MULx</i>	PLL clock * x (x = 2..14, 6.5, 16..32)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PLL */
```

```
rcu_pll_config(RCU_PLLSRC_HXTAL, RCU_PLL_MUL10);
```

### rcu\_predv0\_config

The description of rcu\_predv0\_config is shown as below:

**Table 3-326. 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>	-
Input parameter{in}	
<b>predv0_source</b>	PREDV0 input clock source selection
<i>RCU_PREDV0SRC_HXTAL</i>	select HXTAL as PREDV0 input source clock
<i>RCU_PREDV0SRC_CKPLL1</i>	select CK_PLL1 as PREDV0 input source clock
Input parameter{in}	
<b>predv0_div</b>	PREDV0 division factor
<i>RCU_PREDV0_DIVx</i>	PREDV0 input source clock is divided x (x=1..16)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PREDV0 division factor */
```



```
rcu_predv0_config(RCU_PREDV0SRC_HXTAL, RCU_PREDV0_DIV4);
```

### rcu\_predv1\_config

The description of rcu\_predv1\_config is shown as below:

**Table 3-327. Function rcu\_predv1\_config**

<b>Function name</b>	rcu_predv1_config
<b>Function prototype</b>	void rcu_predv1_config(uint32_t predv1_div);
<b>Function descriptions</b>	configure the PREDV1 division factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>predv1_div</b>	PREDV1 division factor
<i>RCU_PREDV1_DIVx</i>	PREDV1 input source clock is divided x (x=1..16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PREDV1 division factor */
```

```
rcu_predv1_config(RCU_PREDV1_DIV8);
```

### rcu\_pll1\_config

The description of rcu\_pll1\_config is shown as below:

**Table 3-328. 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..16,20)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL1 clock */
```

```
rcu_pll1_config(RCU_PLL1_MUL8);
```

## rcu\_pll2\_config

The description of rcu\_pll2\_config is shown as below:

**Table 3-329. 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.. 16,20)
<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 3-330. Function rcu\_adc\_clock\_config**

<b>Function name</b>	rcu_adc_clock_config
<b>Function prototype</b>	void rcu_adc_clock_config(uint32_t adc_psc);
<b>Function descriptions</b>	configure the ADC prescaler factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_psc</b>	ADC prescaler factor
<i>RCU_CKADC_CKAPB2_DIV2</i>	CK_ADC = CK_APB2 / 2
<i>RCU_CKADC_CKAPB2_DIV4</i>	CK_ADC = CK_APB2 / 4
<i>RCU_CKADC_CKAPB2_DIV6</i>	CK_ADC = CK_APB2 / 6
<i>RCU_CKADC_CKAPB2_DIV8</i>	CK_ADC = CK_APB2 / 8

<i>RCU_CKADC_CKAPB2_DIV12</i>	$CK\_ADC = CK\_APB2 / 12$
<i>RCU_CKADC_CKAPB2_DIV16</i>	$CK\_ADC = CK\_APB2 / 16$
<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 3-331. Function rcu\_usb\_clock\_config**

<b>Function name</b>	rcu_usb_clock_config
<b>Function prototype</b>	void rcu_usb_clock_config(uint32_t usb_psc);
<b>Function descriptions</b>	configure the USB prescaler factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usb_psc</b>	USB prescaler factor
<i>RCU_CKUSB_CKPLL_DIV1_5</i>	$CK\_USBFS = CK\_PLL / 1.5$
<i>RCU_CKUSB_CKPLL_DIV1</i>	$CK\_USBFS = CK\_PLL / 1$
<i>RCU_CKUSB_CKPLL_DIV2_5</i>	$CK\_USBFS = CK\_PLL / 2.5$
<i>RCU_CKUSB_CKPLL_DIV2</i>	$CK\_USBFS = CK\_PLL / 2$
<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 3-332. Function rcu\_rtc\_clock\_config**

<b>Function name</b>	rcu_rtc_clock_config
<b>Function prototype</b>	void rcu_rtc_clock_config(uint32_t rtc_clock_source);
<b>Function descriptions</b>	configure the RTC clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_clock_source</b>	RTC clock source selection
<i>RCU_RTCSRC_NONE</i>	no clock selected
<i>RCU_RTCSRC_LXTAL</i>	select CK_LXTAL as RTC source clock
<i>RCU_RTCSRC_IRC40K</i>	select CK_IRC40K as RTC source clock
<i>RCU_RTCSRC_HXTAL_DIV_128</i>	select CK_HXTAL/128 as RTC source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the RTC clock source selection */
rcu_rtc_clock_config(RCU_RTCSRC_IRC40K);
```

## rcu\_i2s1\_clock\_config

The description of rcu\_i2s1\_clock\_config is shown as below:

**Table 3-333. Function rcu\_i2s1\_clock\_config**

<b>Function name</b>	rcu_i2s1_clock_config
<b>Function prototype</b>	void rcu_i2s1_clock_config(uint32_t i2s_clock_source);
<b>Function descriptions</b>	configure the I2S1 clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2s_clock_source</b>	I2S clock source selection
<i>RCU_I2S1SRC_CKSYS</i>	select system clock as I2S1 source clock
<i>RCU_I2S1SRC_CKPLL2_MUL2</i>	select CK_PLL2 * 2 as I2S1 source clock
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* configure the I2S1 clock source selection */
```

```
rcu_i2s1_clock_config(RCU_I2S1SRC_CKPLL2_MUL2);
```

### rcu\_i2s2\_clock\_config

The description of rcu\_i2s2\_clock\_config is shown as below:

**Table 3-334. Function rcu\_i2s2\_clock\_config**

<b>Function name</b>	rcu_i2s2_clock_config
<b>Function prototype</b>	void rcu_i2s2_clock_config(uint32_t i2s_clock_source);
<b>Function descriptions</b>	configure the I2S2 clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2s_clock_source</b>	I2S clock source selection
<i>RCU_I2S2SRC_CKSYS</i>	select system clock as I2S2 source clock
<i>RCU_I2S2SRC_CKPLL2_MUL2</i>	select CK_PLL2 * 2 as I2S2 source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the I2S2 clock source selection */
```

```
rcu_i2s2_clock_config(RCU_I2S2SRC_CKPLL2_MUL2);
```

### rcu\_flag\_get

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

**Table 3-335. Function rcu\_flag\_get**

<b>Function name</b>	rcu_flag_get
<b>Function prototype</b>	FlagStatus rcu_flag_get(rcu_flag_enum flag);
<b>Function descriptions</b>	get the clock stabilization and peripheral reset flags
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>flag</b>	the clock stabilization and peripheral reset flags, refer to rcu_flag_enum
<i>RCU_FLAG_IRC8MST</i> <i>B</i>	IRC8M stabilization flag
<i>RCU_FLAG_HXTALST</i> <i>B</i>	HXTAL stabilization flag
<i>RCU_FLAG_PLLSTB</i>	PLL stabilization flag
<i>RCU_FLAG_PLL1STB</i>	PLL1 stabilization flag
<i>RCU_FLAG_PLL2STB</i>	PLL2 stabilization flag
<i>RCU_FLAG_LXTALST</i> <i>B</i>	LXTAL stabilization flag
<i>RCU_FLAG_IRC40KST</i> <i>B</i>	IRC40K stabilization flag
<i>RCU_FLAG_EPRST</i>	external PIN reset flag
<i>RCU_FLAG_PORRST</i>	power reset flag
<i>RCU_FLAG_SWRST</i>	software reset flag
<i>RCU_FLAG_FWDGTR</i> <i>ST</i>	free watchdog timer reset flag
<i>RCU_FLAG_WWDGTR</i> <i>ST</i>	window watchdog timer reset flag
<i>RCU_FLAG_LPRST</i>	low-power reset flag
Output parameter{out}	
-	-
Return value	
<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 3-336. 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>	-
Input parameter{in}	

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

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 3-337. 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>	-
Input parameter{in}	
<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_LX TALSTB</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
<i>RCU_INT_FLAG_PLL2STB</i>	PLL2 stabilization interrupt flag
<i>RCU_INT_FLAG_CKM</i>	HXTAL clock stuck interrupt flag
Output parameter{out}	
-	-
Return value	
<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 3-338. Function rcu\_interrupt\_flag\_clear**

Function name	rcu_interrupt_flag_clear
Function prototype	void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag)
Function descriptions	clear the interrupt flags
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	clock stabilization and stuck interrupt flags clear, refer to rcu_int_flag_clear_enum
RCU_INT_FLAG_IRC40KSTB_CLR	IRC40K stabilization interrupt flag clear
RCU_INT_FLAG_LXTALSTB_CLR	LXTAL stabilization interrupt flag clear
RCU_INT_FLAG_IRC8MSTB_CLR	IRC8M stabilization interrupt flag clear
RCU_INT_FLAG_HXTALSTB_CLR	HXTAL stabilization interrupt flag clear
RCU_INT_FLAG_PLLSTB_CLR	PLL stabilization interrupt flag clear
RCU_INT_FLAG_PLL1STB_CLR	PLL1 stabilization interrupt flag clear
RCU_INT_FLAG_PLL2STB_CLR	PLL2 stabilization interrupt flag clear
RCU_INT_FLAG_CKM_CLR	clock stuck interrupt flag clear
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* clear the interrupt HXTAL stabilization interrupt flag */

rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);

```



## rcu\_interrupt\_enable

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

**Table 3-339. 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
<i>RCU_INT_PLL2STB</i>	PLL2 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 3-340. Function rcu\_interrupt\_disable**

<b>Function name</b>	rcu_interrupt_disable
<b>Function prototype</b>	void rcu_interrupt_disable(rcu_int_enum interrupt);
<b>Function descriptions</b>	disable the stabilization interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	clock stabilization interrupt, refer to 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
<i>RCU_INT_PLL2STB</i>	PLL2 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\_osci\_stab\_wait

The description of rcu\_osci\_stab\_wait is shown as below:

**Table 3-341. Function rcu\_osci\_stab\_wait**

<b>Function name</b>	rcu_osci_stab_wait
<b>Function prototype</b>	ErrStatus rcu_osci_stab_wait(rcu_osci_type_enum osci);
<b>Function descriptions</b>	wait for oscillator stabilization flags is SET or oscillator startup is timeout
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to rcu_osci_type_enum
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<i>RCU_IRC8M</i>	internal 8M RC oscillators(IRC8M)
<i>RCU_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
<i>RCU_PLL2_CK</i>	phase locked loop 2
<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 3-342. 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 rcu_osci_type_enum
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<i>RCU_IRC8M</i>	internal 8M RC oscillators(IRC8M)
<i>RCU_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
<i>RCU_PLL2_CK</i>	phase locked loop 2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn on the high speed crystal oscillator */
```

```
rcu_osci_on(RCU_HXTAL);
```

## rcu\_osci\_off

The description of rcu\_osci\_off is shown as below:

**Table 3-343. Function rcu\_osci\_off**

<b>Function name</b>	rcu_osci_off
<b>Function prototype</b>	void rcu_osci_off(rcu_osci_type_enum osci);
<b>Function descriptions</b>	turn off the oscillator
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to rcu_osci_type_enum
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<i>RCU_IRC8M</i>	internal 8M RC oscillators(IRC8M)
<i>RCU_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
<i>RCU_PLL2_CK</i>	phase locked loop 2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn off the high speed crystal oscillator */
```

```
rcu_osci_off(RCU_HXTAL);
```

### rcu\_osci\_bypass\_mode\_enable

The description of rcu\_osci\_bypass\_mode\_enable is shown as below:

**Table 3-344. Function rcu\_osci\_bypass\_mode\_enable**

<b>Function name</b>	rcu_osci_bypass_mode_enable
<b>Function prototype</b>	void rcu_osci_bypass_mode_enable(rcu_osci_type_enum osci);
<b>Function descriptions</b>	enable the oscillator bypass mode
<b>Precondition</b>	HXTALEN or LXTALEN must be reset before it
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to rcu_osci_type_enum
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_enable(RCU_HXTAL);
```

### rcu\_osci\_bypass\_mode\_disable

The description of rcu\_osci\_bypass\_mode\_disable is shown as below:

**Table 3-345. Function rcu\_osci\_bypass\_mode\_disable**

<b>Function name</b>	rcu_osci_bypass_mode_disable
<b>Function prototype</b>	void rcu_osci_bypass_mode_disable(rcu_osci_type_enum osci);
<b>Function descriptions</b>	disable the oscillator bypass mode

<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_osci_type_enum
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_disable(RCU_HXTAL);
```

### rcu\_hxtal\_clock\_monitor\_enable

The description of rcu\_hxtal\_clock\_monitor\_enable is shown as below:

**Table 3-346. 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 3-347. 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 3-348. Function rcu\_irc8m\_adjust\_value\_set**

<b>Function name</b>	rcu_irc8m_adjust_value_set
<b>Function prototype</b>	void rcu_irc8m_adjust_value_set(uint32_t irc8m_adjval);
<b>Function descriptions</b>	set the IRC8M adjust value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
irc8m_adjval	IRC8M adjust value, must be between 0 and 0x1F
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-349. Function rcu\_deepsleep\_voltage\_set**

<b>Function name</b>	rcu_deepsleep_voltage_set
<b>Function prototype</b>	void rcu_deepsleep_voltage_set(uint32_t dsvol);
<b>Function descriptions</b>	set the deep-sleep mode voltage value

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dsvol</b>	deep sleep mode voltage
<i>RCU_DEEPSLEEP_V_1_2</i>	the core voltage is 1.2V in deep-sleep mode
<i>RCU_DEEPSLEEP_V_1_1</i>	the core voltage is 1.1V in deep-sleep mode
<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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-350. 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
<i>CK_SYS</i>	system clock frequency
<i>CK_AHB</i>	AHB clock frequency
<i>CK_APB1</i>	APB1 clock frequency
<i>CK_APB2</i>	APB2 clock frequency
<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);
```

## 3.18. 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 [3.18.1](#), the FWDGT firmware functions are introduced in chapter [3.18.2](#).

### 3.18.1. Descriptions of Peripheral registers

RTC registers are listed in the table shown as below:

**Table 3-351. 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

### 3.18.2. Descriptions of Peripheral functions

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

**Table 3-352. RTC firmware function**

Function name	Function description
rtc_configuration_mode_enter	enter RTC configuration mode
rtc_configuration_mode_exit	exit RTC configuration mode
rtc_counter_set	set RTC counter value
rtc_prescaler_set	set RTC prescaler value
rtc_lwoff_wait	wait RTC last write operation finished flag set
rtc_register_sync_wait	wait RTC registers synchronized flag set
rtc_alarm_config	set RTC alarm value
rtc_counter_get	get RTC counter value



Function name	Function description
rtc_divider_get	get RTC divider value
rtc_flag_get	get RTC flag status
rtc_flag_clear	clear RTC flag status
rtc_interrupt_flag_get	get RTC interrupt flag status
rtc_interrupt_flag_clear	clear RTC interrupt flag status
rtc_interrupt_enable	enable RTC interrupt
rtc_interrupt_disable	disable RTC interrupt

### rtc\_configuration\_mode\_enter

The description of rtc\_configuration\_mode\_enter is shown as below:

**Table 3-353. 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>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 3-354. 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>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* exit RTC configuration mode */
rtc_configuration_mode_exit ( );
```

### rtc\_counter\_set

The description of rtc\_counter\_set is shown as below:

**Table 3-355. 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>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 3-356. Function rtc\_prescaler\_set**

<b>Function name</b>	rtc_interrupt_rtc_prescaler_set
<b>Function prototype</b>	void rtc_prescaler_set(uint32_t psc);
<b>Function descriptions</b>	set RTC prescaler value
<b>Precondition</b>	before using this function, you must call rtc_lwoff_wait ( ) function (wait until LWOFF flag is set).
<b>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\_lwoff\_wait

The description of rtc\_lwoff\_wait is shown as below:

**Table 3-357. 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>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 3-358. 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>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* wait for RTC registers synchronization */
```

```
rtc_register_sync_wait( );
```

### rtc\_alarm\_config

The description of rtc\_alarm\_config is shown as below:

**Table 3-359. Function rtc\_alarm\_config**

<b>Function name</b>	rtc_alarm_config
<b>Function prototype</b>	void rtc_alarm_config(uint32_t alarm);
<b>Function descriptions</b>	set RTC alarm value
<b>Precondition</b>	before using this function, you must call rtc_lwoff_wait ( ) function (wait until LWOFF flag is set)
<b>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\_counter\_get

The description of rtc\_counter\_get is shown as below:

**Table 3-360. 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>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

Return value	
<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\_divider\_get

The description of rtc\_divider\_get is shown as below:

**Table 3-361. 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>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\_getrtc\_interrupt\_enable is shown as below:

**Table 3-362. 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>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 3-363. 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>Input parameter{in}</b>	
<b>flag</b>	specify which RTC flag status to clear
<i>RTC_FLAG_SECOND</i>	second interrupt flag
<i>RTC_FLAG_ALARM</i>	alarm interrupt flag
<i>RTC_FLAG_OVERFLOW</i> <i>W</i>	overflow interrupt flag
<i>RTC_FLAG_RSYN</i>	registers synchronized flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the RTC alarm flag */
```

```
rtc_flag_clear (RTC_FLAG_ALARM);
```

### rtc\_interrupt\_flag\_get

The description of rtc\_interrupt\_flag\_getrtc\_interrupt\_enable is shown as below:

Table 3-364. Function rtc\_interrupt\_flag\_get

<b>Function name</b>	rtc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus rtc_interrupt_flag_get(uint32_t flag);
<b>Function descriptions</b>	get RTC interrupt flag status
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which RTC interrupt flag status to get
<i>RTC_INT_FLAG_SEC OND</i>	second interrupt flag
<i>RTC_INT_FLAG_ALAR M</i>	alarm interrupt flag
<i>RTC_INT_FLAG_OVE RFLOW</i>	overflow interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the RTC alarm interrupt status */
```

```
FlagStatus alarm_status;
```

```
alarm_status = rtc_interrupt_flag_get (RTC_INT_FLAG_ALARM);
```

### rtc\_interrupt\_flag\_clear

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

Table 3-365. Function rtc\_interrupt\_flag\_clear

<b>Function name</b>	rtc_interrupt_flag_clear
<b>Function prototype</b>	void rtc_interrupt_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear RTC interrupt flag status
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which RTC interrupt flag status to clear
<i>RTC_INT_FLAG_SEC OND</i>	second interrupt flag
<i>RTC_INT_FLAG_ALAR M</i>	alarm interrupt flag
<i>RTC_INT_FLAG_OVE RFLOW</i>	overflow interrupt flag
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* clear the RTC alarm interrupt flag */
```

```
rtc_interrupt_flag_clear (RTC_INT_FLAG_ALARM);
```

### rtc\_interrupt\_enable

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

**Table 3-366. Function rtc\_interrupt\_enable**

<b>Function name</b>	rtc_interrupt_enable
<b>Function prototype</b>	void rtc_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable RTC interrupt
<b>Precondition</b>	before using this function, you must call rtc_lwoff_wait ( ) function (wait until LWOFF flag is set)
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which RTC interrupt to enable
<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( );
```

```
/* 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 3-367. Function rtc\_interrupt\_disable**

<b>Function name</b>	rtc_interrupt_disable
<b>Function prototype</b>	void rtc_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable RTC interrupt
<b>Precondition</b>	before using this function, you must call rtc_lwoff_wait ( ) function (wait until LWOFF flag is set)



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

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);
```

## 3.19. 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 [3.19.1](#), the SPI/I2S firmware functions are introduced in chapter [3.19.2](#).

### 3.19.1. Descriptions of Peripheral registers

SPI/I2S registers are listed in the table shown as below:

**Table 3-368. 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

### 3.19.2. Descriptions of Peripheral functions

SPI/I2S firmware functions are listed in the table shown as below:

Table 3-369. 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_bidirectional_transfer_config	configure SPI bidirectional transfer direction
spi_i2s_data_transmit	SPI transmit data
spi_i2s_data_receive	SPI receive data
spi_i2s_format_error_clear	clear SPI/I2S format error flag status
spi_crc_polynomial_set	set SPI CRC polynomial
spi_crc_polynomial_get	get SPI CRC polynomial
spi_crc_on	turn on SPI CRC function
spi_crc_off	turn off SPI CRC function
spi_crc_next	SPI next data is CRC value
spi_crc_get	get SPI CRC send value or receive value
spi_crc_error_clear	clear SPI CRC error flag status
spi_ti_mode_enable	enable SPI TI mode
spi_ti_mode_disable	disable SPI TI mode
spi_nssp_mode_enable	enable SPI NSS pulse mode
spi_nssp_mode_disable	disable SPI NSS pulse mode
spi_i2s_flag_get	get SPI and I2S flag status
spi_i2s_interrupt_enable	enable SPI and I2S interrupt
spi_i2s_interrupt_disable	disable SPI and I2S interrupt
spi_i2s_interrupt_flag_get	get SPI and I2S interrupt status

### Structure spi\_parameter\_struct

Table 3-370. Structure spi\_parameter\_struct

Member name	Function description
device_mode	SPI master or slave

Member name	Function description
	(SPI_MASTER, SPI_SLAVE)
trans_mode	SPI transtype (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	SPI frame size (SPI_FRAME_SIZE_xBIT, x=8/16)
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 3-371. 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 SPIx and I2Sx peripheral
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI/I2S peripheral
<i>SPIx</i>	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 3-372. Function spi\_i2s\_deinit

Function name	spi_struct_para_init
Function prototype	void spi_struct_para_init(spi_parameter_struct* spi_struct);
Function descriptions	Initialize the parameters of SPI struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
spi_struct	SPI init parameter struct, the structure members can refer to <a href="#">Table 3-370. Structure spi_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

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 3-373. Function spi\_init

Function name	spi_init
Function prototype	void spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);
Function descriptions	Initialize SPIx peripheral parameter
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Input parameter{in}	
spi_struct	SPI parameter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-370. Structure spi_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize SPI0 */

```

```

spi_parameter_struct spi_init_struct;

spi_init_struct.trans_mode          = SPI_TRANSMODE_BDTRANSMIT;

spi_init_struct.device_mode         = SPI_MASTER;

spi_init_struct.frame_size          = SPI_FRAME_SIZE_8BIT;

spi_init_struct.clock_polarity_phase = SPI_CLOCK_POLARITY_HIGH_PHASE_2EDGE;

spi_init_struct.nss                 = SPI_NSS_SOFT;

spi_init_struct.prescale            = SPI_PSC_8;

spi_init_struct.endian              = SPI_ENDIAN_MSB;

spi_init(SPI0, &spi_init_struct);

```

## spi\_enable

The description of spi\_enable is shown as below:

**Table 3-374. 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 SPIx
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* enable SPI0 */

spi_enable(SPI0);

```

## spi\_disable

The description of spi\_disable is shown as below:

**Table 3-375. 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 SPIx

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 */
spi_disable(SPI0);
```

### i2s\_init

The description of i2s\_init is shown as below:

**Table 3-376. Function i2s\_init**

<b>Function name</b>	i2s_init
<b>Function prototype</b>	void i2s_init(uint32_t spi_periph, uint32_t mode, uint32_t standard, uint32_t ckpl);
<b>Function descriptions</b>	Initialize I2Sx peripheral parameter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2Sx peripheral
<i>SPIx</i>	x=1,2
<b>Input parameter{in}</b>	
<b>mode</b>	I2S operation mode
<i>I2S_MODE_SLAVETX</i>	I2S slave transmit mode
<i>I2S_MODE_SLAVERX</i>	I2S slave receive mode
<i>I2S_MODE_MASTERTX</i>	I2S master transmit mode
<i>I2S_MODE_MASTERRX</i>	I2S master receive mode
<b>Input parameter{in}</b>	
<b>standard</b>	I2S standard
<i>I2S_STD_PHILLIPS</i>	I2S phillips standard
<i>I2S_STD_MSB</i>	I2S MSB standard
<i>I2S_STD_LSB</i>	I2S LSB standard
<i>I2S_STD_PCMSHORT</i>	I2S PCM short standard
<i>I2S_STD_PCMLONG</i>	I2S PCM long standard

Input parameter{in}	
<b>ckpl</b>	I2S idle state clock polarity
<i>I2S_CKPL_LOW</i>	I2S clock polarity low level
<i>I2S_CKPL_HIGH</i>	I2S clock polarity high level
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize I2S1 */
```

```
i2s_init(SPI1, I2S_MODE_MASTERTX, I2S_STD_PHILLIPS, I2S_CKPL_LOW);
```

### i2s\_psc\_config

The description of i2s\_psc\_config is shown as below:

**Table 3-377. 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 audiosample, uint32_t frameformat, uint32_t mckout);
<b>Function descriptions</b>	Configure I2Sx prescaler
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get
Input parameter{in}	
<b>spi_periph</b>	I2Sx peripheral
<i>SPIx</i>	x=1,2
Input parameter{in}	
<b>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_9</i> 6K	audio sample rate is 96KHz
<i>I2S_AUDIOSAMPLE_1</i> 92K	audio sample rate is 192KHz
<b>Input parameter{in}</b>	
<b>frameformat</b>	I2S data length and channel length
<i>I2S_FRAMEFORMAT_DT16B_CH16B</i>	I2S data length is 16 bit and channel length is 16 bit
<i>I2S_FRAMEFORMAT_DT16B_CH32B</i>	I2S data length is 16 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT24B_CH32B</i>	I2S data length is 24 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT32B_CH32B</i>	I2S data length is 32 bit and channel length is 32 bit
<b>Input parameter{in}</b>	
<b>mckout</b>	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 3-378. 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 I2Sx
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2Sx peripheral
<i>SPIx</i>	x=1,2



Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2S1*/
i2s_enable(SPI1);
```

### i2s\_disable

The description of i2s\_disable is shown as below:

**Table 3-379. Function i2s\_disable**

Function name	i2s_disable
Function prototype	void i2s_disable(uint32_t spi_periph);
Function descriptions	Disable I2Sx
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	I2Sx peripheral
SPIx	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2S1 */
i2s_disable(SPI1);
```

### spi\_nss\_output\_enable

The description of spi\_nss\_output\_enable is shown as below:

**Table 3-380. Function spi\_nss\_output\_enable**

Function name	spi_nss_output_enable
Function prototype	void spi_nss_output_enable(uint32_t spi_periph);
Function descriptions	Enable SPIx NSS output function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPIx peripheral

<i>SPIx</i>	<i>x</i> =0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 NSS output */
spi_nss_output_enable(SPI0);
```

### spi\_nss\_output\_disable

The description of spi\_nss\_output\_disable is shown as below:

**Table 3-381. 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 SPIx NSS output function
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>spi_periph</b>	SPIx peripheral
<i>SPIx</i>	<i>x</i> =0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 NSS output */
spi_nss_output_disable(SPI0);
```

### spi\_nss\_internal\_high

The description of spi\_nss\_internal\_high is shown as below:

**Table 3-382. 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>	-
Input parameter{in}	

<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 NSS pin is pulled high level in software mode */
```

```
spi_nss_internal_high(SPI0);
```

### spi\_nss\_internal\_low

The description of spi\_nss\_internal\_low is shown as below:

**Table 3-383. Function spi\_nss\_internal\_low**

<b>Function name</b>	spi_nss_internal_low
<b>Function prototype</b>	void spi_nss_internal_low(uint32_t spi_periph);
<b>Function descriptions</b>	SPI NSS pin low level in software mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 NSS pin is pulled low level in software mode */
```

```
spi_nss_internal_low(SPI0);
```

### spi\_dma\_enable

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

**Table 3-384. 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 SPIx 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 3-385. 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 SPIx 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:

```
/* 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 3-386. Function spi\_i2s\_data\_frame\_format\_config**

<b>Function name</b>	spi_i2s_data_frame_format_config
<b>Function prototype</b>	void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
<b>Function descriptions</b>	Configure SPIx/I2Sx data frame format
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>frame_format</b>	SPI frame size
<i>SPI_FRAME_SIZE_xBIT</i>	SPI frame size is x bits, x=8/16
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SPI1/I2S1 data frame format size is 16 bits */
```

```
spi_i2s_data_frame_format_config(SPI1, SPI_FRAME_SIZE_16BIT);
```

## spi\_bidirectional\_transfer\_config

The description of spi\_bidirectional\_transfer\_config is shown as below:

**Table 3-387. 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 SPIx bidirectional transfer direction
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>transfer_direction</b>	SPI transfer direction
<i>SPI_BIDIRECTIONAL_TRANSMIT</i>	SPI work in transmit-only mode

<i>SPI_BIDIRECTIONAL_RECEIVE</i>	SPI work in receive-only mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

### **spi\_i2s\_data\_transmit**

The description of spi\_i2s\_data\_transmit is shown as below:

**Table 3-388. Function spi\_i2s\_data\_transmit**

<b>Function name</b>	spi_i2s_data_transmit
<b>Function prototype</b>	void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);
<b>Function descriptions</b>	SPI transmit data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>data</b>	16-bit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 transmit data */
```

```
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n]);
```

### **spi\_i2s\_data\_receive**

The description of spi\_i2s\_data\_receive is shown as below:

**Table 3-389. Function spi\_i2s\_data\_receive**

<b>Function name</b>	spi_i2s_data_receive
<b>Function prototype</b>	uint16_t spi_i2s_data_receive(uint32_t spi_periph);
<b>Function descriptions</b>	SPI receive data

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	16-bit data

Example:

```
/* SPI0 receive data */
```

```
spi0_receive_array[receive_n] = spi_i2s_data_receive(SPI0);
```

### spi\_i2s\_format\_error\_clear

The description of spi\_i2s\_format\_error\_clear is shown as below:

**Table 3-390. Function spi\_i2s\_format\_error\_clear**

<b>Function name</b>	spi_i2s_format_error_clear
<b>Function prototype</b>	void spi_i2s_format_error_clear(uint32_t spi_periph, uint32_t flag);
<b>Function descriptions</b>	clear SPI/I2S format error flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>flag</b>	SPI/I2S frame format error flag
<i>SPI_FLAG_FERR</i>	only for SPI work in TI mode
<i>I2S_FLAG_FERR</i>	for I2S
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* clear SPI0 format error flag status */
```

```
spi_crc_error_clear(SPI0, SPI_FLAG_FERR );
```

### spi\_crc\_polynomial\_set

The description of spi\_crc\_polynomial\_set is shown as below:

Table 3-391. Function spi\_crc\_polynomial\_set

Function name	spi_crc_polynomial_set
Function prototype	void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly);
Function descriptions	Set SPI CRC polynomial
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Input parameter{in}	
crc_poly	CRC polynomial value
Output parameter{out}	
-	-
Return value	
-	-

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 3-392. Function spi\_crc\_polynomial\_get

Function name	spi_crc_polynomial_get
Function prototype	uint16_t spi_crc_polynomial_get(uint32_t spi_periph);
Function descriptions	Get SPI CRC polynomial
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	
uint16_t	16 bit CRC polynomial value

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 3-393. Function spi\_crc\_on**

<b>Function name</b>	spi_crc_on
<b>Function prototype</b>	void spi_crc_on(uint32_t spi_periph);
<b>Function descriptions</b>	Turn on CRC function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn on SPI0 CRC function */
```

```
spi_crc_on(SPI0);
```

## spi\_crc\_off

The description of spi\_crc\_off is shown as below:

**Table 3-394. Function spi\_crc\_off**

<b>Function name</b>	spi_crc_off
<b>Function prototype</b>	void spi_crc_off(uint32_t spi_periph);
<b>Function descriptions</b>	Turn off CRC function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn off SPI0 CRC function */
```

```
spi_crc_off(SPI0);
```

## spi\_crc\_next

The description of spi\_crc\_next is shown as below:

**Table 3-395. Function spi\_crc\_next**

<b>Function name</b>	spi_crc_next
<b>Function prototype</b>	void spi_crc_next(uint32_t spi_periph);
<b>Function descriptions</b>	SPI next data is CRC value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 next data is CRC value */
```

```
spi_crc_next(SPI0);
```

## spi\_crc\_get

The description of spi\_crc\_get is shown as below:

**Table 3-396. Function spi\_crc\_get**

<b>Function name</b>	spi_crc_get
<b>Function prototype</b>	uint16_t spi_crc_get(uint32_t spi_periph, uint8_t crc);
<b>Function descriptions</b>	Get SPI CRC send value or receive value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<i>crc</i>	SPI crc value
<i>SPI_CRC_TX</i>	get transmit crc value
<i>SPI_CRC_RX</i>	get receive crc value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	16-bit CRC value

Example:

```
/* get SPI0 CRC send value */

uint16_t crc_val;

crc_val = spi_crc_get(SPI0,SPI_CRC_TX);
```

### spi\_crc\_error\_clear

The description of spi\_crc\_error\_clear is shown as below:

**Table 3-397. Function spi\_crc\_error\_clear**

<b>Function name</b>	spi_crc_error_clear
<b>Function prototype</b>	void spi_crc_error_clear(uint32_t spi_periph);
<b>Function descriptions</b>	Clear SPI CRC error flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* clear SPI0 CRC error flag status */

spi_crc_error_clear(SPI0);
```

### spi\_ti\_mode\_enable

The description of spi\_ti\_mode\_enable is shown as below:

**Table 3-398. Function spi\_ti\_mode\_enable**

<b>Function name</b>	spi_ti_mode_enable
<b>Function prototype</b>	void spi_ti_mode_enable(uint32_t spi_periph);
<b>Function descriptions</b>	Enable SPI TI mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-

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 3-399. Function spi\_ti\_mode\_disable**

Function name	spi_ti_mode_disable
Function prototype	void spi_ti_mode_disable(uint32_t spi_periph);
Function descriptions	Disable SPI TI mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	

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 3-400. Function spi\_nssp\_mode\_enable**

Function name	spi_nssp_mode_enable
Function prototype	void spi_nssp_mode_enable(uint32_t spi_periph);
Function descriptions	Enable SPI NSS pulse mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	

-	-
Return value	

Example:

```
/* enable SPI0 NSS pulse mode */
spi_nssp_mode_enable(SPI0);
```

### spi\_nssp\_mode\_disable

The description of spi\_nssp\_mode\_disable is shown as below:

**Table 3-401. Function spi\_nssp\_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 NSS pulse mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* disable SPI0 NSS pulse mode */
spi_nssp_mode_disable(SPI0);
```

### spi\_i2s\_flag\_get

The description of spi\_i2s\_flag\_get is shown as below:

**Table 3-402. 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, uint8_t interrupt);
<b>Function descriptions</b>	Get SPIx and I2Sx flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2

Input parameter{in}	
<b>flag</b>	SPI/I2S flag status
<i>SPI_FLAG_TBE</i>	transmit buffer empty flag
<i>SPI_FLAG_RBNE</i>	receive buffer not empty flag
<i>SPI_FLAG_TRANS</i>	transmit on-going flag
<i>SPI_FLAG_RXORERR</i>	receive overrun error flag
<i>SPI_FLAG_CONFERR</i>	mode config error flag
<i>SPI_FLAG_CRCERR</i>	CRC error flag
<i>SPI_FLAG_FERR</i>	SPI format error interrupt flag
<i>I2S_FLAG_TBE</i>	transmit buffer empty flag
<i>I2S_FLAG_RBNE</i>	receive buffer not empty flag
<i>I2S_FLAG_TRANS</i>	transmit on-going flag
<i>I2S_FLAG_RXORERR</i>	overrun error flag
<i>I2S_FLAG_TXURERR</i>	underrun error flag
<i>I2S_FLAG_CH</i>	channel side flag
<i>I2S_FLAG_FERR</i>	I2S format error interrupt flag
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get SPI0 transmit buffer empty flag status */
while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
```

### spi\_i2s\_interrupt\_enable

The description of spi\_i2s\_interrupt\_enable is shown as below:

**Table 3-403. 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 SPIx and I2Sx interrupt
<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>interrupt</b>	SPI/I2S interrupt
<i>SPI_I2SINT_TBE</i>	transmit buffer empty interrupt

<i>SPI_I2S_INT_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_ERR</i>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* enable SPI0 transmit buffer empty interrupt */
spi_i2s_interrupt_enable(SPI0, SPI_I2S_INT_TBE);
```

### spi\_i2s\_interrupt\_disable

The description of spi\_i2s\_interrupt\_disable is shown as below:

**Table 3-404. 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 SPIx and I2Sx interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>interrupt</b>	SPI/I2S interrupt
<i>SPI_I2SINT_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_ERR</i>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* disable SPI0 transmit buffer empty interrupt */
spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_TBE);
```

### spi\_i2s\_interrupt\_flag\_get

The description of spi\_i2s\_interrupt\_flag\_get is shown as below:

Table 3-405. 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 SPIx and I2Sx interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>interrupt</b>	SPI/I2S interrupt
<i>SPI_I2S_INT_FLAG_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_FLAG_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_FLAG_RXORERR</i>	overrun interrupt
<i>SPI_INT_FLAG_CONFIGERR</i>	config error interrupt
<i>SPI_INT_FLAG_CRCERR</i>	CRC error interrupt
<i>I2S_INT_FLAG_TXURERR</i>	underrun error interrupt
<i>SPI_I2S_INT_FLAG_FORMATERR</i>	format error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```

/* get SPI0 transmit buffer empty interrupt status */
if(RESET != spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE)){
    while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
    spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
}

```

## 3.20. 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 are divided into five sorts: advanced timer (TIMER0), general level0 timer (TIMER1,2,3,4), Basic timer (TIMER5,6). The specific functions of different types of timer are different. The TIMER registers are listed in chapter [3.20.1](#), the TIMER firmware functions are introduced in chapter [3.20.2](#).

### 3.20.1. Descriptions of Peripheral registers

TIMERx registers are listed in the table shown as below:

**Table 3-406. TIMERx Registers**

Registers	Descriptions
TIMER_CTL0(timerx, x=0..6)	Control register 0
TIMERx_CTL1(timerx, x=0..6)	Control register 1
TIMERx_SMCFG(timerx, x=0..4)	Slave mode configuration register
TIMERx_DMAINTEN(timerx, x=0..6)	DMA and interrupt enable register
TIMERx_INTF(timerx, x=0..6)	Interrupt flag register
TIMERx_SWEVG(timerx, x=0..6)	Software event generation register
TIMERx_CHCTL0(timerx, x=0..4)	Channel control register 0
TIMERx_CHCTL1(timerx, x=0..4)	Channel control register 1
TIMERx_CHCTL2(timerx, x=0..4)	Channel control register 2
TIMERx_CNT(timerx, x=0..6)	Counter register
TIMERx_PSC(timerx, x=0..6)	Prescaler register
TIMERx_CAR(timerx, x=0..6)	Counter auto reload register
TIMERx_CREP(timerx, x=0)	Counter repetition register
TIMERx_CH0CV(timerx, x=0..4)	Channel 0 capture/compare value register
TIMERx_CH1CV(timerx, x=0..4)	Channel 1 capture/compare value register
TIMERx_CH2CV(timerx, x=0..2)	Channel 2 capture/compare value register
TIMERx_CH3CV(timerx, x=0..4)	Channel 3 capture/compare value register
TIMERx_CCHP(timerx, x=0)	TIMER complementary channel protection register
TIMERx_DMACFG(timerx, x=0..4)	DMA configuration register
TIMERx_DMATB(timerx, x=0..4)	DMA transfer buffer register

### 3.20.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

**Table 3-407. 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

Function name	Function description
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
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_	configure TIMER channel output shadow function

Function name	Function description
config	
timer_channel_output_fast_config	configure TIMER channel output fast function
timer_channel_output_clear_config	configure TIMER channel output clear function
timer_channel_output_polarity_config	configure TIMER channel output polarity
timer_channel_complementary_output_polarity_config	configure TIMER channel complementary output polarity
timer_channel_output_state_config	configure TIMER channel enable state
timer_channel_complementary_output_state_config	configure TIMER channel complementary output enable state
timer_channel_input_struct_para_init	initialize the parameters of TIMER channel input parameter struct with the default values
timer_input_capture_config	configure TIMER input capture parameter
timer_channel_input_capture_prescaler_config	configure TIMER channel input capture prescaler value
timer_channel_capture_value_register_read	read TIMER channel capture compare register value
timer_input_pwm_capture_config	configure TIMER input pwm capture function
timer_hall_mode_config	configure TIMER hall sensor mode
timer_input_trigger_source_select	select TIMER input trigger source
timer_master_output_trigger_source_select	select TIMER master mode output trigger source
timer_slave_mode_select	select TIMER slave mode
timer_master_slave_mode_config	configure TIMER master slave mode
timer_external_trigger_config	configure TIMER external trigger input
timer_quadrature_decoder_mode_config	configure TIMER quadrature decoder mode
timer_internal_clock_config	configure TIMER internal clock mode
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_interrupt_enable	enable the TIMER interrupt
timer_interrupt_disable	disable the TIMER interrupt
timer_interrupt_flag_get	get timer interrupt flag
timer_interrupt_flag_clear	clear TIMER interrupt flag
timer_flag_get	get TIMER flags

Function name	Function description
timer_flag_clear	clear TIMER flags

### Structure timer\_parameter\_struct

**Table 3-408. 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 3-409. 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 3-410. 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,

Member name	Function description
	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 3-411. 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 3-412. Function timer\_deinit**

<b>Function name</b>	timer_deinit
<b>Function prototype</b>	void timer_deinit(uint32_t timer_periph);
<b>Function descriptions</b>	deinit a TIMER
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..6)</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 3-413. Function timer\_struct\_para\_init

Function name	timer_struct_para_init
Function prototype	void timer_struct_para_init(timer_parameter_struct* initpara);
Function descriptions	initialize the parameters of TIMER init parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
initpara	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-408. Structure timer_parameter_struct.</a>
Output parameter{out}	
-	-
Return value	
-	-

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 3-414. Function timer\_init

Function name	timer_init
Function prototype	void timer_init(uint32_t timer_periph, timer_parameter_struct* initpara);
Function descriptions	initialize TIMER counter
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..6)	TIMER peripheral selection
Input parameter{in}	
initpara	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-408. Structure timer_parameter_struct.</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER0 */
```

```

timer_parameter_struct timer_initpara;

timer_initpara.prescaler      = 107;

timer_initpara.alignedmode    = TIMER_COUNTER_EDGE;

timer_initpara.counterdirection = TIMER_COUNTER_UP;

timer_initpara.period         = 999;

timer_initpara.clockdivision  = TIMER_CKDIV_DIV1;

timer_initpara.repetitioncounter = 1;

timer_init(TIMERO,&timer_initpara);

```

### timer\_enable

The description of timer\_enable is shown as below:

**Table 3-415. Function timer\_enable**

<b>Function name</b>	timer_enable
<b>Function prototype</b>	void timer_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable a timer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..6)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* enable TIMERO */

timer_enable (TIMERO);

```

### timer\_disable

The description of timer\_disable is shown as below:

**Table 3-416. 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>	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..6)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 */
```

```
timer_disable (TIMER0);
```

### timer\_auto\_reload\_shadow\_enable

The description of timer\_auto\_reload\_shadow\_enable is shown as below:

**Table 3-417. 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>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..6)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_enable (TIMER0);
```

### timer\_auto\_reload\_shadow\_disable

The description of timer\_auto\_reload\_shadow\_disable is shown as below:

**Table 3-418. 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>	-



The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..6)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_disable (TIMER0);
```

### timer\_update\_event\_enable

The description of timer\_update\_event\_enable is shown as below:

**Table 3-419. Function timer\_update\_event\_enable**

Function name	timer_update_event_enable
Function prototype	void timer_update_event_enable(uint32_t timer_periph);
Function descriptions	enable the update event
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..6)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TIMER0 the update event */
```

```
timer_update_event_enable (TIMER0);
```

### timer\_update\_event\_disable

The description of timer\_update\_event\_disable is shown as below:

**Table 3-420. Function timer\_update\_event\_disable**

Function name	timer_update_event_disable
Function prototype	void timer_update_event_disable (uint32_t timer_periph);
Function descriptions	disable the update event

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..6)</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 3-421. 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)</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_COUNTER_DOWN</i>	Center-aligned and counting down assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in <i>TIMERx_CHCTL0</i> register). Only when the counter is counting down, compare interrupt flag of channels can be set.
<i>TIMER_COUNTER_COUNTER_UP</i>	Center-aligned and counting up assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in <i>TIMERx_CHCTL0</i> register). Only when the counter is counting up, compare interrupt flag of channels can be set.
<i>TIMER_COUNTER_COUNTER_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 <i>TIMERx_CHCTL0</i> register). Both when the counter is counting up and counting down, compare interrupt flag of channels can be set.

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set TIMER0 counter center-aligned and counting up assert mode */
```

```
timer_counter_alignment (TIMER0, TIMER_COUNTER_CENTER_UP);
```

### timer\_counter\_up\_direction

The description of timer\_counter\_up\_direction is shown as below:

**Table 3-422. Function timer\_counter\_up\_direction**

Function name	timer_counter_up_direction
Function prototype	void timer_counter_up_direction(uint32_t timer_periph);
Function descriptions	set TIMER counter up direction
Precondition	set TIMER counter no center-aligned mode (edge-aligned mode)
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

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 3-423. Function timer\_counter\_down\_direction**

Function name	timer_counter_down_direction
Function prototype	void timer_counter_down_direction(uint32_t timer_periph);
Function descriptions	set TIMER counter down direction
Precondition	set TIMER counter no center-aligned mode (edge-aligned mode)
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral

<i>TIMERx</i> ( <i>x</i> =0..4)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

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 3-424. Function timer\_prescaler\_config**

<b>Function name</b>	timer_prescaler_config
<b>Function prototype</b>	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint8_t pscreload);
<b>Function descriptions</b>	configure TIMER prescaler
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i> ( <i>x</i> =0..6)	TIMER peripheral selection
Input parameter{in}	
<b>prescaler</b>	prescaler value (0~65535)
Input parameter{in}	
pscreload	prescaler reload mode
<i>TIMER_PSC_RELOAD_NOW</i>	the prescaler is loaded right now
<i>TIMER_PSC_RELOAD_UPDATE</i>	the prescaler is loaded at the next update event
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 prescaler */
```

```
timer_prescaler_config (TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

## timer\_repetition\_value\_config

The description of timer\_repetition\_value\_config is shown as below:

**Table 3-425. Function timer\_repetition\_value\_config**

<b>Function name</b>	timer_repetition_value_config
<b>Function prototype</b>	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
<b>Function descriptions</b>	configure TIMER repetition register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>repetition</b>	the counter repetition value (0~255)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 repetition register value */
```

```
timer_repetition_value_config (TIMER0, 98);
```

## timer\_autoreload\_value\_config

The description of timer\_autoreload\_value\_config is shown as below:

**Table 3-426. Function timer\_autoreload\_value\_config**

<b>Function name</b>	timer_autoreload_value_config
<b>Function prototype</b>	void timer_autoreload_value_config(uint32_t timer_periph, uint16_t autoreload);
<b>Function descriptions</b>	configure TIMER autoreload register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..6)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>autoreload</b>	the counter auto-reload value (0-65535)
<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 3-427. Function timer\_counter\_value\_config**

<b>Function name</b>	timer_counter_value_config
<b>Function prototype</b>	void timer_counter_value_config(uint32_t timer_periph, uint16_t counter);
<b>Function descriptions</b>	configure TIMER counter register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..6)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>counter</b>	the counter value (0-65535)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 counter register value */
timer_counter_value_config (TIMER0, 3000);
```

### timer\_counter\_read

The description of timer\_counter\_read is shown as below:

**Table 3-428. Function timer\_counter\_read**

<b>Function name</b>	timer_counter_read
<b>Function prototype</b>	uint32_t timer_counter_read(uint32_t timer_periph);
<b>Function descriptions</b>	read TIMER counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..6)</i>	TIMER peripheral selection

Output parameter{out}	
-	-
Return value	
uint32_t	counter value (0~65535)

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 3-429. Function timer\_prescaler\_read**

Function name	timer_prescaler_read
Function prototype	uint16_t timer_prescaler_read(uint32_t timer_periph);
Function descriptions	read TIMER prescaler value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..6)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
uint16_t	prescaler register value (0~65535)

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 3-430. Function timer\_single\_pulse\_mode\_config**

Function name	timer_single_pulse_mode_config
Function prototype	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);
Function descriptions	configure TIMER single pulse mode
Precondition	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..6)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>spmode</b>	pulse mode
<i>TIMER_SP_MODE_SINGLE</i>	single pulse mode
<i>TIMER_SP_MODE_REPETITIVE</i>	repetitive pulse mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 single pulse mode */
```

```
timer_single_pulse_mode_config (TIMER0, TIMER_SP_MODE_SINGLE);
```

### timer\_update\_source\_config

The description of timer\_update\_source\_config is shown as below:

**Table 3-431. 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..6)</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 3-432. Function timer\_dma\_enable**

<b>Function name</b>	timer_dma_enable
<b>Function prototype</b>	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
<b>Function descriptions</b>	enable the TIMER DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>dma</b>	timer DMA source enable
<i>TIMER_DMA_UPD</i>	update DMA enable, <i>TIMERx</i> (x=0..6)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable, <i>TIMERx</i> (x=0..4)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable, <i>TIMERx</i> (x=0..4)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA enable, <i>TIMERx</i> (x=0..4)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA enable, <i>TIMERx</i> (x=0..4)
<i>TIMER_DMA_CMTD</i>	commutation DMA request enable, <i>TIMERx</i> (x=0)
<i>TIMER_DMA_TRGD</i>	trigger DMA enable, <i>TIMERx</i> (x=0..4)
<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 3-433. 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 enable, TIMERx(x=0..6)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable, TIMERx(x=0..4)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable, TIMERx(x=0..4)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA enable, TIMERx(x=0..4)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA enable, TIMERx(x=0..4)
<i>TIMER_DMA_CMTD</i>	commutation DMA request enable, TIMERx(x=0)
<i>TIMER_DMA_TRGD</i>	trigger DMA enable, TIMERx(x=0..4)
<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 3-434. Function timer\_channel\_dma\_request\_source\_select**

<b>Function name</b>	timer_channel_dma_request_source_select
<b>Function prototype</b>	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
<b>Function descriptions</b>	channel DMA request source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>dma_request</b>	channel DMA request source selection
<i>TIMER_DMAREQUEST_CHANNELEVENT</i>	DMA request of channel n is sent when channel y event occurs

<i>TIMER_DMAREQUEST_UPDATEEVENT</i>	DMA request of channel n is sent when update event occurs
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* TIMER0 channel DMA request of channel n is sent when channel y event occurs */
```

```
timer_channel_dma_request_source_select(TIMER0,  
TIMER_DMAREQUEST_CHANNELEVENT);
```

### timer\_dma\_transfer\_config

The description of timer\_dma\_transfer\_config is shown as below:

**Table 3-435. Function timer\_dma\_transfer\_config**

<b>Function name</b>	timer_dma_transfer_config
<b>Function prototype</b>	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
<b>Function descriptions</b>	configure the TIMER DMA transfer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<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)
<i>TIMER_DMACFG_DMA_TA_CTL1</i>	DMA transfer address is TIMER_CTL1, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA_TA_SMCFG</i>	DMA transfer address is TIMER_SMCFG, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA_TA_DMAINTEN</i>	DMA transfer address is TIMER_DMAINTEN, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA_TA_INTF</i>	DMA transfer address is TIMER_INTF, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA_TA_SWEVG</i>	DMA transfer address is TIMER_SWEVG, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA_TA_CHCTL0</i>	DMA transfer address is TIMER_CHCTL0, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA_TA_CHCTL1</i>	DMA transfer address is TIMER_CHCTL1, TIMERx(x=0..4)

<i>TA_CHCTL1</i>	
<i>TIMER_DMACFG_DMA</i> <i>TA_CHCTL2</i>	DMA transfer address is <i>TIMER_CHCTL2</i> , <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_DMACFG_DMA</i> <i>TA_CNT</i>	DMA transfer address is <i>TIMER_CNT</i> , <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_DMACFG_DMA</i> <i>TA_PSC</i>	DMA transfer address is <i>TIMER_PSC</i> , <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_DMACFG_DMA</i> <i>TA_CAR</i>	MA transfer address is <i>TIMER_CAR</i> , <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_DMACFG_DMA</i> <i>TA_CREP</i>	DMA transfer address is <i>TIMER_CREP</i> , <i>TIMERx</i> ( <i>x</i> =0)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH0CV</i>	DMA transfer address is <i>TIMER_CH0CV</i> , <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH1CV</i>	DMA transfer address is <i>TIMER_CH1CV</i> , <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH2CV</i>	DMA transfer address is <i>TIMER_CH2CV</i> , <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH3CV</i>	DMA transfer address is <i>TIMER_CH3CV</i> , <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_DMACFG_DMA</i> <i>TA_CCHP</i>	DMA transfer address is <i>TIMER_CCHP</i> , <i>TIMERx</i> ( <i>x</i> =0)
<i>TIMER_DMACFG_DMA</i> <i>TA_DMACFG</i>	DMA transfer address is <i>TIMER_DMACFG</i> , <i>TIMERx</i> ( <i>x</i> =0..4)
<b>Input parameter{in}</b>	
<b><i>dma_lenth</i></b>	DMA transfer count
<i>TIMER_DMACFG_DMA</i> <i>TC_xTRANSFER</i>	<i>x</i> =1..18, DMA transfer <i>x</i> time
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the TIMER0 DMA transfer */
```

```
timer_dma_transfer_config(TIMER0, TIMER_DMACFG_DMATA_CTL0,  
TIMER_DMACFG_DMATC_5TRANSFER);
```

### timer\_event\_software\_generate

The description of `timer_event_software_generate` is shown as below:

**Table 3-436. Function `timer_event_software_generate`**

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

<b>Function prototype</b>	void timer_event_software_generate(uint32_t timer_periph, uint16_t event);
<b>Function descriptions</b>	software generate events
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>event</b>	the timer software event generation sources
<i>TIMER_EVENT_SRC_UPG</i>	update event, TIMERx(x=0..6)
<i>TIMER_EVENT_SRC_CH0G</i>	channel 0 capture or compare event generation, TIMERx(x=0..4)
<i>TIMER_EVENT_SRC_CH1G</i>	channel 1 capture or compare event generation, TIMERx(x=0..4)
<i>TIMER_EVENT_SRC_CH2G</i>	channel 2 capture or compare event generation, TIMERx(x=0..4)
<i>TIMER_EVENT_SRC_CH3G</i>	channel 3 capture or compare event generation, TIMERx(x=0..4)
<i>TIMER_EVENT_SRC_COMG</i>	channel commutation event generation, TIMERx(x=0)
<i>TIMER_EVENT_SRC_TRIGG</i>	trigger event generation, TIMERx(x=0..4)
<i>TIMER_EVENT_SRC_BREAKG</i>	break event generation, TIMERx(x=0)
<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 3-437. 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 3-409. Structure timer break parameter struct.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-438. 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)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>breakpara</b>	TIMER break parameter struct, the structure members can refer to <a href="#">Table 3-409. Structure timer break parameter struct.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 break function */
```

```
timer_break_parameter_struct timer_breakpara;
```

```

timer_breakpara.runoffstate      = TIMER_ROS_STATE_DISABLE;

timer_breakpara.ideloffstate     = TIMER_IOS_STATE_DISABLE ;

timer_breakpara.deadtime         = 255;

timer_breakpara.breakpolarity    = TIMER_BREAK_POLARITY_LOW;

timer_breakpara.outputautostate  = TIMER_OUTAUTO_ENABLE;

timer_breakpara.protectmode       = TIMER_CCHP_PROT_0;

timer_breakpara.breakstate       = TIMER_BREAK_ENABLE;

timer_break_config(TIMER0, &timer_breakpara);

```

### timer\_break\_enable

The description of timer\_break\_enable is shown as below:

**Table 3-439. Function timer\_break\_enable**

<b>Function name</b>	timer_break_enable
<b>Function prototype</b>	void timer_break_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable TIMER break function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0)</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 3-440. Function timer\_break\_disable**

<b>Function name</b>	timer_break_disable
<b>Function prototype</b>	void timer_break_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER break function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in

	TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 break function*/
```

```
timer_break_disable (TIMER0);
```

### timer\_automatic\_output\_enable

The description of timer\_automatic\_output\_enable is shown as below:

**Table 3-441. Function timer\_automatic\_output\_enable**

<b>Function name</b>	timer_automatic_output_enable
<b>Function prototype</b>	void timer_automatic_output_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable TIMER output automatic function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0)</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 3-442. Function timer\_automatic\_output\_disable**

<b>Function name</b>	timer_automatic_output_disable
----------------------	--------------------------------



<b>Function prototype</b>	void timer_automatic_output_disable (uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER output automatic function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0)</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 3-443. 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, 14..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 primary output function */
```

```
timer_primary_output_config (TIMER0, ENABLE);
```

## timer\_channel\_control\_shadow\_config

The description of timer\_channel\_control\_shadow\_config is shown as below:

**Table 3-444. Function timer\_channel\_control\_shadow\_config**

<b>Function name</b>	timer_channel_control_shadow_config
<b>Function prototype</b>	void timer_channel_control_shadow_config(uint32_t timer_periph, ControlStatus newvalue);
<b>Function descriptions</b>	channel commutation control shadow register enable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* channel capture/compare control shadow register enable */
timer_channel_control_shadow_config (TIMER0, ENABLE);
```

## timer\_channel\_control\_shadow\_update\_config

The description of timer\_channel\_control\_shadow\_update\_config is shown as below:

**Table 3-445. 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)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>ccuctl</b>	channel control shadow register update control
<i>TIMER_UPDATECTL_C</i>	the shadow registers update by when CMTG bit is set

<i>CU</i>	
<i>TIMER_UPDATECTL_C</i> <i>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 3-446. Function timer\_channel\_output\_struct\_para\_init**

<b>Function name</b>	timer_channel_output_struct_para_init
<b>Function prototype</b>	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpa);
<b>Function descriptions</b>	initialize the parameters of TIMER channel output parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ocpara</b>	TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-410. 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 3-447. Function timer\_channel\_output\_config**

<b>Function name</b>	timer_channel_output_config
----------------------	-----------------------------

<b>Function prototype</b>	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara);
<b>Function descriptions</b>	configure TIMER channel output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4))
<b>Input parameter{in}</b>	
<b>ocpara</b>	TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-410. 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 3-448. Function timer\_channel\_output\_mode\_config**

<b>Function name</b>	timer_channel_output_mode_config
<b>Function prototype</b>	void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t

	channel, uint16_t ocmode);
<b>Function descriptions</b>	configure TIMER channel output compare mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx (x=0..4))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx (x=0..4))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx (x=0..4))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx (x=0..4))
<b>Input parameter{in}</b>	
<b>ocmode</b>	channel output compare mode
<i>TIMER_OC_MODE_TIMING</i>	timing mode
<i>TIMER_OC_MODE_ACTIVE</i>	set the channel output
<i>TIMER_OC_MODE_INACTIVE</i>	clear the channel output
<i>TIMER_OC_MODE_TOGGLE</i>	toggle on match
<i>TIMER_OC_MODE_LOW</i>	force low mode
<i>TIMER_OC_MODE_HIGH</i>	force high mode
<i>TIMER_OC_MODE_PWM0</i>	PWM mode 0
<i>TIMER_OC_MODE_PWM1</i>	PWM mode 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel PWM 0 mode */
```

```
timer_channel_output_mode_config(TIMER0, TIMER_CH_0, TIMER_OC_MODE_PWM0);
```

### timer\_channel\_output\_pulse\_value\_config

The description of timer\_channel\_output\_pulse\_value\_config is shown as below:

Table 3-449. Function timer\_channel\_output\_pulse\_value\_config

Function name	timer_channel_output_pulse_value_config
Function prototype	void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse);
Function descriptions	configure TIMER channel output pulse value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0 (TIMERx (x=0..4))
TIMER_CH_1	TIMER channel 1 (TIMERx (x=0..4))
TIMER_CH_2	TIMER channel 2 (TIMERx (x=0..4))
TIMER_CH_3	IMER channel 3 (TIMERx (x=0..4))
Input parameter{in}	
pulse	channel output pulse value (0~65535)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output pulse value */
timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

### timer\_channel\_output\_shadow\_config

The description of timer\_channel\_output\_shadow\_config is shown as below:

Table 3-450. Function timer\_channel\_output\_shadow\_config

Function name	timer_channel_output_shadow_config
Function prototype	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
Function descriptions	configure TIMER channel output shadow function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured

<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx (x=0..4))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx (x=0..4))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx (x=0..4))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx (x=0..4))
<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 3-451. Function timer\_channel\_output\_fast\_config**

<b>Function name</b>	timer_channel_output_fast_config
<b>Function prototype</b>	void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);
<b>Function descriptions</b>	configure TIMER channel output fast function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx (x=0..4))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx (x=0..4))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx (x=0..4))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx (x=0..4))
<b>Input parameter{in}</b>	
<b>ocfast</b>	channel output fast function
<i>TIMER_OC_FAST_ENA</i>	channel output fast function enable

<i>BLE</i>	
<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 3-452. Function timer\_channel\_output\_clear\_config**

<b>Function name</b>	timer_channel_output_clear_config
<b>Function prototype</b>	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);
<b>Function descriptions</b>	configure TIMER channel output clear function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER periphera
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx (x=0..4))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx (x=0..4))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx (x=0..4))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx (x=0..4))
<b>Input parameter{in}</b>	
<b>occlear</b>	channel output clear function
<i>TIMER_OC_CLEAR_ENABLE</i>	channel output clear function enable
<i>TIMER_OC_CLEAR_DISABLE</i>	channel output clear function disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:



```
/* configure TIMER0 channel 0 output clear function */
```

```
timer_channel_output_clear_config (TIMER0, TIMER_CH_0,  
TIMER_OC_CLEAR_ENABLE);
```

### timer\_channel\_output\_polarity\_config

The description of timer\_channel\_output\_polarity\_config is shown as below:

**Table 3-453. Function timer\_channel\_output\_polarity\_config**

<b>Function name</b>	timer_channel_output_polarity_config
<b>Function prototype</b>	void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
<b>Function descriptions</b>	configure TIMER channel output polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx (x=0..4))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx (x=0..4))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx (x=0..4))
<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 3-454. Function timer\_channel\_complementary\_output\_polarity\_config

Function name	timer_channel_complementary_output_polarity_config
Function prototype	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnpolarity);
Function descriptions	configure TIMER channel complementary output polarity
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	TIMER peripheral selection
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0(TIMERx (x=0))
TIMER_CH_1	TIMER channel 1(TIMERx (x=0))
TIMER_CH_2	TIMER channel 2(TIMERx (x=0))
Input parameter{in}	
ocpolarity	channel complementary output polarity
TIMER_OCN_POLARITY_HIGH	channel complementary output polarity is high
TIMER_OCN_POLARITY_LOW	channel complementary output polarity is low
Output parameter{out}	
-	-
Return value	
-	-

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 3-455. Function timer\_channel\_output\_state\_config

Function name	timer_channel_output_state_config
Function prototype	void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
Function descriptions	configure TIMER channel enable state
Precondition	-
The called functions	-
Input parameter{in}	

<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> ( <i>x</i> =0..4))
<i>TIMER_CH_1</i>	TIMER channel 1( <i>TIMERx</i> ( <i>x</i> =0..4))
<i>TIMER_CH_2</i>	TIMER channel 2( <i>TIMERx</i> ( <i>x</i> =0..4))
<i>TIMER_CH_3</i>	TIMER channel 3( <i>TIMERx</i> ( <i>x</i> =0..4))
<b>Input parameter{in}</b>	
<b>state</b>	TIMER channel enable state
<i>TIMER_CCX_ENABLE</i>	channel enable
<i>TIMER_CCX_DISABLE</i>	channel disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 enable state */
```

```
timer_channel_output_state_config (TIMER0, TIMER_CH_0, TIMER_CCX_ENABLE);
```

### timer\_channel\_complementary\_output\_state\_config

The description of timer\_channel\_complementary\_output\_state\_config is shown as below:

**Table 3-456. 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,..16)</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>TIMERx</i> ( <i>x</i> =0)
<i>TIMER_CH_1</i>	TIMER channel 1, <i>TIMERx</i> ( <i>x</i> =0)
<i>TIMER_CH_2</i>	TIMER channel 2, <i>TIMERx</i> ( <i>x</i> =0)
<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_DISABL</i> <i>E</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 3-457. Function timer\_channel\_input\_struct\_para\_init**

<b>Function name</b>	timer_channel_input_struct_para_init
<b>Function prototype</b>	void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);
<b>Function descriptions</b>	initialize the parameters of TIMER channel input parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
icpara	TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-411. 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 3-458. Function timer\_input\_capture\_config**

<b>Function name</b>	timer_input_capture_config
----------------------	----------------------------

<b>Function prototype</b>	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);
<b>Function descriptions</b>	configure TIMER input capture parameter
<b>Precondition</b>	-
<b>The called functions</b>	timer_channel_input_capture_prescaler_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx (x=0..4))
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx (x=0..4))
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx (x=0..4))
<i>TIMER_CH_3</i>	TIMER channel 3(TIMERx (x=0..4))
<b>Input parameter{in}</b>	
<b>icpara</b>	TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-411. Structure timer_ic_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure TIMER0 input capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);

```

### timer\_channel\_input\_capture\_prescaler\_config

The description of timer\_channel\_input\_capture\_prescaler\_config is shown as below:

**Table 3-459. Function timer\_channel\_input\_capture\_prescaler\_config**

<b>Function name</b>	timer_channel_input_capture_prescaler_config
<b>Function prototype</b>	void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler);
<b>Function descriptions</b>	configure TIMER channel input capture prescaler value
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx (x=0..4))
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx (x=0..4))
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx (x=0..4))
<i>TIMER_CH_3</i>	TIMER channel 3(TIMERx (x=0..4))
<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 3-460. Function timer\_channel\_capture\_value\_register\_read**

<b>Function name</b>	timer_channel_capture_value_register_read
<b>Function prototype</b>	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
<b>Function descriptions</b>	read TIMER channel capture compare register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx (x=0..4))

<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx (x=0..4))
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx (x=0..4))
<i>TIMER_CH_3</i>	TIMER channel 3(TIMERx (x=0..4))
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	channel capture compare register value (0~65535)

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 3-461. Function timer\_input\_pwm\_capture\_config**

<b>Function name</b>	timer_input_pwm_capture_config
<b>Function prototype</b>	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
<b>Function descriptions</b>	configure TIMER input pwm capture function
<b>Precondition</b>	-
<b>The called functions</b>	timer_channel_input_capture_prescaler_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0
<i>TIMER_CH_1</i>	TIMER channel 1
<b>Input parameter{in}</b>	
<b>icpwm</b>	TIMER channel input pwm parameter struct, the structure members can refer to <a href="#">Table 3-411. Structure timer_ic_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 input pwm capture parameter */
```

```
timer_ic_parameter_struct timer_icinitpara;
```

```

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_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 3-462. Function timer\_hall\_mode\_config**

Function name	timer_hall_mode_config
Function prototype	void timer_hall_mode_config(uint32_t timer_periph, uint8_t hallmode);
Function descriptions	configure TIMER hall sensor mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..4)	TIMER peripheral selection
Input parameter{in}	
hallmode	TIMER hall sensor mode state
TIMER_HALLINTERFACE_ENABLE	TIMER hall sensor mode enable
TIMER_HALLINTERFACE_DISABLE	TIMER hall sensor mode disable
Output parameter{out}	
-	-
Return value	
-	-

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 3-463. Function timer\_input\_trigger\_source\_select**

Function name	timer_input_trigger_source_select
Function prototype	void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger);



<b>Function descriptions</b>	select TIMER input trigger source
<b>Precondition</b>	SMC[2:0] = 000
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>intrigger</b>	trigger selection
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI0</i>	Internal trigger input 0 (ITI0, TIMERx(x=0..4))
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI1</i>	Internal trigger input 0 (ITI1, TIMERx(x=0..4))
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI2</i>	Internal trigger input 0 (ITI2, TIMERx(ITI2, TIMERx(x=0..4))
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI3</i>	Internal trigger input 0 (ITI3, TIMERx(x=0..4))
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOF_ED</i>	CIO edge flag (CIOF_ED, TIMERx(x=0..4))
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOFE0</i>	channel 0 input Filtered output(CIOFE0, TIMERx(x=0..4))
<i>TIMER_SMCFG_TRGS</i> <i>EL_CI1FE1</i>	channel 1 input Filtered output(CI1FE1, TIMERx(x=0..4))
<i>TIMER_SMCFG_TRGS</i> <i>EL_ETIFP</i>	External trigger input filter output(ETIFP, TIMERx(x=0..4))
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select TIMER0 input trigger source */
```

```
timer_input_trigger_source_select (TIMER0, TIMER_SMCFG_TRGSEL_ITI0);
```

### timer\_master\_output\_trigger\_source\_select

The description of timer\_master\_output\_trigger\_source\_select is shown as below:

**Table 3-464. 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 outtrigger);
<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..6)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>outrigger</b>	master mode control
<i>TIMER_TRI_OUT_SRC_RESET</i>	Reset. When the UPG bit in the <i>TIMERx_SWEVG</i> register is set or a reset is generated by the slave mode controller, a TRGO pulse occurs. And in the latter case, the signal on TRGO is delayed compared to the actual reset
<i>TIMER_TRI_OUT_SRC_ENABLE</i>	Enable. This mode is useful to start several timers at the same time or to control a window in which a slave timer is enabled. In this mode the master mode controller selects the counter enable signal as TRGO. The counter enable signal is set when CEN control bit is set or the trigger input in pause mode is high. There is a delay between the trigger input in pause mode and the TRGO output, except if the master-slave mode is selected.
<i>TIMER_TRI_OUT_SRC_UPDATE</i>	Update. In this mode the master mode controller selects the update event as TRGO.
<i>TIMER_TRI_OUT_SRC_CH0</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 3-465. Function `timer_slave_mode_select`**

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

<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)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>slavemode</b>	slave mode
<i>TIMER_SLAVE_MODE_DISABLE</i>	slave mode disable, <i>TIMERx(x=0..4)</i>
<i>TIMER_QUAD_DECODER_MODE0</i>	quadrature decoder mode 0, <i>TIMERx(x=0..4)</i>
<i>TIMER_QUAD_DECODER_MODE1</i>	quadrature decoder mode 1, <i>TIMERx(x=0..4)</i>
<i>TIMER_QUAD_DECODER_MODE2</i>	quadrature decoder mode 2, <i>TIMERx(x=0..4)</i>
<i>TIMER_SLAVE_MODE_RESTART</i>	restart mode, <i>TIMERx(x=0..4)</i>
<i>TIMER_SLAVE_MODE_PAUSE</i>	pause mode, <i>TIMERx(x=0..4)</i>
<i>TIMER_SLAVE_MODE_EVENT</i>	event mode, <i>TIMERx(x=0..4)</i>
<i>TIMER_SLAVE_MODE_EXTERNAL0</i>	external clock mode 0, <i>TIMERx(x=0..4)</i>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select TIMER0 slave mode */
```

```
timer_slave_mode_select (TIMER0, TIMER_QUAD_DECODER_MODE0);
```

### timer\_master\_slave\_mode\_config

The description of timer\_master\_slave\_mode\_config is shown as below:

**Table 3-466. Function timer\_master\_slave\_mode\_config**

<b>Function name</b>	timer_master_slave_mode_config
<b>Function prototype</b>	void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t masterslave);
<b>Function descriptions</b>	configure TIMER master slave mode

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
TIMERx(x=0..4)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>masterslave</b>	master slave mode state
TIMER_MASTER_SLAVE_MODE_ENABLE	master slave mode enable
TIMER_MASTER_SLAVE_MODE_DISABLE	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 3-467. 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
TIMERx(x=0..4)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	external trigger prescaler
TIMER_EXT_TRI_PSC_OFF	no divided
TIMER_EXT_TRI_PSC_DIV2	divided by 2
TIMER_EXT_TRI_PSC_DIV4	divided by 4
TIMER_EXT_TRI_PSC_DIV8	divided by 8

<i>DIV8</i>	
<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 3-468. 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)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>decomode</b>	quadrature decoder mode
<i>TIMER_QUAD_DECODER_MODE0</i>	counter counts on CI0FE0 edge depending on CI1FE1 level
<i>TIMER_QUAD_DECODER_MODE1</i>	counter counts on CI1FE1 edge depending on CI0FE0 level
<i>TIMER_QUAD_DECODER_MODE2</i>	counter counts on both CI0FE0 and CI1FE1 edges depending on the level of the other input
<b>Input parameter{in}</b>	
<b>ic0polarity</b>	IC0 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge

<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
<b>Input parameter{in}</b>	
<b>ic1polarity</b>	IC1 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 quadrature decoder mode */
```

```
timer_quadrature_decoder_mode_config (TIMER0, TIMER_QUAD_DECODER_MODE0,
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

### timer\_internal\_clock\_config

The description of timer\_internal\_clock\_config is shown as below:

**Table 3-469. 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)</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 3-470. 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)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>intrigger</b>	trigger selection
<i>TIMER_SMCFG_TRGS_EL_ITI0</i>	Internal trigger input 0 (ITI0), TIMERx(x=0..4)
<i>TIMER_SMCFG_TRGS_EL_ITI1</i>	Internal trigger input 0 (ITI1) , TIMERx(x=0..4)
<i>TIMER_SMCFG_TRGS_EL_ITI2</i>	Internal trigger input 0 (ITI2) , TIMERx(x=0..4)
<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 3-471. Function timer\_external\_trigger\_as\_external\_clock\_config

<b>Function name</b>	timer_external_trigger_as_external_clock_config
<b>Function prototype</b>	void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t extrigger, uint16_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external trigger as external clock input
<b>Precondition</b>	-
<b>The called functions</b>	timer_input_trigger_source_select
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extrigger</b>	external trigger selection

<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOF_ED</i>	CI0 edge flag (CI0F_ED)
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOFE0</i>	channel 0 input Filtered output (CI0FE0)
<i>TIMER_SMCFG_TRGS</i> <i>EL_C1FE1</i>	channel 1 input Filtered output (CI1FE1)
<b>Input parameter{in}</b>	
<b>expolarity</b>	external trigger polarity
<i>TIMER_IC_POLARITY_</i> <i>RISING</i>	active high or rising edge active
<i>TIMER_IC_POLARITY_</i> <i>FALLING</i>	active low or falling edge active
<i>TIMER_IC_POLARITY_</i> <i>BOTH_EDGE</i>	falling edge 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 the external trigger CIOFE0 as external clock input */
```

```
timer_external_trigger_as_external_clock_config (TIMER0,  
TIMER_SMCFG_TRGSEL_CIOFE0, TIMER_IC_POLARITY_RISING, 0);
```

### timer\_external\_clock\_mode0\_config

The description of timer\_external\_clock\_mode0\_config is shown as below:

**Table 3-472. 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)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	ETI external trigger prescaler
<i>TIMER_EXT_TRI_PSC_</i>	no divided



OFF	
TIMER_EXT_TRI_PSC_DIV2	divided by 2
TIMER_EXT_TRI_PSC_DIV4	divided by 4
TIMER_EXT_TRI_PSC_DIV8	divided by 8
<b>Input parameter{in}</b>	
<b>expolarity</b>	ETI external trigger polarity
TIMER_ETP_FALLING	active low or falling edge active
TIMER_ETP_RISING	active high or rising edge active
<b>Input parameter{in}</b>	
<b>extfilter</b>	ETI external trigger filter control (0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 the external clock mode0 */
```

```
timer_external_clock_mode0_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 0);
```

### timer\_external\_clock\_mode1\_config

The description of timer\_external\_clock\_mode1\_config is shown as below:

**Table 3-473. Function timer\_external\_clock\_mode1\_config**

<b>Function name</b>	timer_external_clock_mode1_config
<b>Function prototype</b>	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external clock mode1
<b>Precondition</b>	-
<b>The called functions</b>	timer_external_trigger_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
TIMERx(x=0..4)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	ETI external trigger prescaler
TIMER_EXT_TRI_PSC_OFF	no divided
TIMER_EXT_TRI_PSC_DIV2	divided by 2

<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 3-474. Function timer\_external\_clock\_mode1\_disable**

<b>Function name</b>	timer_external_clock_mode1_disable
<b>Function prototype</b>	void timer_external_clock_mode1_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER the external clock mode1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_disable (TIMER0);
```

## timer\_interrupt\_enable

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

**Table 3-475. 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..6)
<i>TIMER_INT_CH0</i>	channel 0 interrupt enable, TIMERx(x=0..4)
<i>TIMER_INT_CH1</i>	channel 1 interrupt enable, TIMERx(x=0..4)
<i>TIMER_INT_CH2</i>	channel 2 interrupt enable, TIMERx(x=0..4)
<i>TIMER_INT_CH3</i>	channel 3 interrupt enable , TIMERx(x=0..4)
<i>TIMER_INT_CMT</i>	commutation interrupt enable, TIMERx (x=0)
<i>TIMER_INT_TRG</i>	trigger interrupt enable, TIMERx(x=0..4)
<i>TIMER_INT_BRK</i>	break interrupt enable, TIMERx (x=0)
<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 3-476. 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, <i>TIMERx</i> ( <i>x</i> =0..6)
<i>TIMER_INT_CH0</i>	channel 0 interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_CH1</i>	channel 1 interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_CH2</i>	channel 2 interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_CH3</i>	channel 3 interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_CMT</i>	commutation interrupt disable, <i>TIMERx</i> ( <i>x</i> =0)
<i>TIMER_INT_TRG</i>	trigger interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_BRK</i>	break interrupt disable, <i>TIMERx</i> ( <i>x</i> =0)
<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 3-477. Function timer\_interrupt\_flag\_get**

<b>Function name</b>	timer_interrupt_flag_get
<b>Function prototype</b>	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	get timer interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..6)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, <i>TIMERx</i> <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, <i>TIMERx</i> <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_FLAG_CMT</i> <i>T</i>	channel commutation interrupt flag, <i>TIMERx</i> ( <i>x</i> =0)

<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, <i>TIMERx</i> ( <i>x</i> =0)
<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 3-478. Function `timer_interrupt_flag_clear`**

<b>Function name</b>	<code>timer_interrupt_flag_clear</code>
<b>Function prototype</b>	<code>void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t interrupt);</code>
<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, <i>TIMERx</i> ( <i>x</i> =0..6)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, <i>TIMERx</i> <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, <i>TIMERx</i> <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_FLAG_CMT</i>	channel commutation interrupt flag, <i>TIMERx</i> ( <i>x</i> =0)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, <i>TIMERx</i> ( <i>x</i> =0)
<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 3-479. Function timer\_flag\_get**

<b>Function name</b>	timer_flag_get
<b>Function prototype</b>	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);
<b>Function descriptions</b>	get TIMER flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>flag</b>	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag, TIMERx(x=0..6)
<i>TIMER_FLAG_CH0</i>	channel 0 flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH1</i>	channel 1 flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH2</i>	channel 2 flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH3</i>	channel 3 flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CMT</i>	channel commutation flag, TIMERx(x=0)
<i>TIMER_FLAG_TRG</i>	trigger flag, TIMERx(x=0..4)
<i>TIMER_FLAG_BRK</i>	break flag, TIMERx(x=0)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag, TIMERx(x=0..4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TIMER0 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get (TIMER0, TIMER_FLAG_UP);
```

## timer\_flag\_clear

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

Table 3-480. 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..6)
<i>TIMER_FLAG_CH0</i>	channel 0 flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH1</i>	channel 1 flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH2</i>	channel 2 flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH3</i>	channel 3 flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CMT</i>	channel commutation flag, TIMERx(x=0)
<i>TIMER_FLAG_TRG</i>	trigger flag, TIMERx(x=0..4)
<i>TIMER_FLAG_BRK</i>	break flag, TIMERx(x=0)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag, TIMERx(x=0..4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TIMER0 update flags */
timer_flag_clear (TIMER0, TIMER_FLAG_UP);
```

## 3.21. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter [3.21.1](#), the USART firmware functions are introduced in chapter [3.21.2](#).

### 3.21.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

**Table 3-481. USART Registers**

Registers	Descriptions
USART_STAT	Status register
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

### 3.21.2. Descriptions of Peripheral functions

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

**Table 3-482. USART firmware function**

Function name	Function description
usart_deinit	reset USART
usart_baudrate_set	configure USART baud rate value
usart_parity_config	configure USART parity function
usart_word_length_set	configure USART word length
usart_stop_bit_set	configure USART stop bit length
usart_enable	enable USART
usart_disable	disable USART
usart_transmit_config	configure USART transmitter
usart_receive_config	configure USART receiver
usart_data_transmit	USART transmit data function
usart_data_receive	USART receive data function
usart_address_config	configure the address of the USART in wake up by address match mode
usart_mute_mode_enable	receiver in mute mode
usart_mute_mode_disable	receiver in active mode
usart_mute_mode_wakeup_config	configure wakeup method in mute mode
usart_lin_mode_enable	enable LIN mode
usart_lin_mode_disable	disable LIN mode
usart_lin_break_dection_length_config	configure LIN break frame length
usart_send_break	send break frame
usart_halfduplex_enable	enable half duplex mode
usart_halfduplex_disable	disable half duplex mode
usart_synchronous_clock_enable	enable USART clock
usart_synchronous_clock_disable	disable USART clock
usart_synchronous_clock_config	configure USART synchronous mode parameters
usart_guard_time_config	configure guard time value in smartcard mode



Function name	Function description
usart_smartcard_mode_enable	enable smartcard mode
usart_smartcard_mode_disable	disable smartcard mode
usart_smartcard_mode_nack_enable	enable NACK in smartcard mode
usart_smartcard_mode_nack_disable	disable NACK in smartcard mode
usart_irda_mode_enable	enable IrDA mode
usart_irda_mode_disable	disable IrDA mode
usart_prescaler_config	configure the peripheral clock prescaler in USART IrDA low-power mode
usart_irda_lowpower_config	configure IrDA low-power
usart_hardware_flow_rts_config	configure hardware flow control RTS
usart_hardware_flow_cts_config	configure hardware flow control CTS
usart_dma_receive_config	configure USART DMA for reception
usart_dma_transmit_config	configure USART DMA for transmission
usart_flag_get	get flag in STAT/RFCs register
usart_flag_clear	clear flag in STAT register
usart_interrupt_enable	enable USART interrupt
usart_interrupt_disable	disable USART interrupt
usart_interrupt_flag_get	get USART interrupt and flag status
usart_interrupt_flag_clear	clear USART interrupt flag in STAT register

## usart\_deinit

The description of usart\_deinit is shown as below:

**Table 3-483. Function usart\_deinit**

<b>Function name</b>	usart_deinit
<b>Function prototype</b>	void usart_deinit(uint32_t usart_periph);
<b>Function descriptions</b>	reset USART/UART
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* reset USART0 */
usart_deinit(USART0);

```

## usart\_baudrate\_set

The description of usart\_baudrate\_set is shown as below:

**Table 3-484. Function usart\_baudrate\_set**

<b>Function name</b>	usart_baudrate_set
<b>Function prototype</b>	void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);
<b>Function descriptions</b>	configure USART baud rate value
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>baudval</b>	baud rate value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 baud rate value */
usart_baudrate_set(USART0, 115200);
```

## usart\_parity\_config

The description of usart\_parity\_config is shown as below:

**Table 3-485. Function usart\_parity\_config**

<b>Function name</b>	usart_parity_config
<b>Function prototype</b>	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
<b>Function descriptions</b>	configure USART parity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>paritycfg</b>	configure USART parity
USART_PM_NONE	no parity
USART_PM_ODD	odd parity
USART_PM_EVEN	even parity

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 parity */
```

```
usart_parity_config(USART0, USART_PM_EVEN);
```

### usart\_word\_length\_set

The description of usart\_word\_length\_set is shown as below:

**Table 3-486. Function usart\_word\_length\_set**

Function name	usart_word_length_set
Function prototype	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);
Function descriptions	configure USART word length
Precondition	-
The called functions	-
Input parameter{in}	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
UARTx	x=3,4
Input parameter{in}	
<b>wlen</b>	USART word length configure
USART_WL_8BIT	8 bits
USART_WL_9BIT	9 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 word length */
```

```
usart_word_length_set(USART0, USART_WL_9BIT);
```

### usart\_stop\_bit\_set

The description of usart\_stop\_bit\_set is shown as below:

**Table 3-487. Function usart\_stop\_bit\_set**

Function name	usart_stop_bit_set
Function prototype	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);

<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>	usart peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>stblen</b>	USART stop bit configure
<i>USART_STB_1BIT</i>	1 bit
<i>USART_STB_0_5BIT</i>	0.5 bit
<i>USART_STB_2BIT</i>	2 bits
<i>USART_STB_1_5BIT</i>	1.5 bits
<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 3-488. Function usart\_enable**

<b>Function name</b>	usart_enable
<b>Function prototype</b>	void usart_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable USART
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 */
```

```
usart_enable(USART0);
```

## usart\_disable

The description of usart\_disable is shown as below:

**Table 3-489. Function usart\_disable**

<b>Function name</b>	usart_disable
<b>Function prototype</b>	void usart_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable USART
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 */
usart_disable(USART0);
```

## usart\_transmit\_config

The description of usart\_transmit\_config is shown as below:

**Table 3-490. Function usart\_transmit\_config**

<b>Function name</b>	usart_transmit_config
<b>Function prototype</b>	void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);
<b>Function descriptions</b>	configure USART transmitter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>txconfig</b>	enable or disable USART transmitter
USART_TRANSMIT_ENABLE	enable USART transmission
USART_TRANSMIT_DISABLE	disable USART transmission

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

Example:

```
/* configure USART0 transmitter */
```

```
usart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

### usart\_receive\_config

The description of usart\_receive\_config is shown as below:

**Table 3-491. Function usart\_receive\_config**

<b>Function name</b>	usart_receive_config
<b>Function prototype</b>	void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);
<b>Function descriptions</b>	configure USART receiver
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
Input parameter{in}	
<b>rxconfig</b>	enable or disable USART receiver
<i>USART_RECEIVE_ENABLE</i>	enable USART reception
<i>USART_RECEIVE_DISABLE</i>	disable USART reception
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* Configure USART0 receiver */
```

```
usart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

### usart\_data\_transmit

The description of usart\_data\_transmit is shown as below:

Table 3-492. Function usart\_data\_transmit

<b>Function name</b>	usart_data_transmit
<b>Function prototype</b>	void usart_data_transmit(uint32_t usart_periph, uint16_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>	usart peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>data</b>	data of transmission
0-0x01FF	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 3-493. Function usart\_data\_receive

<b>Function name</b>	usart_data_receive
<b>Function prototype</b>	void usart_data_receive(uint32_t usart_periph);
<b>Function descriptions</b>	USART receive data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	data of received (0-0x1FF)

Example:

```
/* USART0 receive data */
```

```
uint16_t temp;
```

```
temp = usart_data_receive(USART0);
```

### usart\_address\_config

The description of usart\_address\_config is shown as below:

**Table 3-494. 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 terminal
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>addr</b>	address of USART
0-0xFF	address of USART
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure address of the USART0 */
```

```
usart_address_config(USART0, 0x00);
```

### usart\_mute\_mode\_enable

The description of usart\_mute\_mode\_enable is shown as below:

**Table 3-495. Function usart\_mute\_mode\_enable**

<b>Function name</b>	usart_mute_mode_enable
<b>Function prototype</b>	void usart_mute_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable mute mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
UARTx	x=3,4



Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 receiver in mute mode */
usart_mute_mode_enable(USART0);
```

### usart\_mute\_mode\_disable

The description of usart\_mute\_mode\_disable is shown as below:

**Table 3-496. Function usart\_mute\_mode\_disable**

Function name	usart_mute_mode_disable
Function prototype	void usart_mute_mode_disable(uint32_t usart_periph);
Function descriptions	disable mute mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2
UARTx	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 receiver in mute mode */
usart_mute_mode_disable(USART0);
```

### usart\_mute\_mode\_wakeup\_config

The description of usart\_mute\_mode\_wakeup\_config is shown as below:

**Table 3-497. Function usart\_mute\_mode\_wakeup\_config**

Function name	usart_mute_mode_wakeup_config
Function prototype	void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);
Function descriptions	configure wakeup method in mute mode
Precondition	-
The called functions	-

Input parameter{in}	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
Input parameter{in}	
<b>wmethod</b>	two methods be used to enter or exit the mute mode
<i>USART_WM_IDLE</i>	idle line
<i>USART_WM_ADDR</i>	address mask
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 wakeup method in mute mode */
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

### usart\_lin\_mode\_enable

The description of usart\_lin\_mode\_enable is shown as below:

**Table 3-498. Function usart\_lin\_mode\_enable**

<b>Function name</b>	usart_lin_mode_enable
<b>Function prototype</b>	void usart_lin_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable LIN mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 LIN mode enable */
usart_lin_mode_enable(USART0);
```

### usart\_lin\_mode\_disable

The description of usart\_lin\_mode\_disable is shown as below:

Table 3-499. Function `usart_lin_mode_disable`

Function name	<code>usart_lin_mode_disable</code>
Function prototype	<code>void usart_lin_mode_disable(uint32_t usart_periph);</code>
Function descriptions	disable LIN mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	usart peripheral
<code>USARTx</code>	x=0,1,2
<code>UARTx</code>	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 LIN mode disable */
```

```
usart_lin_mode_disable(USART0);
```

### `usart_lin_break_dection_length_config`

The description of `usart_lin_break_dection_length_config` is shown as below:

Table 3-500. Function `usart_lin_break_dection_length_config`

Function name	<code>usart_lin_break_dection_length_config</code>
Function prototype	<code>void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lblen);</code>
Function descriptions	configure lin break frame length
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	usart peripheral
<code>USARTx</code>	x=0,1,2
<code>UARTx</code>	x=3,4
Input parameter{in}	
<code>lblen</code>	two methods be used to enter or exit the mute mode
<code>USART_LBLEN_10B</code>	10 bits
<code>USART_LBLEN_11B</code>	11 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure LIN break frame length */
```

```
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

### usart\_send\_break

The description of usart\_send\_break is shown as below:

**Table 3-501. Function usart\_send\_break**

<b>Function name</b>	usart_send_break
<b>Function prototype</b>	void usart_send_break(uint32_t usart_periph);
<b>Function descriptions</b>	send break frame
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 send break frame */
```

```
usart_send_break(USART0);
```

### usart\_halfduplex\_enable

The description of usart\_halfduplex\_enable is shown as below:

**Table 3-502. Function usart\_halfduplex\_enable**

<b>Function name</b>	usart_halfduplex_enable
<b>Function prototype</b>	void usart_halfduplex_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable half-duplex mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* enable USART0 half duplex mode*/
```

```
usart_halfduplex_enable(USART0);
```

### usart\_halfduplex\_disable

The description of usart\_halfduplex\_disable is shown as below:

**Table 3-503. Function usart\_halfduplex\_disable**

Function name	usart_halfduplex_disable
Function prototype	void usart_halfduplex_disable(uint32_t usart_periph);
Function descriptions	disable half-duplex mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2
UARTx	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 half duplex mode*/
```

```
usart_halfduplex_disable(USART0);
```

### usart\_synchronous\_clock\_enable

The description of usart\_synchronous\_clock\_enable is shown as below:

**Table 3-504. Function usart\_synchronous\_clock\_enable**

Function name	usart_synchronous_clock_enable
Function prototype	void usart_synchronous_clock_enable(uint32_t usart_periph);
Function descriptions	enable CK pin in synchronous mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 CK pin in synchronous mode */
```

```
usart_synchronous_clock_enable(USART0);
```

### usart\_synchronous\_clock\_disable

The description of usart\_synchronous\_clock\_disable is shown as below:

**Table 3-505. Function usart\_synchronous\_clock\_disable**

Function name	usart_synchronous_clock_disable
Function prototype	void usart_synchronous_clock_disable(uint32_t usart_periph);
Function descriptions	disable CK pin in synchronous mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 CK pin in synchronous mode */
```

```
usart_synchronous_clock_disable(USART0);
```

### usart\_synchronous\_clock\_config

The description of usart\_synchronous\_clock\_config is shown as below:

**Table 3-506. Function usart\_synchronous\_clock\_config**

Function name	usart_synchronous_clock_config
Function prototype	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
Function descriptions	configure USART synchronous mode parameters
Precondition	-
The called functions	-
Input parameter{in}	

<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>clen</b>	last bit clock pulse
<i>USART_CLEN_NONE</i>	clock pulse of the last data bit (MSB) is not output to the CK pin
<i>USART_CLEN_EN</i>	clock pulse of the last data bit (MSB) is output to the CK pin
<b>Input parameter{in}</b>	
<b>cph</b>	clock phase
<i>USART_CPH_1CK</i>	first clock transition is the first data capture edge
<i>USART_CPH_2CK</i>	second clock transition is the first data capture edge
<b>Input parameter{in}</b>	
<b>cpl</b>	clock polarity
<i>USART_CPL_LOW</i>	steady low value on CK pin
<i>USART_CPL_HIGH</i>	steady high value on CK pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 synchronous mode parameters */
```

```
usart_synchronous_clock_config(USART0,USART_CLEN_EN,USART_CPH_2CK,  
USART_CPL_HIGH);
```

### usart\_guard\_time\_config

The description of usart\_guard\_time\_config is shown as below:

**Table 3-507. 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,uint8_t guat);
<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>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>guat</b>	guard time value
<i>0-0xFF</i>	guard time value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* configure USART0 guard time value in smartcard mode */
```

```
usart_guard_time_config(USART0, 0x55);
```

### usart\_smartcard\_mode\_enable

The description of usart\_smartcard\_mode\_enable is shown as below:

**Table 3-508. 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>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 smartcard mode enable */
```

```
usart_smartcard_mode_enable(USART0);
```

### usart\_smartcard\_mode\_disable

The description of usart\_smartcard\_mode\_disable is shown as below:

**Table 3-509. Function usart\_smartcard\_mode\_disable**

<b>Function name</b>	usart_smartcard_mode_disable
<b>Function prototype</b>	void usart_smartcard_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-



Return value	
-	-

Example:

```
/* USART0 smartcard mode disable */
usart_smartcard_mode_disable(USART0);
```

### usart\_smartcard\_mode\_nack\_enable

The description of usart\_smartcard\_mode\_nack\_enable is shown as below:

**Table 3-510. Function usart\_smartcard\_mode\_nack\_enable**

Function name	usart_smartcard_mode_nack_enable
Function prototype	void usart_smartcard_mode_nack_enable(uint32_t usart_periph);
Function descriptions	enable NACK in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_enable(USART0);
```

### usart\_smartcard\_mode\_nack\_disable

The description of usart\_smartcard\_mode\_nack\_disable is shown as below:

**Table 3-511. Function usart\_smartcard\_mode\_nack\_disable**

Function name	usart_smartcard_mode_nack_disable
Function prototype	void usart_smartcard_mode_nack_disable(uint32_t usart_periph);
Function descriptions	disable NACK in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_disable(USART0);
```

### usart\_irda\_mode\_enable

The description of usart\_irda\_mode\_enable is shown as below:

**Table 3-512. 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>	usart peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 IrDA mode */
usart_irda_mode_enable(USART0);
```

### usart\_irda\_mode\_disable

The description of usart\_irda\_mode\_disable is shown as below:

**Table 3-513. 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>	usart peripheral

USARTx	x=0,1,2
UARTx	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 IrDA mode */
usart_irda_mode_disable(USART0);
```

### usart\_prescaler\_config

The description of usart\_prescaler\_config is shown as below:

**Table 3-514. Function usart\_prescaler\_config**

Function name	usart_prescaler_config
Function prototype	void usart_prescaler_config(uint32_t usart_periph, uint8_t psc);
Function descriptions	configure the peripheral clock prescaler in USART IrDA low-power mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2
UARTx	x=3,4
Input parameter{in}	
psc	clock prescaler
0x00-0xFF	clock prescaler
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the USART0 peripheral clock prescaler in USART IrDA low-power mode */
usart_prescaler_config(USART0, 0x00);
```

### usart\_irda\_lowpower\_config

The description of usart\_irda\_lowpower\_config is shown as below:

**Table 3-515. Function usart\_irda\_lowpower\_config**

Function name	usart_irda_lowpower_config
---------------	----------------------------

<b>Function prototype</b>	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
<b>Function descriptions</b>	configure IrDA low-power
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>irlp</b>	IrDA low-power or normal
USART_IRLP_LOW	low-power
USART_IRLP_NORMAL	normal
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 IrDA low-power */
```

```
usart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

### usart\_hardware\_flow\_rts\_config

The description of usart\_hardware\_flow\_rts\_config is shown as below:

**Table 3-516. 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>	usart peripheral
USARTx	x=0,1,2
<b>Input parameter{in}</b>	
<b>rtsconfig</b>	enable or disable RTS
USART_RTS_ENABLE	enable RTS
USART_RTS_DISABLE	disable RTS
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* configure USART0 hardware flow control RTS */
```

```
usart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);
```

### usart\_hardware\_flow\_cts\_config

The description of usart\_hardware\_flow\_cts\_config is shown as below:

**Table 3-517. Function usart\_hardware\_flow\_cts\_config**

Function name	usart_hardware_flow_cts_config
Function prototype	void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
Function descriptions	configure hardware flow control RTS
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2
Input parameter{in}	
ctsconfig	enable or disable CTS
USART_CTS_ENABLE	enable CTS
USART_CTS_DISABLE	disable CTS
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 hardware flow control CTS */
```

```
usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

### usart\_dma\_receive\_config

The description of usart\_dma\_receive\_config is shown as below:

**Table 3-518. Function usart\_dma\_receive\_config**

Function name	usart_dma_receive_config
Function prototype	void usart_dma_receive_config(uint32_t usart_periph, uint8_t dmaconfig);
Function descriptions	configure USART DMA reception

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3
<b>Input parameter{in}</b>	
<b>dmaconfig</b>	enable or disable DMA for reception
<i>USART_RECEIVE_DMA_ENABLE</i>	DMA enable for reception
<i>USART_RECEIVE_DMA_DISABLE</i>	DMA disable for reception
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 DMA enable for reception */
```

```
usart_dma_receive_config(USART0, USART_RECEIVE_DMA_ENABLE);
```

### usart\_dma\_transmit\_config

The description of usart\_dma\_transmit\_config is shown as below:

**Table 3-519. Function usart\_dma\_transmit\_config**

<b>Function name</b>	usart_dma_transmit_config
<b>Function prototype</b>	void usart_dma_transmit_config(uint32_t usart_periph, uint8_t dmaconfig);
<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>	usart peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3
<b>Input parameter{in}</b>	
<b>dmaconfig</b>	enable or disable DMA for transmission
<i>USART_TRANSMIT_DMA_ENABLE</i>	DMA enable for transmission
<i>USART_TRANSMIT_DMA_DISABLE</i>	DMA disable for transmission
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* USART0 DMA enable for transmission */
```

```
usart_dma_transmit_config(USART0, USART_TRANSMIT_DMA_ENABLE);
```

### usart\_flag\_get

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

**Table 3-520. Function usart\_flag\_get**

Function name	usart_flag_get
Function prototype	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
Function descriptions	get flag in STAT register
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2
UARTx	x=3,4
Input parameter{in}	
flag	USART flags, refer to usart_flag_enum only one among these parameters can be selected
USART_FLAG_PERR	parity error flag
USART_FLAG_FERR	frame error flag
USART_FLAG_NERR	noise error flag
USART_FLAG_ORER R	overrun error
USART_FLAG_IDLE	idle line detected flag
USART_FLAG_RBNE	read data buffer not empty
USART_FLAG_TC	transmission completed
USART_FLAG_TBE	transmit data register empty
USART_FLAG_LBD	LIN break detected flag
USART_FLAG_CTS	CTS change flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get flag USART0 state */
```

```
FlagStatus status;
```

```
status = usart_flag_get(USART0,USART_FLAG_TBE);
```

## usart\_flag\_clear

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

**Table 3-521. 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 STAT register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags, refer to usart_flag_enum only one among these parameters can be selected
USART_FLAG_TC	transmission complete flag
USART_FLAG_LBD	LIN break detected flag
USART_FLAG_CTS	CTS change flag
USART_FLAG_RBNE	read data buffer not empty
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear USART0 flag */
```

```
usart_flag_clear(USART0,USART_FLAG_TC);
```

## usart\_interrupt\_enable

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

**Table 3-522. Function usart\_interrupt\_enable**

<b>Function name</b>	usart_interrupt_enable
<b>Function prototype</b>	void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);
<b>Function descriptions</b>	enable USART interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	



<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt type
<i>USART_INT_IDLE</i>	idle interrupt
<i>USART_INT_RBNE</i>	read data buffer not empty interrupt and overrun error interrupt enable interrupt
<i>USART_INT_TC</i>	transmission complete interrupt
<i>USART_INT_TBE</i>	transmit data register empty interrupt
<i>USART_INT_PERR</i>	parity error interrupt
<i>USART_INT_LBD</i>	LIN break detection interrupt
<i>USART_INT_ERR</i>	error interrupt enable in multibuffer communication
<i>USART_INT_CTS</i>	CTS interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 TBE interrupt */
```

```
usart_interrupt_enable(USART0, USART_INT_TBE);
```

### usart\_interrupt\_disable

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

**Table 3-523. Function usart\_interrupt\_disable**

<b>Function name</b>	usart_interrupt_disable
<b>Function prototype</b>	void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);
<b>Function descriptions</b>	disable USART interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>interrupt</b>	USART interrupt flag
<i>USART_INT_IDLE</i>	idle interrupt
<i>USART_INT_RBNE</i>	read data buffer not empty interrupt and overrun error interrupt enable interrupt

<i>USART_INT_TC</i>	transmission complete interrupt
<i>USART_INT_TBE</i>	transmit data register empty interrupt
<i>USART_INT_PERR</i>	parity error interrupt
<i>USART_INT_LBD</i>	LIN break detection interrupt
<i>USART_INT_ERR</i>	error interrupt enable in multibuffer communication
<i>USART_INT_CTS</i>	CTS interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 TBE interrupt */
```

```
usart_interrupt_disable(USART0, USART_INT_TBE);
```

### usart\_interrupt\_flag\_get

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

**Table 3-524. Function usart\_interrupt\_flag\_get**

<b>Function name</b>	usart_interrupt_flag_get
<b>Function prototype</b>	FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	get USART interrupt and flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag
<i>USART_INT_FLAG_PERRR</i>	parity error interrupt and flag
<i>USART_INT_FLAG_TBRE</i>	transmitter buffer empty interrupt and flag
<i>USART_INT_FLAG_TCR</i>	transmission complete interrupt and flag
<i>USART_INT_FLAG_RBNE</i>	read data buffer not empty interrupt and flag
<i>USART_INT_FLAG_RBNE_ORERR</i>	read data buffer not empty interrupt and overrun error flag
<i>USART_INT_FLAG_IDLELE</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_NERR</i>	error interrupt and noise error flag
<i>USART_INT_FLAG_ER R_ORERR</i>	error interrupt and overrun error
<i>USART_INT_FLAG_ER R_FERR</i>	error interrupt and frame error 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 3-525. 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 flag);
<b>Function descriptions</b>	clear USART interrupt flag in STAT register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>flag</b>	USART interrupt flag
<i>USART_INT_FLAG_TC</i>	transmission complete flag
<i>USART_INT_FLAG_LB D</i>	LIN break detected flag
<i>USART_INT_FLAG_CT S</i>	CTS change flag
<i>USART_INT_FLAG_RB</i>	read buffer not empty flag

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

Example:

```
/* clear the USART0 interrupt flag */
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_TC);
```

## 3.22. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.22.1](#), the WWDGT firmware functions are introduced in chapter [3.22.2](#).

### 3.22.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

**Table 3-526. WWDGT Registers**

Registers	Descriptions
WWDGT_CTL	Control register
WWDGT_CFG	Configuration register
WWDGT_STAT	Status register

### 3.22.2. Descriptions of Peripheral functions

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

**Table 3-527. WWDGT firmware function**

Function name	Function description
wwdgt_deinit	reset the WWDGT counter configuration
wwdgt_enable	start the WWDGT counter
wwdgt_counter_update	configure the WWDGT 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 3-528. Function wwdgt\_deinit**

<b>Function name</b>	wwdgt_deinit
<b>Function prototype</b>	void wwdgt_deinit(void);
<b>Function descriptions</b>	reset the WWDGT counter configuration
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset the WWDGT configuration */
```

```
wwdgt_deinit ( );
```

## wwdgt\_enable

The description of wwdgt\_enable is shown as below:

**Table 3-529. Function wwdgt\_enable**

<b>Function name</b>	wwdgt_enable
<b>Function prototype</b>	void wwdgt_enable (void);
<b>Function descriptions</b>	start the WWDGT counter
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* start the WWDGT counter */
```

```
wwdgt_enable ( );
```

## wwdgt\_counter\_update

The description of wwdgt\_counter\_update is shown as below:

Table 3-530. Function `wwdgt_counter_update`

<b>Function name</b>	<code>wwdgt_counter_update</code>
<b>Function prototype</b>	<code>void wwdgt_counter_update(uint16_t counter_value);</code>
<b>Function descriptions</b>	configure the WWDGT counter value
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>counter_value</b>	counter_value: 0x00000000 - 0x0000007F
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* update WWDGT counter to 0x7F */
```

```
wwdgt_counter_update(127);
```

### **wwdgt\_config**

The description of `wwdgt_config` is shown as below:

Table 3-531. Function `wwdgt_config`

<b>Function name</b>	<code>wwdgt_config</code>
<b>Function prototype</b>	<code>void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler);</code>
<b>Function descriptions</b>	configure counter value, window value, and prescaler divider value
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>counter</b>	counter: 0x00000000 - 0x0000007F
<b>Input parameter{in}</b>	
<b>window</b>	window: 0x00000000 - 0x0000007F
<b>Input parameter{in}</b>	
<b>prescaler</b>	wwdgt prescaler value
<code>WWDGT_CFG_PSC_D</code> <code>IV1</code>	the time base of WWDGT counter = (PCLK1/4096)/1
<code>WWDGT_CFG_PSC_D</code> <code>IV2</code>	the time base of WWDGT counter = (PCLK1/4096)/2
<code>WWDGT_CFG_PSC_D</code> <code>IV4</code>	the time base of WWDGT counter = (PCLK1/4096)/4
<code>WWDGT_CFG_PSC_D</code> <code>IV8</code>	the time base of WWDGT counter = (PCLK1/4096)/8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-532. 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>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 3-533. 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>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 3-534. Function wwdgt\_flag\_clear**

<b>Function name</b>	wwdgt_flag_clear
<b>Function prototype</b>	void wwdgt_flag_clear(void);
<b>Function descriptions</b>	clear early wakeup interrupt state of WWDGT
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* clear early wakeup interrupt state of WWDGT */

wwdgt_flag_clear( );

```



## 4. Revision history

**Table 4-1. Revision history**

Revision No.	Description	Date
1.0	Initial Release	Sept.27, 2019
1.1	I2C and SPI with few changes	Jul.12, 2022
1.2	EXTI \ ECLIC \ SPI \ TIMER with few changes	Jan.3, 2023
1.3	DAC with few changes	Jan.5, 2024

## Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company under the intellectual property laws and treaties of the People's Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as its suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.