

**GigaDevice Semiconductor Inc.**

**GD32F1x0**

**ARM® Cortex™-M3 32-bit MCU**

**Firmware Library  
User Guide**

Revision 1.3

(Jan. 2026)

# 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>22</b>
<b>1.1. Rules of User Manual and Firmware Library .....</b>	<b>22</b>
1.1.1. Peripherals.....	22
1.1.2. Naming rules.....	23
<b>2. Firmware Library Overview .....</b>	<b>24</b>
<b>2.1. File Structure of Firmware Library .....</b>	<b>24</b>
2.1.1. Examples Folder .....	25
2.1.2. Firmware Folder.....	25
2.1.3. Template Folder .....	25
2.1.4. Utilities Folder .....	28
<b>2.2. File descriptions of Firmware Library .....</b>	<b>30</b>
<b>3. Firmware Library of Standard Peripherals .....</b>	<b>31</b>
<b>3.1. Overview of Firmware Library of Standard Peripherals.....</b>	<b>31</b>
<b>3.2. ADC .....</b>	<b>31</b>
3.2.1. Descriptions of Peripheral registers.....	31
3.2.2. Descriptions of Peripheral functions .....	32
<b>3.3. CEC.....</b>	<b>52</b>
3.3.1. Descriptions of Peripheral registers.....	53
3.3.2. Descriptions of Peripheral functions .....	53
<b>3.4. CMP .....</b>	<b>70</b>
3.4.1. Descriptions of Peripheral registers.....	70
3.4.2. Descriptions of Peripheral functions .....	70
<b>3.5. CRC .....</b>	<b>78</b>
3.5.1. Descriptions of Peripheral registers.....	78
3.5.2. Descriptions of Peripheral functions .....	78
<b>3.6. DAC .....</b>	<b>85</b>
3.6.1. Peripheral register description .....	85
3.6.2. Descriptions of Peripheral functions .....	85
<b>3.7. DBG .....</b>	<b>97</b>
3.7.1. Descriptions of Peripheral registers.....	98
3.7.2. Descriptions of Peripheral functions .....	98

<b>3.8. DMA</b>	<b>102</b>
3.8.1. Descriptions of Peripheral registers	102
3.8.2. Descriptions of Peripheral functions	103
<b>3.9. EXTI</b>	<b>121</b>
3.9.1. Descriptions of Peripheral registers	121
3.9.2. Descriptions of Peripheral functions	122
<b>3.10. FMC</b>	<b>129</b>
3.10.1. Descriptions of Peripheral registers	129
3.10.2. Descriptions of Peripheral functions	130
<b>3.11. FWDGT</b>	<b>148</b>
3.11.1. Descriptions of Peripheral registers	148
3.11.2. Descriptions of Peripheral functions	148
<b>3.12. GPIO</b>	<b>153</b>
3.12.1. Descriptions of Peripheral registers	153
3.12.2. Descriptions of Peripheral functions	154
<b>3.13. I2C</b>	<b>163</b>
3.13.1. Descriptions of Peripheral registers	163
3.13.2. Descriptions of Peripheral functions	164
<b>3.14. MISC</b>	<b>186</b>
3.14.1. Descriptions of Peripheral registers	186
3.14.2. Descriptions of Peripheral functions	187
<b>3.15. PMU</b>	<b>193</b>
3.15.1. Descriptions of Peripheral registers	193
3.15.2. Descriptions of Peripheral functions	193
<b>3.16. RCU</b>	<b>200</b>
3.16.1. Descriptions of Peripheral registers	201
3.16.2. Descriptions of Peripheral functions	201
<b>3.17. RTC</b>	<b>237</b>
3.17.1. Descriptions of Peripheral registers	237
3.17.2. Descriptions of Peripheral functions	238
<b>3.18. SLCD</b>	<b>258</b>
3.18.1. Descriptions of Peripheral registers	258
3.18.2. Descriptions of Peripheral functions	258
<b>3.19. SPI/I2S</b>	<b>273</b>
3.19.1. Descriptions of Peripheral registers	273
3.19.2. Descriptions of Peripheral functions	273
<b>3.20. SYSCFG</b>	<b>294</b>
3.20.1. Descriptions of Peripheral registers	294
3.20.2. Descriptions of Peripheral functions	295

<b>3.21. TIMER.....</b>	<b>301</b>
3.21.1. Descriptions of Peripheral registers.....	302
3.21.2. Descriptions of Peripheral functions .....	303
<b>3.22. TSI .....</b>	<b>360</b>
3.22.1. Descriptions of Peripheral registers.....	360
3.22.2. Descriptions of Peripheral functions .....	360
<b>3.23. USART.....</b>	<b>381</b>
3.23.1. Descriptions of Peripheral registers.....	381
3.23.2. Descriptions of Peripheral functions .....	382
<b>3.24. WWDGT.....</b>	<b>424</b>
3.24.1. Descriptions of Peripheral registers.....	425
3.24.2. Descriptions of Peripheral functions .....	425
<b>4. Revision history.....</b>	<b>430</b>

## List of Figures

Figure 2-1. File structure of firmware library of GD32F1x0 .....	24
Figure 2-2. Select peripheral example files .....	26
Figure 2-3. Copy the peripheral example files .....	27
Figure 2-4. Open the project file .....	27
Figure 2-5. Configure project files .....	28
Figure 2-6. Compile-debug-download .....	28

## List of Tables

Table 1-1. Peripherals .....	22
Table 2-1. Function descriptions of Firmware Library .....	30
Table 3-1. Peripheral function format of Firmware Library .....	31
Table 3-2. ADC Registers .....	31
Table 3-3. ADC firmware function.....	32
Table 3-4. Function adc_deinit.....	33
Table 3-5. Function adc_enable .....	33
Table 3-6. Function adc_disable .....	34
Table 3-7. Function adc_calibration_enable.....	34
Table 3-8. Function adc_dma_mode_enable .....	35
Table 3-9. Function adc_dma_mode_disable.....	35
Table 3-10. Function adc_tempsensor_vrefint_enable.....	36
Table 3-11. Function adc_tempsensor_vrefint_disable.....	36
Table 3-12. Function adc_vbat_enable .....	37
Table 3-13. Function adc_vbat_disable .....	37
Table 3-14. Function adc_discontinuous_mode_config .....	38
Table 3-15. Function adc_special_function_config .....	38
Table 3-16. Function adc_data_alignment_config.....	39
Table 3-17. Function adc_channel_length_config.....	40
Table 3-18. Function adc_regular_channel_config .....	40
Table 3-19. Function adc_inserted_channel_config .....	41
Table 3-20. Function adc_inserted_channel_offset_config.....	42
Table 3-21. Function adc_external_trigger_config.....	43
Table 3-22. Function adc_external_trigger_source_config .....	44
Table 3-23. Function adc_software_trigger_enable .....	45
Table 3-24. Function adc_regular_data_read.....	46
Table 3-25. Function adc_inserted_data_read .....	46
Table 3-26. Function adc_flag_get .....	47
Table 3-27. Function adc_flag_clear .....	47
Table 3-28. Function adc_interrupt_flag_get.....	48
Table 3-29. Function adc_interrupt_flag_clear .....	49
Table 3-30. Function adc_interrupt_enable .....	49
Table 3-31. Function adc_interrupt_disable .....	50
Table 3-32. Function adc_watchdog_single_channel_enable.....	50
Table 3-33. Function adc_watchdog_group_channel_enable.....	51
Table 3-34. Function adc_watchdog_disable .....	51
Table 3-35. Function adc_watchdog_threshold_config .....	52
Table 3-36. CEC Registers .....	53
Table 3-37. CEC firmware function.....	53
Table 3-38. Function cec_deinit.....	54

Table 3-39. Function cec_init .....	54
Table 3-40. Function cec_error_config .....	55
Table 3-41. Function cec_enable .....	57
Table 3-42. Function cec_disable .....	57
Table 3-43. Function cec_transmission_start .....	58
Table 3-44. Function cec_transmission_end .....	58
Table 3-45. Function cec_listen_mode_enable .....	59
Table 3-46. Function cec_listen_mode_disable .....	59
Table 3-47. Function cec_own_address_config .....	60
Table 3-48. Function cec_sft_config .....	60
Table 3-49. Function cec_generate_errorbit_config .....	61
Table 3-50. Function cec_stop_receive_bre_config .....	62
Table 3-51. Function cec_reception_tolerance_enable .....	63
Table 3-52. Function cec_reception_tolerance_disable .....	63
Table 3-53. Function cec_data_send .....	64
Table 3-54. Function cec_data_receive .....	64
Table 3-55. Function cec_flag_get .....	65
Table 3-56. Function cec_flag_clear .....	65
Table 3-57. Function cec_interrupt_enable .....	66
Table 3-58. Function cec_interrupt_disable .....	67
Table 3-59. Function cec_interrupt_flag_get .....	68
Table 3-60. Function cec_interrupt_flag_clear .....	69
Table 3-61. CMP registers .....	70
Table 3-62. CMP firmware function .....	70
Table 3-63. Enum cmp_enum .....	70
Table 3-64. Function cmp_deinit .....	71
Table 3-65. Function cmp_mode_init .....	71
Table 3-66. Function cmp_output_init .....	72
Table 3-67. Function cmp_enable .....	74
Table 3-68. Function cmp_disable .....	74
Table 3-69. Function cmp_switch_enable .....	75
Table 3-70. Function cmp_switch_disable .....	75
Table 3-71. Function cmp_window_enable .....	76
Table 3-72. Function cmp_window_disable .....	76
Table 3-73. Function cmp_lock_enable .....	77
Table 3-74. Function cmp_output_level_get .....	77
Table 3-75. CRC Registers .....	78
Table 3-76. CRC firmware function .....	78
Table 3-77. Function crc_deinit .....	79
Table 3-78. Function crc_reverse_output_data_enable .....	79
Table 3-79. Function crc_reverse_output_data_disable .....	80
Table 3-80. Function crc_data_register_reset .....	80
Table 3-81. Function crc_data_register_read .....	81

Table 3-82. Function <code>crc_free_data_register_read</code> .....	81
Table 3-83. Function <code>crc_free_data_register_write</code> .....	82
Table 3-84. Function <code>crc_init_data_register_write</code> .....	82
Table 3-85. Function <code>crc_input_data_reverse_config</code> .....	83
Table 3-86. Function <code>crc_single_data_calculate</code> .....	83
Table 3-87. Function <code>crc_block_data_calculate</code> .....	84
Table 3-88. DAC Registers .....	85
Table 3-89. DAC firmware functions .....	85
Table 3-90. Function <code>dac_deinit</code> .....	86
Table 3-91. Function <code>dac_enable</code> .....	86
Table 3-92. Function <code>dac_disable</code> .....	87
Table 3-93. Function <code>dac_dma_enable</code> .....	87
Table 3-94. Function <code>dac_dma_disable</code> .....	88
Table 3-95. Function <code>dac_output_buffer_enable</code> .....	89
Table 3-96. Function <code>dac_output_buffer_disable</code> .....	89
Table 3-97. Function <code>dac_output_value_get</code> .....	90
Table 3-98. Function <code>dac_data_set</code> .....	90
Table 3-99. Function <code>dac_trigger_enable</code> .....	91
Table 3-100. Function <code>dac_trigger_disable</code> .....	92
Table 3-101. Function <code>dac_trigger_source_config</code> .....	92
Table 3-102. Function <code>dac_software_trigger_enable</code> .....	93
Table 3-103. Function <code>dac_flag_get</code> .....	94
Table 3-104. Function <code>dac_flag_clear</code> .....	94
Table 3-105. Function <code>dac_interrupt_enable</code> .....	95
Table 3-106. Function <code>dac_interrupt_disable</code> .....	96
Table 3-107. Function <code>dac_interrupt_flag_get</code> .....	96
Table 3-108. Function <code>dac_interrupt_flag_clear</code> .....	97
Table 3-109. DBG Registers.....	98
Table 3-110. DBG firmware function .....	98
Table 3-111. Enum <code>dbg_periph_enum</code> .....	98
Table 3-112. Function <code>dbg_deinit</code> .....	99
Table 3-113. Function <code>dbg_id_get</code> .....	99
Table 3-114. Function <code>dbg_low_power_enable</code> .....	100
Table 3-115. Function <code>dbg_low_power_disable</code> .....	100
Table 3-116. Function <code>dbg_periph_enable</code> .....	101
Table 3-117. Function <code>dbg_periph_disable</code> .....	101
Table 3-118. DMA Registers .....	102
Table 3-119. DMA firmware function .....	103
Table 3-120. Enum <code>dma_channel_enum</code> .....	103
Table 3-121. Structure <code>dma_parameter_struct</code> .....	104
Table 3-122. Function <code>dma_deinit</code> .....	104
Table 3-123. Function <code>dma_para_init</code> .....	105
Table 3-124. Function <code>dma_init</code> .....	105



Table 3-125. Function dma_circulation_enable .....	106
Table 3-126. Function dma_circulation_disable .....	107
Table 3-127. Function dma_memory_to_memory_enable .....	107
Table 3-128. Function dma_memory_to_memory_disable .....	108
Table 3-129. Function dma_channel_enable .....	108
Table 3-130. Function dma_channel_disable .....	109
Table 3-131. Function dma_periph_address_config .....	109
Table 3-132. Function dma_memory_address_config .....	110
Table 3-133. Function dma_transfer_number_config .....	111
Table 3-134. Function dma_transfer_number_get .....	111
Table 3-135. Function dma_priority_config .....	112
Table 3-136. Function dma_memory_width_config .....	113
Table 3-137. Function dma_periph_width_config .....	113
Table 3-138. Function dma_memory_increase_enable .....	114
Table 3-139. Function dma_memory_increase_disable .....	115
Table 3-140. Function dma_periph_increase_enable .....	115
Table 3-141. Function dma_periph_increase_disable .....	116
Table 3-142. Function dma_transfer_direction_config .....	116
Table 3-143. Function dma_flag_get .....	117
Table 3-144. Function dma_flag_clear .....	118
Table 3-145. Function dma_interrupt_enable .....	118
Table 3-146. Function dma_interrupt_disable .....	119
Table 3-147. Function dma_interrupt_flag_get .....	120
Table 3-148. Function dma_interrupt_flag_clear .....	120
Table 3-149. EXTI Registers .....	121
Table 3-150. EXTI firmware function .....	122
Table 3-151. Enum exti_line_enum .....	122
Table 3-152. Enum exti_mode_enum .....	123
Table 3-153. Enum exti_trig_type_enum .....	123
Table 3-154. Function exti_deinit .....	123
Table 3-155. Function exti_init .....	124
Table 3-156. Function exti_interrupt_enable .....	124
Table 3-157. Function exti_interrupt_disable .....	125
Table 3-158. Function exti_event_enable .....	125
Table 3-159. Function exti_event_disable .....	126
Table 3-160. Function exti_software_interrupt_enable .....	126
Table 3-161. Function exti_software_interrupt_disable .....	127
Table 3-162. Function exti_flag_get .....	127
Table 3-163. Function exti_flag_clear .....	128
Table 3-164. Function exti_interrupt_flag_get .....	128
Table 3-165. Function exti_interrupt_flag_clear .....	129
Table 3-166. FMC Registers .....	129
Table 3-167. FMC firmware function .....	130

Table 3-168. Enum <code>fmc_state_enum</code> .....	130
Table 3-169. Function <code>fmc_unlock</code> .....	131
Table 3-170. Function <code>fmc_lock</code> .....	131
Table 3-171. Function <code>fmc_wscnt_set</code> .....	132
Table 3-172. Function <code>fmc_wscnt_set</code> .....	132
Table 3-173. Function <code>fmc_wscnt_set</code> .....	133
Table 3-174. Function <code>fmc_page_erase</code> .....	133
Table 3-175. Function <code>fmc_mass_erase</code> .....	134
Table 3-176. Function <code>fmc_word_program</code> .....	135
Table 3-177. Function <code>fmc_halfword_program</code> .....	135
Table 3-178. Function <code>fmc_word_reprogram</code> .....	136
Table 3-179. Function <code>ob_unlock</code> .....	137
Table 3-180. Function <code>ob_lock</code> .....	137
Table 3-181. Function <code>ob_reset</code> .....	138
Table 3-182. Function <code>ob_erase</code> .....	138
Table 3-183. Function <code>ob_write_protection_enable</code> .....	139
Table 3-184. Function <code>ob_security_protection_config</code> .....	140
Table 3-185. Function <code>ob_user_write</code> .....	141
Table 3-186. Function <code>ob_data_program</code> .....	142
Table 3-187. Function <code>ob_user_get</code> .....	142
Table 3-188. Function <code>ob_data_get</code> .....	143
Table 3-189. Function <code>ob_write_protection_get</code> .....	143
Table 3-190. Function <code>ob_obstat_plevel_get</code> .....	144
Table 3-191. Function <code>fmc_flag_get</code> .....	144
Table 3-192. Function <code>fmc_flag_clear</code> .....	145
Table 3-193. Function <code>fmc_interrupt_enable</code> .....	145
Table 3-194. Function <code>fmc_interrupt_disable</code> .....	146
Table 3-195. Function <code>fmc_interrupt_flag_get</code> .....	147
Table 3-196. Function <code>fmc_interrupt_flag_clear</code> .....	147
Table 3-197. FWDGT Registers .....	148
Table 3-198. FWDGT firmware function .....	148
Table 3-199. Function <code> fwdgt_write_enable</code> .....	149
Table 3-200. Function <code> fwdgt_write_disable</code> .....	149
Table 3-201. Function <code> fwdgt_enable</code> .....	149
Table 3-202. Function <code> fwdgt_prescaler_value_config</code> .....	150
Table 3-203. Function <code> fwdgt_reload_value_config</code> .....	151
Table 3-204. Function <code> fwdgt_window_value_config</code> .....	151
Table 3-205. Function <code> fwdgt_counter_reload</code> .....	152
Table 3-206. Function <code> fwdgt_config</code> .....	152
Table 3-207. Function <code> fwdgt_flag_get</code> .....	153
Table 3-208. GPIO Registers .....	153
Table 3-209. GPIO firmware function .....	154
Table 3-210. Function <code> gpio_deinit</code> .....	154

Table 3-211. Function gpio_mode_set.....	155
Table 3-212. Function gpio_output_options_set .....	156
Table 3-213. Function gpio_bit_set .....	157
Table 3-214. Function gpio_bit_reset .....	157
Table 3-215. Function gpio_bit_write.....	158
Table 3-216. Function gpio_port_write .....	159
Table 3-217. Function gpio_input_bit_get.....	159
Table 3-218. Function gpio_input_port_get.....	160
Table 3-219. Function gpio_output_bit_get .....	160
Table 3-220. Function gpio_output_port_get .....	161
Table 3-221. Function gpio_af_set .....	162
Table 3-222. Function gpio_pin_lock .....	163
Table 3-223. I2C Registers .....	163
Table 3-224. I2C firmware function.....	164
Table 3-225. Enum i2c_flag_enum.....	165
Table 3-226. Enum i2c_interrupt_flag_enum .....	165
Table 3-227. Enum i2c_interrupt_enum .....	166
Table 3-228. Function i2c_deinit .....	166
Table 3-229. Function i2c_clock_config.....	166
Table 3-230. Function i2c_mode_addr_config .....	167
Table 3-231. Function i2c_smbus_type_config .....	168
Table 3-232. Function i2c_ack_config .....	169
Table 3-233. Function i2c_ackpos_config .....	169
Table 3-234. Function i2c_master_addressing .....	170
Table 3-235. Function i2c_dualaddr_enable .....	171
Table 3-236. Function i2c_dualaddr_disable .....	171
Table 3-237. Function i2c_enable .....	172
Table 3-238. Function i2c_disable .....	172
Table 3-239. Function i2c_start_on_bus .....	173
Table 3-240. Function i2c_stop_on_bus .....	173
Table 3-241. Function i2c_data_transmit .....	174
Table 3-242. Function i2c_data_receive .....	174
Table 3-243. Function i2c_dma_config.....	175
Table 3-244. Function i2c_dma_last_transfer_config .....	175
Table 3-245. Function i2c_stretch_scl_low_config .....	176
Table 3-246. Function i2c_slave_response_to_gcall_config.....	177
Table 3-247. Function i2c_software_reset_config .....	177
Table 3-248. Function i2c_pec_config .....	178
Table 3-249. Function i2c_pec_transfer_config.....	178
Table 3-250. Function i2c_pec_value_get.....	179
Table 3-251. Function i2c_smbus_alert_config .....	180
Table 3-252. Function i2c_smbus_arp_config .....	180
Table 3-253. Function i2c_flag_get .....	181

Table 3-254. Function i2c_flag_clear .....	182
Table 3-255. Function i2c_interrupt_enable .....	183
Table 3-256. Function i2c_interrupt_disable .....	183
Table 3-257. Function i2c_interrupt_flag_get.....	184
Table 3-258. Function i2c_interrupt_flag_clear.....	185
Table 3-259. NVIC Registers .....	186
Table 3-260. SysTick Registers .....	187
Table 3-261. MISC firmware function .....	187
Table 3-262. Enum IRQn_Type .....	188
Table 3-263. Function nvic_priority_group_set .....	189
Table 3-264. Function nvic_irq_enable.....	190
Table 3-265. Function nvic_irq_disable.....	190
Table 3-266. Function nvic_vector_table_set.....	191
Table 3-267. Function system_lowpower_set .....	191
Table 3-268. Function system_lowpower_reset.....	192
Table 3-269. Function systick_clksource_set .....	192
Table 3-270. PMU Registers.....	193
Table 3-271. PMU firmware function .....	193
Table 3-272. Function pmu_deinit .....	194
Table 3-273. Function pmu_lvd_select.....	194
Table 3-274. Function pmu_lvd_disable.....	195
Table 3-275. Function pmu_to_sleepmode.....	195
Table 3-276. Function pmu_to_deepsleepmode .....	196
Table 3-277. Function pmu_to_standbymode .....	197
Table 3-278. Function pmu_wakeup_pin_enable .....	197
Table 3-279. Function pmu_wakeup_pin_disable .....	198
Table 3-280. Function pmu_backup_write_enable .....	198
Table 3-281. Function pmu_backup_write_disable .....	199
Table 3-282. Function pmu_flag_clear.....	199
Table 3-283. Function pmu_flag_get.....	200
Table 3-284. RCU Registers .....	201
Table 3-285. RCU firmware function .....	201
Table 3-286. Enum reg_idx .....	202
Table 3-287. Enum rcu_periph_enum .....	203
Table 3-288. Enum rcu_periph_sleep_enum .....	204
Table 3-289. Enum rcu_periph_reset_enum .....	204
Table 3-290. Enum rcu_flag_enum.....	205
Table 3-291. Enum rcu_int_flag_enum .....	205
Table 3-292. Enum rcu_int_flag_clear_enum .....	206
Table 3-293. Enum rcu_int_enum.....	206
Table 3-294. Enum rcu_adc_clock_enum .....	206
Table 3-295. Enum rcu_osci_type_enum .....	207
Table 3-296. Enum rcu_clock_freq_enum.....	207

Table 3-297. Function rcu_deinit .....	207
Table 3-298. Function rcu_periph_clock_enable .....	208
Table 3-299. Function rcu_periph_clock_disable .....	209
Table 3-300. Function rcu_periph_clock_sleep_enable .....	210
Table 3-301. Function rcu_periph_clock_sleep_disable .....	210
Table 3-302. Function rcu_periph_reset_enable .....	211
Table 3-303. Function rcu_periph_reset_disable .....	212
Table 3-304. Function rcu_bkp_reset_enable .....	212
Table 3-305. Function rcu_bkp_reset_disable .....	213
Table 3-306. Function rcu_system_clock_source_config .....	213
Table 3-307. Function rcu_system_clock_source_get .....	214
Table 3-308. Function rcu_ahb_clock_config .....	214
Table 3-309. Function rcu_apb1_clock_config .....	215
Table 3-310. Function rcu_apb2_clock_config .....	216
Table 3-311. Function rcu_adc_clock_config .....	216
Table 3-312. Function rcu_usbd_clock_config .....	217
Table 3-313. Function rcu_ckout_config .....	217
Table 3-314. Function rcu_ckout0_config .....	218
Table 3-315. Function rcu_ckout1_config .....	219
Table 3-316. Function rcu_pll_config .....	220
Table 3-317. Function rcu_usart_clock_config .....	221
Table 3-318. Function rcu_cec_clock_config .....	222
Table 3-319. Function rcu_rtc_clock_config .....	222
Table 3-320. Function rcu_slcd_clock_config .....	223
Table 3-321. Function rcu_hxtal_prediv_config .....	223
Table 3-322. Function rcu_lxtal_drive_capability_config .....	224
Table 3-323. Function rcu_flag_get .....	225
Table 3-324. Function rcu_all_reset_flag_clear .....	226
Table 3-325. Function rcu_interrupt_flag_get .....	226
Table 3-326. Function rcu_interrupt_flag_clear .....	227
Table 3-327. Function rcu_interrupt_enable .....	228
Table 3-328. Function rcu_interrupt_disable .....	228
Table 3-329. Function rcu_osci_stab_wait .....	229
Table 3-330. Function rcu_osci_on .....	230
Table 3-331. Function rcu_osci_off .....	230
Table 3-332. Function rcu_osci_bypass_mode_enable .....	231
Table 3-333. Function rcu_osci_bypass_mode_disable .....	231
Table 3-334. Function rcu_hxtal_clock_monitor_enable .....	232
Table 3-335. Function rcu_hxtal_clock_monitor_disable .....	233
Table 3-336. Function rcu_irc8m_adjust_value_set .....	233
Table 3-337. Function rcu_irc14m_adjust_value_set .....	234
Table 3-338. Function rcu_irc28m_adjust_value_set .....	234
Table 3-339. Function rcu_voltage_key_unlock .....	235

Table 3-340. Function rcu_deepsleep_voltage_set .....	235
Table 3-341. Function rcu_power_down_voltage_set .....	236
Table 3-342. Function rcu_clock_freq_get.....	236
Table 3-343. RTC Registers .....	237
Table 3-344. RTC firmware function.....	238
Table 3-345. Structure rtc_parameter_struct.....	239
Table 3-346. Structure rtc_alarm_struct.....	239
Table 3-347. Structure rtc_timestamp_struct .....	239
Table 3-348. Structure rtc_tamper_struct .....	240
Table 3-349. Function rtc_deinit .....	240
Table 3-350. Function rtc_init.....	241
Table 3-351. Function rtc_init_mode_enter .....	241
Table 3-352. Function rtc_init_mode_exit.....	242
Table 3-353. Function rtc_register_sync_wait .....	242
Table 3-354. Function rtc_current_time_get.....	243
Table 3-355. Function rtc_subsecond_get.....	243
Table 3-356. Function rtc_alarm_config.....	244
Table 3-357. Function rtc_alarm_subsecond_config.....	244
Table 3-358. Function rtc_alarm_get.....	245
Table 3-359. Function rtc_alarm_subsecond_get .....	246
Table 3-360. Function rtc_alarm_enable .....	246
Table 3-361. Function rtc_alarm_disable .....	247
Table 3-362. Function rtc_timestamp_enable .....	247
Table 3-363. Function rtc_timestamp_disable .....	248
Table 3-364. Function rtc_timestamp_get.....	248
Table 3-365. Function rtc_timestamp_subsecond_get.....	249
Table 3-366. Function rtc_tamper_enable.....	250
Table 3-367. Function rtc_tamper_disable.....	250
Table 3-368. Function rtc_interrupt_enable .....	251
Table 3-369. Function rtc_interrupt_disable.....	251
Table 3-370. Function rtc_flag_get.....	252
Table 3-371. Function rtc_flag_clear.....	253
Table 3-372. Function rtc_alter_output_config .....	253
Table 3-373. Function rtc_calibration_config.....	254
Table 3-374. Function rtc_hour_adjust.....	255
Table 3-375. Function rtc_second_adjust.....	255
Table 3-376. Function rtc_bypass_shadow_enable .....	256
Table 3-377. Function rtc_bypass_shadow_disable .....	256
Table 3-378. Function rtc_refclock_detection_enable .....	257
Table 3-379. Function rtc_refclock_detection_disable .....	257
Table 3-380. SLCD Registers .....	258
Table 3-381. SLCD firmware function .....	258
Table 3-382. slcd_data_register_enum.....	259

Table 3-383. Function <code>slcd_deinit</code> .....	259
Table 3-384. Function <code>slcd_enable</code> .....	260
Table 3-385. Function <code>slcd_disable</code> .....	260
Table 3-386. Function <code>slcd_bias_voltage_select</code> .....	261
Table 3-387. Function <code>slcd_duty_select</code> .....	261
Table 3-388. Function <code>slcd_clock_config</code> .....	262
Table 3-389. Function <code>slcd_blink_mode_config</code> .....	263
Table 3-390. Function <code>slcd_contrast_ratio_config</code> .....	265
Table 3-391. Function <code>slcd_dead_time_config</code> .....	265
Table 3-392. Function <code>slcd_pulse_on_duration_config</code> .....	266
Table 3-393. Function <code>slcd_com_seg_remap</code> .....	267
Table 3-394. Function <code>slcd_voltage_source_select</code> .....	268
Table 3-395. Function <code>slcd_high_drive_config</code> .....	268
Table 3-396. Function <code>slcd_high_drive_config</code> .....	269
Table 3-397. Function <code>slcd_data_update_request</code> .....	269
Table 3-398. Function <code>slcd_interrupt_config</code> .....	270
Table 3-399. Function <code>slcd_flag_get</code> .....	270
Table 3-400. Function <code>slcd_flag_clear</code> .....	271
Table 3-401. Function <code>slcd_interrupt_flag_get</code> .....	272
Table 3-402. Function <code>slcd_interrupt_flag_clear</code> .....	272
Table 3-403. SPI/I2S registers.....	273
Table 3-404. SPI/I2S firmware function.....	273
Table 3-405. Structure <code>spi_parameter_struct</code> .....	274
Table 3-406. Function <code>spi_i2s_deinit</code> .....	275
Table 3-407. Function <code>spi_struct_para_init</code> .....	275
Table 3-408. Function <code>spi_init</code> .....	276
Table 3-409. Function <code>spi_enable</code> .....	277
Table 3-410. Function <code>spi_disable</code> .....	277
Table 3-411. Function <code>i2s_init</code> .....	278
Table 3-412. Function <code>i2s_psc_config</code> .....	279
Table 3-413. Function <code>i2s_enable</code> .....	280
Table 3-414. Function <code>i2s_disable</code> .....	280
Table 3-415. Function <code>spi_nss_output_enable</code> .....	281
Table 3-416. Function <code>spi_nss_output_disable</code> .....	281
Table 3-417. Function <code>spi_nss_internal_high</code> .....	282
Table 3-418. Function <code>spi_nss_internal_low</code> .....	283
Table 3-419. Function <code>spi_dma_enable</code> .....	283
Table 3-420. Function <code>spi_dma_disable</code> .....	284
Table 3-421. Function <code>spi_i2s_data_frame_format_config</code> .....	284
Table 3-422. Function <code>spi_bidirectional_transfer_config</code> .....	285
Table 3-423. Function <code>spi_i2s_data_transmit</code> .....	286
Table 3-424. Function <code>spi_i2s_data_receive</code> .....	286
Table 3-425. Function <code>i2s_format_error_clear</code> .....	287



Table 3-426. Function spi_crc_polynomial_set .....	287
Table 3-427. Function spi_crc_polynomial_get .....	288
Table 3-428. Function spi_crc_on .....	288
Table 3-429. Function spi_crc_off .....	289
Table 3-430. Function spi_crc_next .....	289
Table 3-431. Function spi_crc_get .....	290
Table 3-432. Function spi_crc_error_clear .....	290
Table 3-433. Function spi_i2s_flag_get .....	291
Table 3-434. Function spi_i2s_interrupt_enable .....	292
Table 3-435. Function spi_i2s_interrupt_disable .....	293
Table 3-436. Function spi_i2s_interrupt_flag_get .....	293
Table 3-437. SYSCFG Registers.....	294
Table 3-438. SYSCFG firmware function .....	295
Table 3-439. Function syscfg_deinit .....	295
Table 3-440. Function syscfg_dma_remap_enable .....	296
Table 3-441. Function syscfg_dma_remap_disable .....	296
Table 3-442. Function syscfg_high_current_enable .....	297
Table 3-443. Function syscfg_high_current_disable .....	298
Table 3-444. Function syscfg_exti_line_config.....	298
Table 3-445. Function syscfg_lock_config .....	299
Table 3-446. Function syscfg_flag_get.....	299
Table 3-447. Function syscfg_flag_clear.....	300
Table 3-448. Function syscfg_compensation_config .....	300
Table 3-449. Function syscfg_cps_rdy_flag_get .....	301
Table 3-450. TIMERx Registers .....	302
Table 3-451. TIMERx firmware function.....	303
Table 3-452. Structure timer_parameter_struct .....	306
Table 3-453. Structure timer_break_parameter_struct.....	306
Table 3-454. Structure timer_oc_parameter_struct.....	306
Table 3-455. Structure timer_ic_parameter_struct.....	307
Table 3-456. Function timer_deinit.....	307
Table 3-457. Function timer_struct_para_init.....	308
Table 3-458. Function timer_init .....	308
Table 3-459. Function timer_enable .....	309
Table 3-460. Function timer_disable .....	309
Table 3-461. Function timer_auto_reload_shadow_enable .....	310
Table 3-462. Function timer_auto_reload_shadow_disable .....	311
Table 3-463. Function timer_update_event_enable .....	311
Table 3-464. Function timer_update_event_disable .....	312
Table 3-465. Function timer_counter_alignment .....	312
Table 3-466. Function timer_counter_up_direction .....	313
Table 3-467. Function timer_counter_down_direction .....	314
Table 3-468. Function timer_prescaler_config.....	314



Table 3-469. Function timer_repetition_value_config .....	315
Table 3-470. Function timer_autoreload_value_config .....	315
Table 3-471. Function timer_counter_value_config.....	316
Table 3-472. Function timer_counter_read .....	317
Table 3-473. Function timer_prescaler_read .....	317
Table 3-474. Function timer_single_pulse_mode_config .....	318
Table 3-475. Function timer_update_source_config.....	318
Table 3-476. Function timer_ocpre_clear_source_config .....	319
Table 3-477. Function timer_interrupt_enable .....	320
Table 3-478. Function timer_interrupt_disable .....	321
Table 3-479. Function timer_interrupt_flag_get.....	321
Table 3-480. Function timer_interrupt_flag_clear.....	322
Table 3-481. Function timer_flag_get .....	323
Table 3-482. Function timer_flag_clear .....	324
Table 3-483. Function timer_dma_enable .....	325
Table 3-484. Function timer_dma_disable .....	325
Table 3-485. Function timer_channel_dma_request_source_select.....	326
Table 3-486. Function timer_dma_transfer_config.....	327
Table 3-487. Function timer_event_software_generate.....	328
Table 3-488. Function timer_break_struct_para_init .....	329
Table 3-489. Function timer_break_config .....	330
Table 3-490. Function timer_break_enable .....	331
Table 3-491. Function timer_break_disable .....	331
Table 3-492. Function timer_automatic_output_enable .....	332
Table 3-493. Function timer_automatic_output_disable .....	332
Table 3-494. Function timer_primary_output_config.....	333
Table 3-495. Function timer_channel_control_shadow_config .....	334
Table 3-496. Function timer_channel_control_shadow_update_config.....	334
Table 3-497. Function timer_channel_output_struct_para_init .....	335
Table 3-498. Function timer_channel_output_config .....	336
Table 3-499. Function timer_channel_output_mode_config .....	337
Table 3-500. Function timer_channel_output_pulse_value_config.....	338
Table 3-501. Function timer_channel_output_shadow_config .....	338
Table 3-502. Function timer_channel_output_fast_config.....	339
Table 3-503. Function timer_channel_output_clear_config .....	340
Table 3-504. Function timer_channel_output_polarity_config.....	341
Table 3-505. Function timer_channel_complementary_output_polarity_config .....	342
Table 3-506. Function timer_channel_output_state_config .....	343
Table 3-507. Function timer_channel_complementary_output_state_config .....	343
Table 3-508. Function timer_channel_input_struct_para_init.....	344
Table 3-509. Function timer_input_capture_config.....	345
Table 3-510. Function timer_channel_input_capture_prescaler_config .....	346
Table 3-511. Function timer_channel_capture_value_register_read .....	346

Table 3-512. Function timer_input_pwm_capture_config .....	347
Table 3-513. Function timer_hall_mode_config .....	348
Table 3-514. Function timer_input_trigger_source_select .....	349
Table 3-515. Function timer_master_output_trigger_source_select .....	350
Table 3-516. Function timer_slave_mode_select .....	351
Table 3-517. Function timer_master_slave_mode_config .....	352
Table 3-518. Function timer_external_trigger_config .....	352
Table 3-519. Function timer_quadrature_decoder_mode_config .....	353
Table 3-520. Function timer_internal_clock_config .....	354
Table 3-521. Function timer_internal_trigger_as_external_clock_config .....	355
Table 3-522. Function timer_external_trigger_as_external_clock_config .....	356
Table 3-523. Function timer_external_clock_mode0_config .....	357
Table 3-524. Function timer_external_clock_mode1_config .....	357
Table 3-525. Function timer_external_clock_mode1_disable .....	358
Table 3-526. Function timer_channel_remap_config .....	359
Table 3-527. TSI Registers .....	360
Table 3-528. TSI firmware function .....	360
Table 3-529. Function tsi_deinit .....	361
Table 3-530. Function tsi_init .....	362
Table 3-531. Function tsi_enable .....	363
Table 3-532. Function tsi_disable .....	364
Table 3-533. Function tsi_sample_pin_enable .....	364
Table 3-534. Function tsi_sample_pin_disable .....	365
Table 3-535. Function tsi_channel_pin_enable .....	365
Table 3-536. Function tsi_channel_pin_disable .....	366
Table 3-537. Function tsi_software_mode_config .....	366
Table 3-538. Function tsi_software_start .....	367
Table 3-539. Function tsi_software_stop .....	367
Table 3-540. Function tsi_hardware_mode_config .....	368
Table 3-541. Function tsi_pin_mode_config .....	368
Table 3-542. Function tsi_extend_charge_config .....	369
Table 3-543. Function tsi_plus_config .....	369
Table 3-544. Function tsi_max_number_config .....	370
Table 3-545. Function tsi_hysteresis_on .....	371
Table 3-546. Function tsi_hysteresis_off .....	372
Table 3-547. Function tsi_analog_on .....	372
Table 3-548. Function tsi_analog_off .....	373
Table 3-549. Function tsi_flag_get .....	373
Table 3-550. Function tsi_flag_clear .....	374
Table 3-551. Function tsi_interrupt_enable .....	374
Table 3-552. Function tsi_interrupt_disable .....	375
Table 3-553. Function tsi_interrupt_flag_get .....	375
Table 3-554. Function tsi_interrupt_flag_clear .....	376

Table 3-555. Function tsi_group_enable .....	376
Table 3-556. Function tsi_group_disable .....	377
Table 3-557. Function tsi_group_status_get .....	377
Table 3-558. Function tsi_group0_cycle_get.....	378
Table 3-559. Function tsi_group1_cycle_get.....	379
Table 3-560. Function tsi_group2_cycle_get.....	379
Table 3-561. Function tsi_group3_cycle_get.....	380
Table 3-562. Function tsi_group4_cycle_get.....	380
Table 3-563. Function tsi_group5_cycle_get.....	381
Table 3-564. USART Registers .....	381
Table 3-565. USART firmware function.....	382
Table 3-566. Enum usart_flag_enum.....	384
Table 3-567. Enum usart_interrupt_flag_enum .....	384
Table 3-568. Enum usart_interrupt_enum .....	385
Table 3-569. Enum usart_invert_enum .....	385
Table 3-570. Function usart_deinit.....	386
Table 3-571. Function usart_baudrate_set .....	386
Table 3-572. Function usart_parity_config .....	387
Table 3-573. Function usart_word_length_set .....	387
Table 3-574. Function usart_stop_bit_set.....	388
Table 3-575. Function usart_enable .....	388
Table 3-576. Function usart_disable .....	389
Table 3-577. Function usart_transmit_config.....	389
Table 3-578. Function usart_receive_config .....	390
Table 3-579. Function usart_data_first_config .....	391
Table 3-580. Function usart_invert_config .....	391
Table 3-581. Function usart_overrun_enable.....	392
Table 3-582. Function usart_overrun_disable.....	393
Table 3-583. Function usart_oversample_config .....	393
Table 3-584. Function usart_sample_bit_config.....	394
Table 3-585. Function usart_receiver_timeout_enable.....	394
Table 3-586. Function usart_receiver_timeout_disable.....	395
Table 3-587. Function usart_receiver_timeout_threshold_config .....	395
Table 3-588. Function usart_data_transmit .....	396
Table 3-589. Function usart_data_receive .....	396
Table 3-590. Function usart_address_config .....	397
Table 3-591. Function usart_address_detection_mode_config .....	398
Table 3-592. Function usart_mute_mode_enable.....	398
Table 3-593. Function usart_mute_mode_disable.....	399
Table 3-594. Function usart_mute_mode_wakeup_config .....	399
Table 3-595. Function usart_lin_mode_enable .....	400
Table 3-596. Function usart_lin_mode_disable .....	400
Table 3-597. Function usart_lin_break_dection_length_config .....	401

Table 3-598. Function <code>usart_halfduplex_enable</code> .....	401
Table 3-599. Function <code>usart_halfduplex_disable</code> .....	402
Table 3-600. Function <code>usart_clock_enable</code> .....	402
Table 3-601. Function <code>usart_clock_disable</code> .....	403
Table 3-602. Function <code>usart_synchronous_clock_config</code> .....	403
Table 3-603. Function <code>usart_guard_time_config</code> .....	404
Table 3-604. Function <code>usart_smartcard_mode_enable</code> .....	405
Table 3-605. Function <code>usart_smartcard_mode_disable</code> .....	405
Table 3-606. Function <code>usart_smartcard_mode_nack_enable</code> .....	406
Table 3-607. Function <code>usart_smartcard_mode_nack_disable</code> .....	406
Table 3-608. Function <code>usart_smartcard_autoretry_config</code> .....	407
Table 3-609. Function <code>usart_block_length_config</code> .....	407
Table 3-610. Function <code>usart_irda_mode_enable</code> .....	408
Table 3-611. Function <code>usart_irda_mode_disable</code> .....	408
Table 3-612. Function <code>usart_prescaler_config</code> .....	409
Table 3-613. Function <code>usart_irda_lowpower_config</code> .....	409
Table 3-614. Function <code>usart_hardware_flow_rts_config</code> .....	410
Table 3-615. Function <code>usart_hardware_flow_cts_config</code> .....	411
Table 3-616. Function <code>usart_rs485_driver_enable</code> .....	411
Table 3-617. Function <code>usart_rs485_driver_disable</code> .....	412
Table 3-618. Function <code>usart_driver_asserttime_config</code> .....	412
Table 3-619. Function <code>usart_driver_deasserttime_config</code> .....	413
Table 3-620. Function <code>usart_depolarity_config</code> .....	413
Table 3-621. Function <code>usart_dma_receive_config</code> .....	414
Table 3-622. Function <code>usart_dma_transmit_config</code> .....	414
Table 3-623. Function <code>usart_reception_error_dma_disable</code> .....	415
Table 3-624. Function <code>usart_reception_error_dma_enable</code> .....	416
Table 3-625. Function <code>usart_wakeup_enable</code> .....	416
Table 3-626. Function <code>usart_wakeup_disable</code> .....	417
Table 3-627. Function <code>usart_wakeup_mode_config</code> .....	417
Table 3-628. Function <code>usart_command_enable</code> .....	418
Table 3-629. Function <code>usart_flag_get</code> .....	418
Table 3-630. Function <code>usart_flag_clear</code> .....	419
Table 3-631. Function <code>usart_interrupt_enable</code> .....	420
Table 3-632. Function <code>usart_interrupt_disable</code> .....	421
Table 3-633. Function <code>usart_interrupt_flag_get</code> .....	422
Table 3-634. Function <code>usart_interrupt_flag_clear</code> .....	423
Table 3-635. WWDGT Registers .....	425
Table 3-636. WWDGT firmware function .....	425
Table 3-637. Function <code>wwdgt_deinit</code> .....	425
Table 3-638. Function <code>wwdgt_enable</code> .....	426
Table 3-639. Function <code>wwdgt_counter_update</code> .....	426
Table 3-640. Function <code>wwdgt_config</code> .....	427

Table 3-641. Function <code>wwdgt_interrupt_enable</code> .....	427
Table 3-642. Function <code>wwdgt_flag_get</code> .....	428
Table 3-643. Function <code>wwdgt_flag_clear</code> .....	428
Table 4-1. Revision history .....	430

## 1. Introduction

This manual introduces firmware library of GD32F1x0 devices which are 32-bit microcontrollers based on the ARM processor.

The firmware library is a firmware function package, including program, data structure and macro definitions, all the performance features of peripherals of GD32F1x0 devices are involved in the package. The peripheral driving code and firmware examples on evaluation board are also included in firmware library. Users need not learn each peripherals in details and it's easy to apply a peripheral by using the firmware library. Using firmware library can greatly reduce programming time, thereby reducing development costs.

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

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

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

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

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

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

#### 1.1.1. Peripherals

**Table 1-1. Peripherals**

Peripherals	Descriptions
ADC	Analog-to-digital converter
CAN	Controller area network
CEC	HDMI-CEC controller
CMP	Comparator
CRC	CRC calculation unit
DAC	Digital-to-analog converter

Peripherals	Descriptions
DBG	Debug
DMA	Direct memory access controller
EXTI	Interrupt/event controller
FMC	Flash memory controller
FWDGT	Free watchdog timer
GPIO	General-purpose I/Os
I2C	Inter-integrated circuit interface
IVREF	Programmable Current and Voltage Reference
MISC	Nested Vectored Interrupt Controller
OPA	Operational amplifiers
PMU	Power management unit
RCU	Reset and clock unit
RTC	Real-time Clock
SLCD	Segment LCD controller
SPI/I2S	Serial peripheral interface/Inter-IC sound
SYSCFG	System configuration
TIMER	TIMER
TSI	Touch sensing interface
USART	Universal synchronous/asynchronous receiver /transmitter
WWDGT	Window watchdog timer
USB	Universal serial bus full-speed device interface

### 1.1.2. Naming rules

The firmware library naming rules are shown as below:

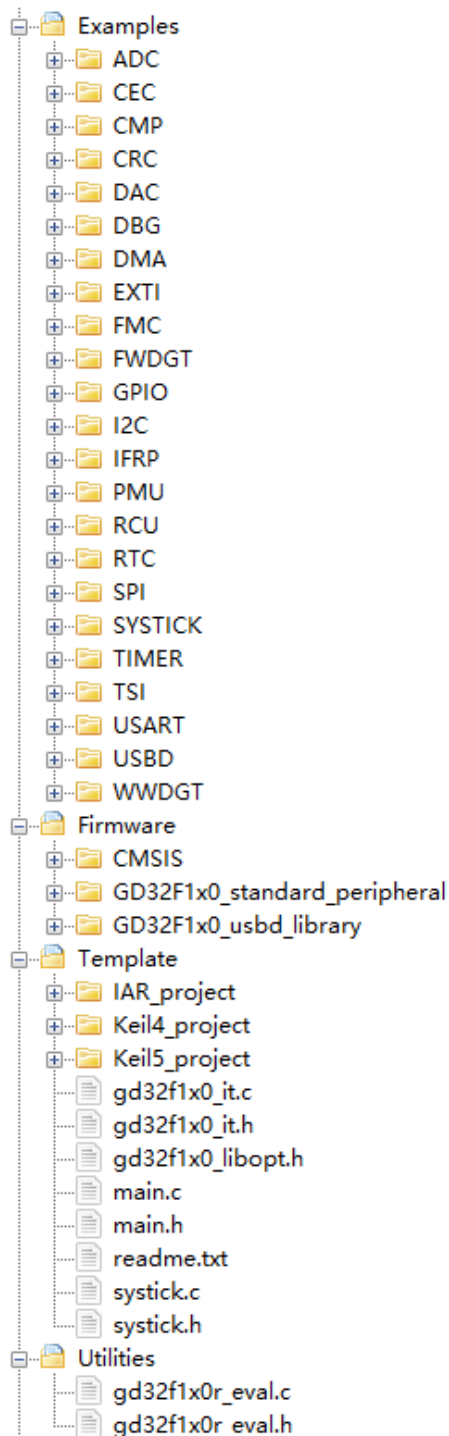
- The peripherals are shortened in XXX format, such as: ADC. More shorten information of peripherals refer to [Peripherals](#);
- The name of sourcefile and header file are started with “GD32f1x0\_”, such as: GD32f1x0\_adc.h;
- The constants used only in one file should be defined in the used file; the constants used in many files should be defined in corresponding header file. All the constants are written in uppcase of English letters;
- Registers are handled as constants. The naming of them are written in uppcase of English letters. In most cases, register names are shortened accord with the user manual;
- Variables are written in lowercase, when concluded by several words, underlines should be adapted among words;
- The naming of peripheral functions are started with the peripheral abbreviation added with an underline, when the function name is concluded by several words, underlines should be adapted among words, and all the peripheral functions are written in lowercase.

## 2. Firmware Library Overview

### 2.1. File Structure of Firmware Library

GD32F1x0\_Firmware\_Library, the file structure is shown as below:

**Figure 2-1. File structure of firmware library of GD32F1x0**





### 2.1.1. Examples Folder

Examples folder, each of GD32 peripheral has a subfolder. Each subfolder contains one or more examples of the peripheral, to show how to use the peripheral correctly. Each of the example subfolder includes the files shown as below:

- readme.txt: the description and using guide of the example;
- GD32f1x0\_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);
- GD32f1x0\_it.c: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- GD32f1x0.it.h: the header file include all the prototypes of the interrupt service routines;
- systick.c: the source file include the precise time delay functions by using systick;
- systick.h: the header file include the prototype of the precise time delay functions by using systick;
- main.c: example code. Note: all the examples are not influenced by software IDEs.

### 2.1.2. Firmware Folder

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

- CMSIS subfolder includes the Cortex M3 kernel support files, the startup file based on the Cortex M3 kernel processor, the global header file of GD32F1x0 and system configuration file;
- GD32F1x0\_standard\_peripheral subfolder:
  - Include subfolder includes all the header files of firmware library, users need not modify this folder;
  - Source subfolder includes all the source files of firmware library, users need not modify this folder;

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

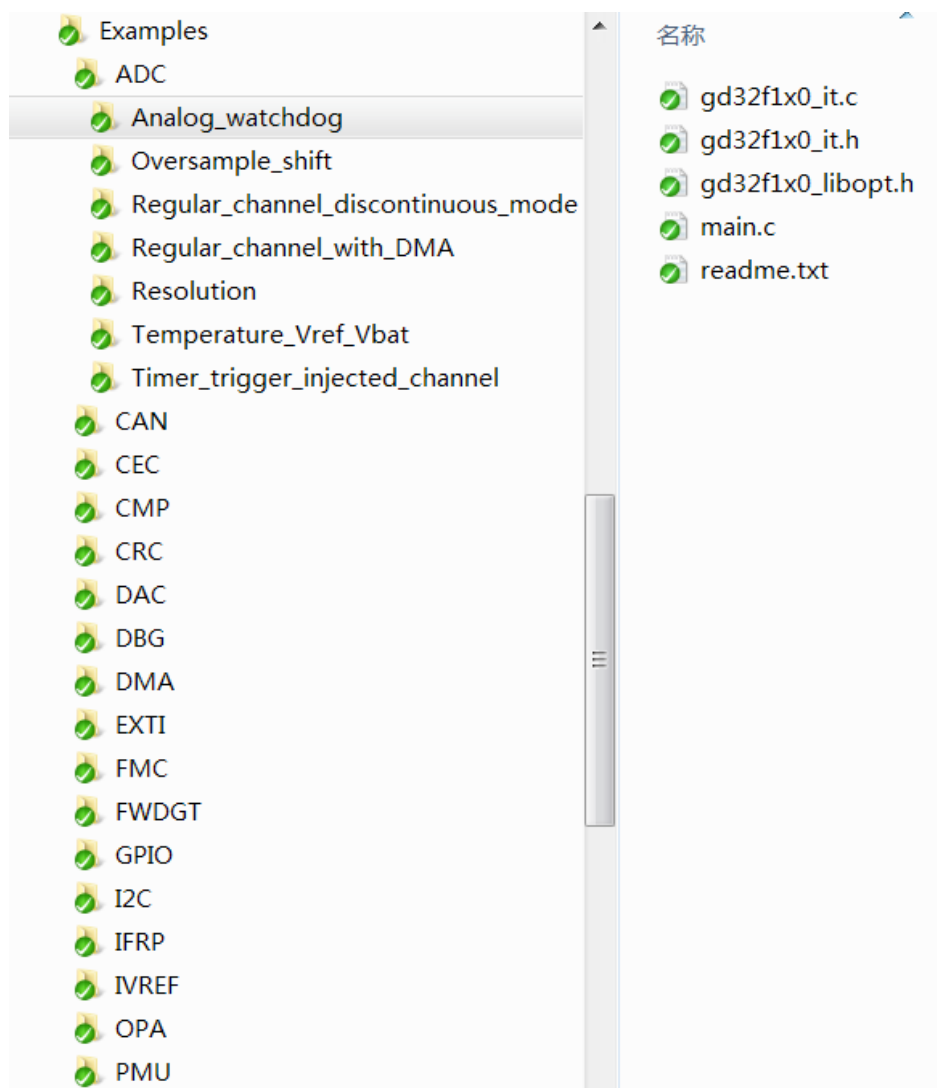
### 2.1.3. Template Folder

Template folder includes a simple demo of how to use LED, how to print by USART and use key to control, (IAR\_project is run in IAR, Keil4\_project is run in Keil4 and Keil5\_project is run in Keil5). User can use the project template to compile the formware examples, the steps are shown as below:

#### Select files

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

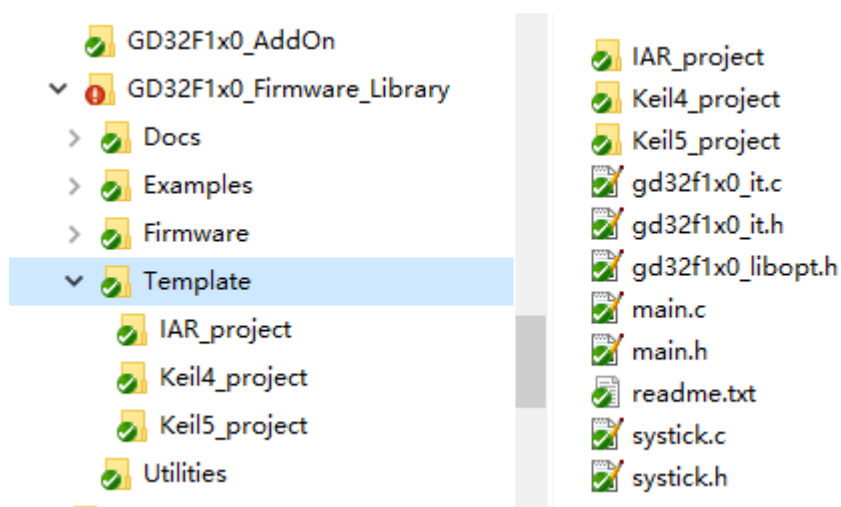
**Figure 2-2. Select peripheral example files**



## Copy files

Open "Template" folder, keep the folders of " IAR\_project", " Keil4\_project" and " Keil5\_project", and delete the other files, then copy all the files in "SPI\_master\_transmit\_slave\_receive\_interrupt" folder to the "Template" subfolder, shown as below:

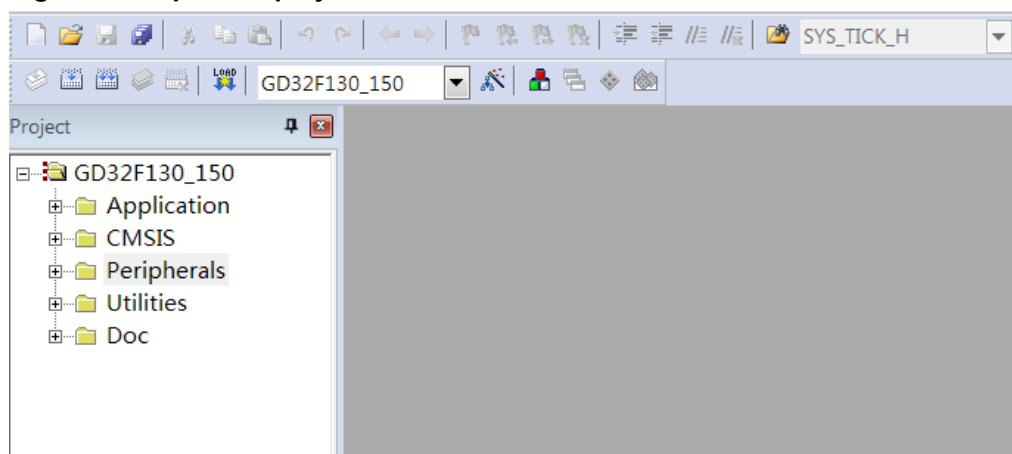
**Figure 2-3. Copy the peripheral example files**



## Open a project

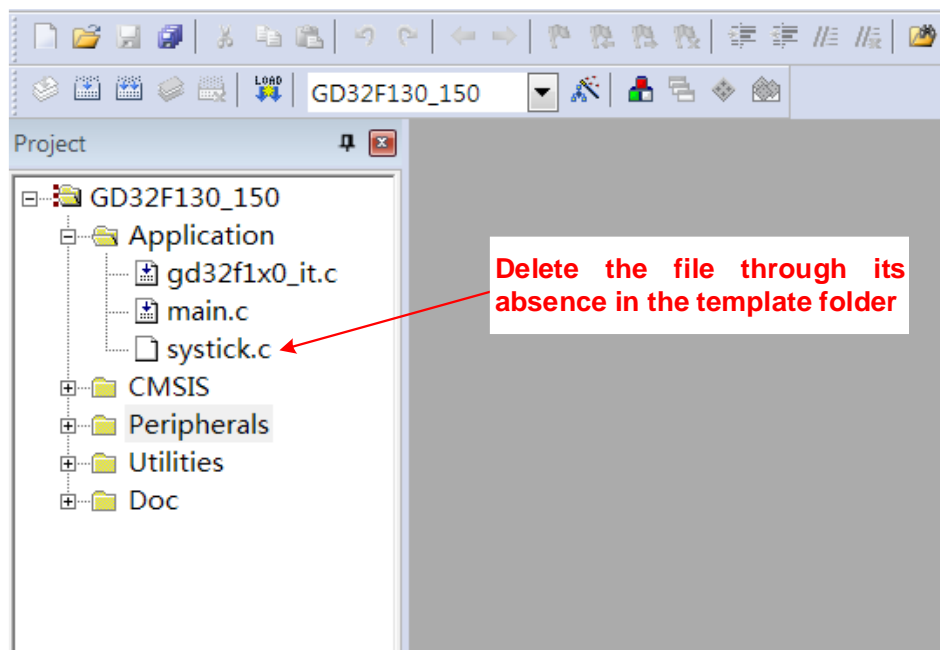
GD provides project in IAR, Keil4 and Keil5, users can open project in different IDEs according to their need, such as "Keil4\_project", open \Template\Keil4\_project\Project.uvproj, shown as below:

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



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

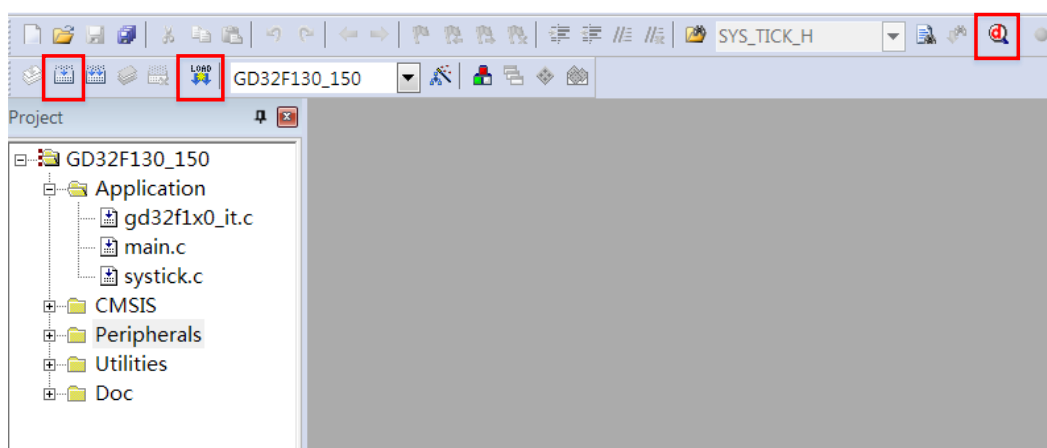
**Figure 2-5. Configure project files**



## Compile-Debug-Download

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

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



### 2.1.4. Utilities Folder

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

- Binary subfolders;
- gd32f1x0r\_eval.h is related header files of the evaluation board about running the firmware examples;

- `gd32f1x0r_eval.c` is related source files of the evaluation board about running the firmware examples.

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

## 2.2. File descriptions of Firmware Library

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

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

Files	Descriptions
gd32f1x0_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.
gd32f1x0_it.h	Header file, including all the prototypes of interrupt service routines.
gd32f1x0_it.c	Source files about interrupt service routines of peripherals. User can written his own interrupt functions in this file. For the different interrupt service requests to the same interrupt vector, users can confirm the interrupt source by functions of judging interrupt flags of peripherals. The functions are included in the firmware library.
gd32f1x0_xxx.h	The header file of peripheral xxx, including functions about peripheral xxx, and the variables used for functions.
gd32f1x0_xxx.c	The C source file for driving peripheral xxx.
systick.h	The header file of systick.c, including prototypes of systick configuration function and delay function.
systick.c	The source file about systick configuration function and delay function.
readme.txt	Description document about how to configure and how to use the firmware example.

## 3. Firmware Library of Standard Peripherals

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

The description format of firmware functions are shown as below:

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

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

### 3.2. ADC

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

#### 3.2.1. Descriptions of Peripheral registers

ADC registers are listed in the table shown as below:

**Table 3-2. ADC Registers**

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

Registers	Descriptions
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

### 3.2.2. Descriptions of Peripheral functions

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

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

Function name	Function description
adc_deinit	reset ADC peripheral
adc_enable	enable ADC interface
adc_disable	disable ADC interface
adc_calibration_enable	ADC calibration and reset calibration
adc_dma_mode_enable	enable DMA request
adc_dma_mode_disable	disable DMA request
adc_tempsensor_vrefint_enable	enable the temperature sensor and Vrefint channel
adc_tempsensor_vrefint_disable	disable the temperature sensor and Vrefint channel
adc_vbat_enable	enable the Vbat channel
adc_vbat_disable	disable the Vbat channel
adc_discontinuous_mode_config	configure ADC discontinuous mode
adc_special_function_config	enable or disable ADC special function
adc_data_alignment_config	configure ADC data alignment
adc_channel_length_config	configure the length of regular channel group or inserted channel group
adc_regular_channel_config	configure ADC regular channel
adc_inserted_channel_config	configure ADC inserted channel
adc_inserted_channel_offset_config	configure ADC inserted channel offset
adc_external_trigger_config	enable ADC external trigger
adc_external_trigger_source_config	configure ADC external trigger source
adc_software_trigger_enable	enable ADC software trigger
adc_regular_data_read	read ADC regular group data register
adc_inserted_data_read	read ADC inserted group data register
adc_flag_get	get the ADC flag bits
adc_flag_clear	clear the ADC flag bits
adc_interrupt_flag_get	get the ADC interrupt bits
adc_interrupt_flag_clear	clear the ADC flag



Function name	Function description
adc_interrupt_enable	enable ADC interrupt
adc_interrupt_disable	disable ADC interrupt
adc_watchdog_single_channel_enable	configure ADC analog watchdog single channel
adc_watchdog_group_channel_enable	configure ADC analog watchdog group channel
adc_watchdog_disable	disable ADC analog watchdog
adc_watchdog_threshold_config	configure ADC analog watchdog threshold

## adc\_deinit

The description of adc\_deinit is shown as below:

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

Function name	adc_deinit
Function prototype	void adc_deinit(void);
Function descriptions	reset ADC peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset ADC */
```

```
adc_deinit();
```

## adc\_enable

The description of adc\_enable is shown as below:

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

Function name	adc_enable
Function prototype	void adc_enable(void);
Function descriptions	enable ADC interface
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC */
```

```
adc_enable();
```

### adc\_disable

The description of adc\_disable is shown as below:

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

Function name	adc_disable
Function prototype	void adc_disable(void);
Function descriptions	disable ADC interface
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
adc_disable();
```

### adc\_calibration\_enable

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

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

Function name	adc_calibration_enable
Function prototype	void adc_calibration_enable(void);
Function descriptions	ADC calibration and reset calibration
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* ADC calibration and reset calibration */
```

```
adc_calibration_enable();
```

### adc\_dma\_mode\_enable

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

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

<b>Function name</b>	adc_dma_mode_enable
<b>Function prototype</b>	void adc_dma_mode_enable(void);
<b>Function descriptions</b>	enable ADC DMA request
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC DMA request */
```

```
adc_dma_mode_enable();
```

### adc\_dma\_mode\_disable

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

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

<b>Function name</b>	adc_dma_mode_disable
<b>Function prototype</b>	void adc_dma_mode_disable(void);
<b>Function descriptions</b>	disable ADC DMA request
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
-	-

Example:

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

### adc\_tempsensor\_vrefint\_enable

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

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

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

Example:

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

### adc\_tempsensor\_vrefint\_disable

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

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

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

-	-
---	---

Example:

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

```
adc_tempsensor_vrefint_disable();
```

### adc\_vbat\_enable

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

**Table 3-12. Function adc\_vbat\_enable**

<b>Function name</b>	adc_vbat_enable
<b>Function prototype</b>	void adc_vbat_enable(void);
<b>Function descriptions</b>	enable the Vbat 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 Vbat channel */
```

```
adc_vbat_enable();
```

### adc\_vbat\_disable

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

**Table 3-13. Function adc\_vbat\_disable**

<b>Function name</b>	adc_vbat_disable
<b>Function prototype</b>	void adc_vbat_disable(void);
<b>Function descriptions</b>	disable the Vbat 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 Vbat channel */
```

```
adc_vbat_disable();
```

### adc\_discontinuous\_mode\_config

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

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

<b>Function name</b>	adc_discontinuous_mode_config
<b>Function prototype</b>	void adc_discontinuous_mode_config(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_channel_group</b>	select the channel group
ADC_REGULAR_CHANNEL	regular channel group
ADC_INSERTED_CHANNEL	inserted channel group
ADC_CHANNEL_DISCON_DISABLE	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 ADC regular channel group discontinuous mode */
```

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

### adc\_special\_function\_config

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

**Table 3-15. Function adc\_special\_function\_config**

<b>Function name</b>	adc_special_function_config
<b>Function prototype</b>	void adc_special_function_config(uint32_t 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>function</b>	the function to config
<i>ADC_SCAN_MODE</i>	scan mode select
<i>ADC_INSERTED_CHANNEL_AUTO</i>	inserted channel group convert automatically
<i>ADC_CONTINUOUS_MODE</i>	continuous mode select
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC scan mode */
```

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

### adc\_data\_alignment\_config

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

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

<b>Function name</b>	adc_data_alignment_config
<b>Function prototype</b>	void adc_data_alignment_config(uint32_t data_alignment);
<b>Function descriptions</b>	configure ADCx data alignment
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>data_alignment</b>	data alignment select
<i>ADC_DATAALIGN_RIGHT</i>	right alignment
<i>ADC_DATAALIGN_LEFT</i>	left alignment
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* configure ADC data alignment */
```

```
adc_data_alignment_config(ADC_DATAALIGN_RIGHT);
```

### adc\_channel\_length\_config

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

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

<b>Function name</b>	adc_channel_length_config
<b>Function prototype</b>	void adc_channel_length_config(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>channel_group</b>	select the channel group
ADC_REGULAR_CHANNEL	regular channel group
ADC_INSERTED_CHANNEL	inserted channel group
<b>Input parameter{in}</b>	
<b>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 ADC regular channel */
```

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

### adc\_regular\_channel\_config

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

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

<b>Function name</b>	adc_regular_channel_config
<b>Function prototype</b>	void adc_regular_channel_config(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>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..18)
<b>Input parameter{in}</b>	
<b>sample_time</b>	the sample time value
<i>ADC_SAMPLETIME_1POINT5</i>	1.5 cycles
<i>ADC_SAMPLETIME_7POINT5</i>	7.5 cycles
<i>ADC_SAMPLETIME_13POINT5</i>	13.5 cycles
<i>ADC_SAMPLETIME_28POINT5</i>	28.5 cycles
<i>ADC_SAMPLETIME_41POINT5</i>	41.5 cycles
<i>ADC_SAMPLETIME_55POINT5</i>	55.5 cycles
<i>ADC_SAMPLETIME_71POINT5</i>	71.5 cycles
<i>ADC_SAMPLETIME_239POINT5</i>	239.5 cycles
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC regular channel */
```

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

### **adc\_inserted\_channel\_config**

The description of `adc_inserted_channel_config` is shown as below:

**Table 3-19. Function `adc_inserted_channel_config`**

<b>Function name</b>	<code>adc_inserted_channel_config</code>
<b>Function prototype</b>	<code>void adc_inserted_channel_config(uint8_t rank, uint8_t channel, uint32_t sample_time);</code>
<b>Function descriptions</b>	configure ADC inserted channel
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rank</b>	the inserted group sequencer rank, this parameter must be between 0 to 3
<b>Input parameter{in}</b>	
<b>channel</b>	the selected ADC channel
<i>ADC_CHANNEL_x</i>	ADC Channelx (x=0..18)
<b>Input parameter{in}</b>	
<b>sample_time</b>	the sample time value
<i>ADC_SAMPLETIME_1POINT5</i>	1.5 cycles
<i>ADC_SAMPLETIME_7POINT5</i>	7.5 cycles
<i>ADC_SAMPLETIME_13POINT5</i>	13.5 cycles
<i>ADC_SAMPLETIME_28POINT5</i>	28.5 cycles
<i>ADC_SAMPLETIME_41POINT5</i>	41.5 cycles
<i>ADC_SAMPLETIME_55POINT5</i>	55.5 cycles
<i>ADC_SAMPLETIME_71POINT5</i>	71.5 cycles
<i>ADC_SAMPLETIME_239POINT5</i>	239.5 cycles
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

The description of `adc_inserted_channel_offset_config` is shown as below:

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

<b>Function name</b>	<code>adc_inserted_channel_offset_config</code>
<b>Function prototype</b>	<code>void adc_inserted_channel_offset_config(uint8_t inserted_channel, uint16_t offset);</code>
<b>Function descriptions</b>	configure ADC inserted channel offset
<b>Precondition</b>	-

The called functions	-
Input parameter{in}	
inserted_channel	insert channel select
ADC_INSERTED_CHANNEL_x	inserted channel, x=0,1,2,3
Input parameter{in}	
offset	the offset data, this parameter must be between 0 to 4095
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC inserted channel offset */
```

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

### adc\_external\_trigger\_config

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

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

Function name	adc_external_trigger_config
Function prototype	void adc_external_trigger_config(uint8_t channel_group, ControlStatus newvalue);
Function descriptions	configure ADC external trigger
Precondition	-
The called functions	-
Input parameter{in}	
channel_group	select the channel group
ADC_REGULAR_CHANNEL	regular channel group
ADC_INSERTED_CHANNEL	inserted channel group
Input parameter{in}	
newvalue	control value
ENABLE	enable function
DISABLE	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

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

The description of `adc_external_trigger_source_config` is shown as below:

**Table 3-22. Function `adc_external_trigger_source_config`**

Function name	<code>adc_external_trigger_source_config</code>
Function prototype	<code>void adc_external_trigger_source_config(uint8_t channel_group, uint32_t external_trigger_source);</code>
Function descriptions	configure ADC external trigger source
Precondition	-
The called functions	-
Input parameter{in}	
<b>channel_group</b>	select the channel group
<code>ADC_REGULAR_CHANNEL</code>	regular channel group
<code>ADC_INSERTED_CHANNEL</code>	inserted channel group
Input parameter{in}	
<b>external_trigger_source</b>	regular or inserted group trigger source
<code>ADC_EXTTRIG_REGULAR_T0_CH0</code>	TIMER0 CH0 event select for regular channel
<code>ADC_EXTTRIG_REGULAR_T0_CH1</code>	TIMER0 CH1 event select for regular channel
<code>ADC1_EXTTRIG_REGULAR_T0_CH2</code>	TIMER0 CH2 event select for regular channel
<code>ADC_EXTTRIG_REGULAR_T1_CH1</code>	TIMER1 CH1 event select for regular channel
<code>ADC_EXTTRIG_REGULAR_T2_TRGO</code>	TIMER2 TRGO event select for regular channel
<code>ADC_EXTTRIG_REGULAR_T14_CH0</code>	TIMER14 CH0 event select for regular channel
<code>ADC_EXTTRIG_REGULAR_EXTI_11</code>	external interrupt line 11 for regular channel
<code>ADC_EXTTRIG_REGULAR_NONE</code>	software trigger for regular channel
<code>ADC_EXTTRIG_INSERTED_T0_TRGO</code>	TIMER0 TRGO event select for inserted channel
<code>ADC_EXTTRIG_INSERTED_T0_CH3</code>	TIMER0 CH3 event select for inserted channel

<i>ADC_EXTTRIG_INSERTED_T1_TRGO</i>	TIMER1 TRGO event select for inserted channel
<i>ADC_EXTTRIG_INSERTED_T1_CH0</i>	TIMER1 CH0 event select for inserted channel
<i>ADC_EXTTRIG_INSERTED_T2_CH3</i>	TIMER2 CH3 event select for inserted channel
<i>ADC_EXTTRIG_INSERTED_T14_TRGO</i>	TIMER14 TRGO event select for inserted channel
<i>ADC_EXTTRIG_INSERTED_EXTI_15</i>	external interrupt line 15 for inserted channel
<i>ADC_EXTTRIG_INSERTED_NONE</i>	software trigger for inserted channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### adc\_software\_trigger\_enable

The description of `adc_software_trigger_enable` is shown as below:

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

<b>Function name</b>	<code>adc_software_trigger_enable</code>
<b>Function prototype</b>	<code>void adc_software_trigger_enable(uint8_t channel_group);</code>
<b>Function descriptions</b>	enable ADC software trigger
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channel_group</b>	select the channel group
<i>ADC_REGULAR_CHANNEL</i>	regular channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC regular channel group software trigger */
adc_software_trigger_enable( ADC_REGULAR_CHANNEL);
```

### adc\_regular\_data\_read

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

**Table 3-24. Function adc\_regular\_data\_read**

<b>Function name</b>	adc_regular_data_read
<b>Function prototype</b>	uint16_t adc_regular_data_read(void);
<b>Function descriptions</b>	read ADC regular group data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	ADC conversion value (0-0xFFFF)

Example:

```
/* read ADC regular group data register */
uint16_t adc_value = 0;
adc_value = adc_regular_data_read();
```

### adc\_inserted\_data\_read

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

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

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

Example:

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

uint16_t adc_value = 0;

adc_value = adc_inserted_data_read (ADC_INSERTED_CHANNEL_0);
```

### adc\_flag\_get

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

**Table 3-26. Function adc\_flag\_get**

Function name	adc_flag_get
Function prototype	FlagStatus adc_flag_get(uint32_t flag);
Function descriptions	get the ADC flag bits
Precondition	-
The called functions	-
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	
FlagStatus	SET or RESET

Example:

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

FlagStatus flag_value;

flag_value = adc_flag_get(ADC_FLAG_WDE);
```

### adc\_flag\_clear

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

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

Function name	adc_flag_clear
Function prototype	void adc_flag_clear(uint32_t flag);
Function descriptions	clear the ADC flag bits
Precondition	-
The called functions	-

Input parameter{in}	
<b>flag</b>	the adc flag bits
<i>ADC_FLAG_WDE</i>	analog watchdog event flag
<i>ADC_FLAG_EOC</i>	end of group conversion flag
<i>ADC_FLAG_EOIC</i>	end of inserted group conversion flag
<i>ADC_FLAG_STIC</i>	start flag of inserted channel group
<i>ADC_FLAG_STRC</i>	start flag of regular channel group
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
adc_flag_clear(ADC_FLAG_WDE);
```

### adc\_interrupt\_flag\_get

The description of `adc_interrupt_flag_get` is shown as below:

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

<b>Function name</b>	<code>adc_interrupt_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus adc_interrupt_flag_get(uint32_t flag);</code>
<b>Function descriptions</b>	get the ADC interrupt bits
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

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

```
FlagStatus flag_value;
```

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



## adc\_interrupt\_flag\_clear

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

**Table 3-29. Function adc\_interrupt\_flag\_clear**

<b>Function name</b>	adc_interrupt_flag_clear
<b>Function prototype</b>	void adc_interrupt_flag_clear(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>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>	
-	-

Example:

```
/* clear the ADC analog watchdog interrupt bits */
adc_interrupt_flag_clear( ADC_INT_FLAG_WDE);
```

## adc\_interrupt\_enable

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

**Table 3-30. Function adc\_interrupt\_enable**

<b>Function name</b>	adc_interrupt_enable
<b>Function prototype</b>	void adc_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable ADC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the adc interrupt
ADC_INT_WDE	analog watchdog interrupt
ADC_INT_EOC	end of group conversion interrupt
ADC_INT_EOIC	end of inserted group conversion interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC analog watchdog interrupt */

adc_interrupt_enable(ADC_INT_WDE);
```

### adc\_interrupt\_disable

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

**Table 3-31. Function adc\_interrupt\_disable**

<b>Function name</b>	adc_interrupt_disable
<b>Function prototype</b>	void adc_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	Disable ADC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the adc interrupt
ADC_INT_WDE	analog watchdog interrupt
ADC_INT_EOC	end of group conversion interrupt
ADC_INT_EOIC	end of inserted group conversion interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable ADC interrupt */

adc_interrupt_disable( ADC_INT_WDE);
```

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

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

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

<b>Function name</b>	adc_watchdog_single_channel_enable
<b>Function prototype</b>	void adc_watchdog_single_channel_enable(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>channel</b>	the selected ADC channel
ADC_CHANNEL_x	ADC Channelx(x=0..18)
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

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

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

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

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

Function name	adc_watchdog_group_channel_enable
Function prototype	void adc_watchdog_group_channel_enable(uint8_t channel_group);
Function descriptions	configure ADC analog watchdog group channel
Precondition	-
The called functions	-
Input parameter{in}	
channel_group	the channel group use analog watchdog
ADC_REGULAR_CHANNEL	regular channel group
ADC_INSERTED_CHANNEL	inserted channel group
ADC_REGULAR_INSERTED_CHANNEL	both regular and inserted group
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### adc\_watchdog\_disable

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

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

Function name	adc_watchdog_disable
Function prototype	void adc_watchdog_disable(void);
Function descriptions	disable ADC analog watchdog
Precondition	-

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

Example:

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

```
adc_watchdog_disable(ADC0);
```

### adc\_watchdog\_threshold\_config

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

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

Function name	adc_watchdog_threshold_config
Function prototype	void adc_watchdog_threshold_config(uint16_t low_threshold, uint16_t high_threshold);
Function descriptions	configure ADC analog watchdog threshold
Precondition	-
The called functions	-
Input parameter{in}	
low_threshold	analog watchdog low threshold, 0..4095
Input parameter{in}	
high_threshold	analog watchdog high threshold, 0..4095
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

## 3.3. CEC

Consumer Electronics Control (CEC) belongs to a part of HDMI (High-Definition Multimedia Interface) standard. CEC, as a kind of protocol, provides the advanced control functions of all kinds of audio-visual products in a user environment. The CEC registers are listed in chapter [3.3.1](#), the CEC firmware functions are introduced in chapter [3.3.2](#).

### 3.3.1. Descriptions of Peripheral registers

CEC registers are listed in the table shown as below:

**Table 3-36. CEC Registers**

Registers	Descriptions
CEC_CTL	Control register
CEC_CFG	Configuration register
CEC_TDATA	Transmit data register
CEC_RDATA	Receive data register
CEC_INTF	Interrupt Flag Register
CEC_INTEN	Interrupt enable register

### 3.3.2. Descriptions of Peripheral functions

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

**Table 3-37. CEC firmware function**

Function name	Function description
cec_deinit	reset HDMI-CEC controller
cec_init	configure signal free time,the signal free time counter start option,own address
cec_error_config	configure generate Error-bit, whether stop receive message when detected bit rising error
cec_enable	enable HDMI-CEC controller
cec_disable	disable HDMI-CEC controller
cec_transmission_start	start CEC message transmission
cec_transmission_end	end CEC message transmission
cec_listen_mode_enable	enable CEC listen mode
cec_listen_mode_disable	disable CEC listen mode
cec_own_address_config	configure and clear own address
cec_sft_config	configure signal free time and the signal free time counter start option
cec_generate_errorbit_config	configure generate Error-bit when detected some abnormal situation or not
cec_stop_receive_bre_config	whether stop receive message when detected bit rising error
cec_reception_tolerance_enable	enable reception bit timing tolerance
cec_reception_tolerance_disable	disable reception bit timing tolerance
cec_data_send	send a data by the CEC peripheral
cec_data_receive	receive a data by the CEC peripheral
cec_flag_get	get CEC status
cec_flag_clear	clear CEC status
cec_interrupt_enable	enable CEC interrupt

Function name	Function description
cec_interrupt_disable	Disable CEC interrupt
cec_interrupt_flag_get	get CEC interrupt flag
cec_interrupt_flag_clear	clear CEC interrupt flag

### cec\_deinit

The description of cec\_deinit is shown as below:

**Table 3-38. Function cec\_deinit**

Function name	cec_deinit
Function prototype	void cec_deinit(void);
Function descriptions	reset HDMI-CEC controller
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset CEC */
cec_deinit();
```

### cec\_init

The description of cec\_init is shown as below:

**Table 3-39. Function cec\_init**

Function name	cec_init
Function prototype	void cec_init(uint32_t sftmopt, uint32_t sft, uint32_t address);
Function descriptions	configure signal free time,the signal free time counter start option,own address
Precondition	-
The called functions	-
Input parameter{in}	
sftopt	signal free time counter start option
CEC_SFT_START_STAOM	signal free time counter starts counting when STAOM is asserted
CEC_SFT_START_LAST	signal free time counter starts automatically after transmission/reception end
Input parameter{in}	

<b>sft</b>	signal free time
<i>CEC_SFT_PROTOCOL_PERIOD</i>	the signal free time will perform as HDMI-CEC protocol description
<i>CEC_SFT_1POINT5_PERIOD</i>	1.5 nominal data bit periods
<i>CEC_SFT_2POINT5_PERIOD</i>	2.5 nominal data bit periods
<i>CEC_SFT_3POINT5_PERIOD</i>	3.5 nominal data bit periods
<i>CEC_SFT_4POINT5_PERIOD</i>	4.5 nominal data bit periods
<i>CEC_SFT_5POINT5_PERIOD</i>	5.5 nominal data bit periods
<i>CEC_SFT_6POINT5_PERIOD</i>	6.5 nominal data bit periods
<i>CEC_SFT_7POINT5_PERIOD</i>	7.5 nominal data bit periods
<b>Input parameter{in}</b>	
<b>address</b>	own address
<i>CEC_OWN_ADDRESS_CLEAR</i>	own address is cleared
<i>CEC_OWN_ADDRESS_x</i>	own address is x
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* init CEC*/
```

```
cec_init(CEC_SFT_START_STAOM, CEC_SFT_PROTOCOL_PERIOD,
CEC_OWN_ADDRESS0);
```

### cec\_error\_config

The description of cec\_error\_config is shown as below:

**Table 3-40. Function cec\_error\_config**

<b>Function name</b>	cec_error_config
<b>Function prototype</b>	void cec_error_config(uint32_t broadcast, uint32_t singlecast_lbpe, uint32_t singlecast_bre, uint32_t rxbretp);
<b>Function descriptions</b>	configure generate Error-bit when detected some abnormal situation or not, whether stop receive message when detected bit rising error

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>broadcast</b>	generate an Error-bit in broadcast message or not
<i>CEC_BROADCAST_ERROR_BIT_ON</i>	generate Error-bit in broadcast
<i>CEC_BROADCAST_ERROR_BIT_OFF</i>	do not generate Error-bit in broadcast
<b>Input parameter{in}</b>	
<b>singlecast_lbpe</b>	generate an Error-bit when detected BPLE in singlecast
<i>CEC_LONG_PERIOD_ERROR_BIT_ON</i>	generate Error-bit on long bit period error
<i>CEC_LONG_PERIOD_ERROR_BIT_OFF</i>	do not generate Error-bit on long bit period error
<b>Input parameter{in}</b>	
<b>singlecast_bre</b>	generate an Error-bit when detected BRE in singlecast
<i>CEC_RISING_PERIOD_ERROR_BIT_ON</i>	generate Error-bit on bit rising error
<i>CEC_RISING_PERIOD_ERROR_BIT_OFF</i>	do not generate Error-bit on bit rising error
<b>Input parameter{in}</b>	
<b>rxbrestp</b>	whether stop receive message when detected BRE
<i>CEC_STOP_RISING_ERROR_BIT_ON</i>	stop reception when detected bit rising error
<i>CEC_STOP_RISING_ERROR_BIT_OFF</i>	do not stop reception when detected bit rising error
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

/\* configure generate Error-bit when detected some abnormal situation or not, whether stop receive message when detected bit rising error \*/

```
cec_error_config(CEC_BROADCAST_ERROR_BIT_ON,
CEC_LONG_PERIOD_ERROR_BIT_ON, CEC_RISING_PERIOD_ERROR_BIT_ON,
CEC_STOP_RISING_ERROR_BIT_ON);
```

### **cec\_enable**

The description of cec\_enable is shown as below:



**Table 3-41. Function cec\_enable**

<b>Function name</b>	cec_enable
<b>Function prototype</b>	void cec_enable (void);
<b>Function descriptions</b>	enable HDMI-CEC controller
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable HDMI-CEC controller */
cec_enable();
```

### cec\_disable

The description of cec\_disable is shown as below:

**Table 3-42. Function cec\_disable**

<b>Function name</b>	cec_disable
<b>Function prototype</b>	void cec_disable (void);
<b>Function descriptions</b>	disable HDMI-CEC controller
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable HDMI-CEC controller */
cec_disable ();
```

### cec\_transmission\_start

The description of cec\_transmission\_start is shown as below:

**Table 3-43. Function cec\_transmission\_start**

<b>Function name</b>	cec_transmission_start
<b>Function prototype</b>	void cec_transmission_start (void);
<b>Function descriptions</b>	start CEC message transmission
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* start CEC message transmission */
cec_transmission_start ();
```

### cec\_transmission\_end

The description of cec\_transmission\_end is shown as below:

**Table 3-44. Function cec\_transmission\_end**

<b>Function name</b>	cec_transmission_end
<b>Function prototype</b>	void cec_transmission_end (void);
<b>Function descriptions</b>	end CEC message transmission
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* end CEC message transmission */
cec_transmission_end ();
```

### cec\_listen\_mode\_enable

The description of cec\_listen\_mode\_enable is shown as below:

Table 3-45. Function cec\_listen\_mode\_enable

Function name	cec_listen_mode_enable
Function prototype	void cec_listen_mode_enable (void);
Function descriptions	enable CEC listen mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable CEC listen mode */
cec_listen_mode_enable ();
```

### cec\_listen\_mode\_disable

The description of cec\_listen\_mode\_disable is shown as below:

Table 3-46. Function cec\_listen\_mode\_disable

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

Example:

```
/* disable CEC listen mode */
cec_listen_mode_disable ();
```

### cec\_own\_address\_config

The description of cec\_own\_address\_config is shown as below:

Table 3-47. Function cec\_own\_address\_config

<b>Function name</b>	cec_own_address_config
<b>Function prototype</b>	void cec_own_address_config(uint32_t address);
<b>Function descriptions</b>	configure and clear own address.the controller can be configured to multiple own address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>address</b>	own address
CEC_OWN_ADDRESS_CLEAR	own address is cleared
CEC_OWN_ADDRESS_x	own address is x
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure and clear own address.the controller can be configured to multiple own address
*/
```

```
cec_own_address_config (CEC_OWN_ADDRESS_CLEAR);
```

### cec\_sft\_config

The description of cec\_sft\_config is shown as below:

Table 3-48. Function cec\_sft\_config

<b>Function name</b>	cec_sft_config
<b>Function prototype</b>	void cec_sft_config(uint32_t sftmopt, uint32_t sft);
<b>Function descriptions</b>	configure signal free time and the signal free time counter start option
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sftmopt</b>	signal free time counter start option
CEC_SFT_START_STAOM	signal free time counter starts counting when STAOM is asserted
CEC_SFT_START_LAST	signal free time counter starts automatically after transmission/reception end
<b>Input parameter{in}</b>	
<b>sft</b>	signal free time
CEC_SFT_PROTOCOL_PERIOD	the signal free time will perform as HDMI-CEC protocol description

<i>CEC_SFT_1POINT5_P ERIOD</i>	1.5 nominal data bit periods
<i>CEC_SFT_2POINT5_P ERIOD</i>	2.5 nominal data bit periods
<i>CEC_SFT_3POINT5_P ERIOD</i>	3.5 nominal data bit periods
<i>CEC_SFT_4POINT5_P ERIOD</i>	4.5 nominal data bit periods
<i>CEC_SFT_5POINT5_P ERIOD</i>	5.5 nominal data bit periods
<i>CEC_SFT_6POINT5_P ERIOD</i>	6.5 nominal data bit periods
<i>CEC_SFT_7POINT5_P ERIOD</i>	7.5 nominal data bit periods
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure signal free time and the signal free time counter start option */
```

```
cec_sft_config (CEC_SFT_START_STAOM, CEC_SFT_PROTOCOL_PERIOD);
```

## cec\_generate\_errorbit\_config

The description of cec\_generate\_errorbit\_config is shown as below:

**Table 3-49. Function cec\_generate\_errorbit\_config**

<b>Function name</b>	cec_generate_errorbit_config
<b>Function prototype</b>	void cec_generate_errorbit_config(uint32_t broadcast, uint32_t singlecast_lbpe, uint32_t singlecast_bre);
<b>Function descriptions</b>	configure generate Error-bit when detected some abnormal situation or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>broadcast</b>	generate Error-bit in broadcast or not
<i>CEC_BROADCAST_E RROR_BIT_ON</i>	generate Error-bit in broadcast
<i>CEC_BROADCAST_E RROR_BIT_OFF</i>	do not generate Error-bit in broadcast
<b>Input parameter{in}</b>	
<b>singlecast_lbpe</b>	generate Error-bit on long bit period error or not
<i>CEC_LONG_PERIOD_</i>	generate Error-bit on long bit period error

<i>ERROR_BIT_ON</i>	
<i>CEC_LONG_PERIOD_ERROR_BIT_OFF</i>	do not generate Error-bit on long bit period error
<b>Input parameter{in}</b>	
<b>singlecast_bre</b>	generate Error-bit on bit rising error or not
<i>CEC_RISING_PERIOD_ERROR_BIT_ON</i>	generate Error-bit on bit rising error
<i>CEC_RISING_PERIOD_ERROR_BIT_OFF</i>	do not generate Error-bit on bit rising error
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure generate Error-bit when detected some abnormal situation or not */
cec_generate_errorbit_config (CEC_BROADCAST_ERROR_BIT_ON,
CEC_LONG_PERIOD_ERROR_BIT_ON, CEC_RISING_PERIOD_ERROR_BIT_ON);
```

### cec\_stop\_receive\_bre\_config

The description of cec\_stop\_receive\_bre\_config is shown as below:

**Table 3-50. Function cec\_stop\_receive\_bre\_config**

<b>Function name</b>	cec_stop_receive_bre_config
<b>Function prototype</b>	void cec_stop_receive_bre_config(uint32_t rxbrestp);
<b>Function descriptions</b>	whether stop receive message when detected bit rising error
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rxbrestp</b>	stop reception when detected bit rising error or not
<i>CEC_STOP_RISING_ERROR_BIT_ON</i>	stop reception when detected bit rising error
<i>CEC_STOP_RISING_ERROR_BIT_OFF</i>	do not stop reception when detected bit rising error
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* whether stop receive message when detected bit rising error */
```

cec\_stop\_receive\_bre\_config (CEC\_STOP\_RISING\_ERROR\_BIT\_ON);

### cec\_reception\_tolerance\_enable

The description of cec\_reception\_tolerance\_enable is shown as below:

**Table 3-51. Function cec\_reception\_tolerance\_enable**

<b>Function name</b>	cec_reception_tolerance_enable
<b>Function prototype</b>	void cec_reception_tolerance_enable (void);
<b>Function descriptions</b>	enable reception bit timing tolerance
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable reception bit timing tolerance */
cec_reception_tolerance_enable ();
```

### cec\_reception\_tolerance\_disable

The description of cec\_reception\_tolerance\_disable is shown as below:

**Table 3-52. Function cec\_reception\_tolerance\_disable**

<b>Function name</b>	cec_reception_tolerance_disable
<b>Function prototype</b>	void cec_reception_tolerance_disable (void);
<b>Function descriptions</b>	disable reception bit timing tolerance
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable reception bit timing tolerance */
cec_reception_tolerance_disable ();
```

## cec\_data\_send

The description of cec\_data\_send is shown as below:

**Table 3-53. Function cec\_data\_send**

<b>Function name</b>	cec_data_send
<b>Function prototype</b>	void cec_data_send(uint8_t data);
<b>Function descriptions</b>	send a data by the CEC peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>data</b>	the data to transmit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* send a data by the CEC peripheral */
```

```
cec_data_send (0x55);
```

## cec\_data\_receive

The description of cec\_data\_receive is shown as below:

**Table 3-54. Function cec\_data\_receive**

<b>Function name</b>	cec_data_receive
<b>Function prototype</b>	uint8_t cec_data_receive(void);
<b>Function descriptions</b>	receive a data by the CEC peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>Uin8_t</b>	the data to receive

Example:

```
/* receive a data by the CEC peripheral */
```

```
uint8_t data = 0;
```

```
data = cec_data_receive ();
```



## cec\_flag\_get

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

**Table 3-55. Function cec\_flag\_get**

<b>Function name</b>	cec_flag_get
<b>Function prototype</b>	FlagStatus cec_flag_get(uint32_t flag);
<b>Function descriptions</b>	get CEC status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag
CEC_FLAG_BR	Rx-byte data received
CEC_FLAG_REND	end of reception
CEC_FLAG_RO	RX overrun
CEC_FLAG_BRE	bit rising error
CEC_FLAG_BPSE	short bit period error
CEC_FLAG_BPLE	long bit period error
CEC_FLAG_RAE	Rx ACK error
CEC_FLAG_ARBF	arbitration lost
CEC_FLAG_TBR	Tx-byte data request
CEC_FLAG_TEND	transmission successfully end
CEC_FLAG_TU	Tx data buffer underrun
CEC_FLAG_TERR	Tx-error
CEC_FLAG_TAERR	Tx ACK error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get CEC_FLAG_BR status */
FlagStatus flag = reset;
flag = cec_flag_get (CEC_INT_BR);
```

## cec\_flag\_clear

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

**Table 3-56. Function cec\_flag\_clear**

<b>Function name</b>	cec_flag_clear
<b>Function prototype</b>	void cec_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear CEC status

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag
<i>CEC_FLAG_BR</i>	Rx-byte data received
<i>CEC_FLAG_REND</i>	end of reception
<i>CEC_FLAG_RO</i>	RX overrun
<i>CEC_FLAG_BRE</i>	bit rising error
<i>CEC_FLAG_BPSE</i>	short bit period error
<i>CEC_FLAG_BPLE</i>	long bit period error
<i>CEC_FLAG_RAE</i>	Rx ACK error
<i>CEC_FLAG_ARBF</i>	arbitration lost
<i>CEC_FLAG_TBR</i>	Tx-byte data request
<i>CEC_FLAG_TEND</i>	transmission successfully end
<i>CEC_FLAG_TU</i>	Tx data buffer underrun
<i>CEC_FLAG_TERR</i>	Tx-error
<i>CEC_FLAG_TAERR</i>	Tx ACK error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear CEC_FLAG_BR status */
```

```
cec_flag_get (CEC_FLAG_BR);
```

### cec\_interrupt\_enable

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

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

<b>Function name</b>	cec_interrupt_enable
<b>Function prototype</b>	void cec_interrupt_enable(uint32_t flag);
<b>Function descriptions</b>	enable interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag
<i>CEC_INT_BR</i>	enable Rx-byte data received interrupt
<i>CEC_INT_REND</i>	enable end of reception interrupt
<i>CEC_INT_RO</i>	enable RX overrun interrupt
<i>CEC_INT_BRE</i>	enable bit rising error interrupt

<i>CEC_INT_BPSE</i>	enable short bit period error interrupt
<i>CEC_INT_BPLE</i>	enable long bit period error interrupt
<i>CEC_INT_RAE</i>	enable Rx ACK error interrupt
<i>CEC_INT_ARBF</i>	enable arbitration lost interrupt
<i>CEC_INT_TBR</i>	enable Tx-byte data request interrupt
<i>CEC_INT_TEND</i>	enable transmission successfully end interrupt
<i>CEC_INT_TU</i>	enable Tx data buffer underrun interrupt
<i>CEC_INT_TERR</i>	enable Tx-error interrupt
<i>CEC_INT_TAERR</i>	enable Tx ACK error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable CEC_INT_BR interrupt */
cec_interrupt_enable (CEC_INT_BR);
```

### cec\_interrupt\_disable

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

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

<b>Function name</b>	cec_interrupt_disable
<b>Function prototype</b>	void cec_interrupt_disable (uint32_t flag);
<b>Function descriptions</b>	disable interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag
<i>CEC_INT_BR</i>	disable Rx-byte data received interrupt
<i>CEC_INT_REND</i>	disable end of reception interrupt
<i>CEC_INT_RO</i>	disable RX overrun interrupt
<i>CEC_INT_BRE</i>	disable bit rising error interrupt
<i>CEC_INT_BPSE</i>	disable short bit period error interrupt
<i>CEC_INT_BPLE</i>	disable long bit period error interrupt
<i>CEC_INT_RAE</i>	disable Rx ACK error interrupt
<i>CEC_INT_ARBF</i>	disable arbitration lost interrupt
<i>CEC_INT_TBR</i>	disable Tx-byte data request interrupt
<i>CEC_INT_TEND</i>	disable transmission successfully end interrupt
<i>CEC_INT_TU</i>	disable Tx data buffer underrun interrupt
<i>CEC_INT_TERR</i>	disable Tx-error interrupt

<i>CEC_INT_TAERR</i>	disable Tx ACK error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable CEC_INT_BR interrupt */
cec_interrupt_disable (CEC_INT_BR);
```

## cec\_interrupt\_flag\_get

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

**Table 3-59. Function cec\_interrupt\_flag\_get**

<b>Function name</b>	cec_interrupt_flag_get
<b>Function prototype</b>	FlagStatus cec_interrupt_flag_get(uint32_t flag);
<b>Function descriptions</b>	get CEC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag
<i>CEC_INT_FLAG_BR</i>	Rx-byte data received
<i>CEC_INT_FLAG_REND</i>	end of reception
<i>CEC_INT_FLAG_RO</i>	RX overrun
<i>CEC_INT_FLAG_BRE</i>	bit rising error
<i>CEC_INT_FLAG_BPSE</i>	short bit period error
<i>CEC_INT_FLAG_BPLE</i>	long bit period error
<i>CEC_INT_FLAG_RAE</i>	Rx ACK error
<i>CEC_INT_FLAG_ARBF</i>	arbitration lost
<i>CEC_INT_FLAG_TBR</i>	Tx-byte data request
<i>CEC_INT_FLAG_TEND</i>	transmission successfully end
<i>CEC_INT_FLAG_TU</i>	Tx data buffer underrun
<i>CEC_INT_FLAG_TERR</i>	Tx-error
<i>CEC_INT_FLAG_TAERR</i>	Tx ACK error flag
<i>R</i>	
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get CEC_INT_FLAG_BR status */
```

```
FlagStatus flag = reset;
```

```
flag = cec_interrupt_flag_get (CEC_INT_FLAG_BR);
```

### cec\_interrupt\_flag\_clear

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

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

<b>Function name</b>	cec_interrupt_flag_clear
<b>Function prototype</b>	void cec_interrupt_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear CEC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag
CEC_INT_FLAG_BR	Rx-byte data received
CEC_INT_FLAG_REND	end of reception
CEC_INT_FLAG_RO	RX overrun
CEC_INT_FLAG_BRE	bit rising error
CEC_INT_FLAG_BPSE	short bit period error
CEC_INT_FLAG_BPLE	long bit period error
CEC_INT_FLAG_RAE	Rx ACK error
CEC_INT_FLAG_ARBF	arbitration lost
CEC_INT_FLAG_TBR	Tx-byte data request
CEC_INT_FLAG_TEND	transmission successfully end
CEC_INT_FLAG_TU	Tx data buffer underrun
CEC_INT_FLAG_TERR	Tx-error
CEC_INT_FLAG_TAERR	Tx ACK error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear CEC_INT_FLAG_BR status */
```

```
cec_interrupt_flag_clear(CEC_INT_FLAG_BR);
```

## 3.4. CMP

The general purpose CMP can work either standalone (all terminal are available on I/Os) or together with the timers. It can be used to wake up the MCU from low-power mode by an analog signal, provide a trigger source when an analog signal is in a certain condition, achieve some current control by working together with a PWM output of a timer and the DAC. The CMP registers are listed in chapter [3.4.1](#), the CMP firmware functions are introduced in chapter [3.4.2](#).

### 3.4.1. Descriptions of Peripheral registers

CMP registers are listed in the table shown as below:

**Table 3-61. CMP registers**

Registers	Descriptions
CMP_CS	CMP control and status register

### 3.4.2. Descriptions of Peripheral functions

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

**Table 3-62. CMP firmware function**

Function name	Function description
cmp_deinit	CMP deinit
cmp_mode_init	CMP mode init
cmp_output_init	CMP output init
cmp_enable	enable CMP
cmp_disable	disable CMP
cmp_switch_enable	enable the switch mode
cmp_switch_disable	disable the switch mode
cmp_window_enable	enable the window mode
cmp_window_disable	disable the window mode
cmp_lock_enable	lock the CMP
cmp_output_level_get	get output level

#### Enum cmp\_enum

**Table 3-63. Enum cmp\_enum**

Member name	Function description
CMP0	comparator 0
CMP1	comparator 1

## cmp\_deinit

The description of cmp\_deinit is shown as below:

**Table 3-64. Function cmp\_deinit**

<b>Function name</b>	cmp_deinit
<b>Function prototype</b>	void cmp_deinit(cmp_enum cmp_periph);
<b>Function descriptions</b>	CMP deinit
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-63. Enum cmp_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* deinitialize CMP0 */
cmp_deinit(CMP0);
```

## cmp\_mode\_init

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

**Table 3-65. Function cmp\_mode\_init**

<b>Function name</b>	cmp_mode_init
<b>Function prototype</b>	void cmp_mode_init(cmp_enum cmp_periph, uint32_t operating_mode, uint32_t inverting_input, uint32_t output_hysteresis);
<b>Function descriptions</b>	CMP mode init
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-63. Enum cmp_enum</a>
<b>Input parameter{in}</b>	
<b>operating_mode</b>	operating mode
<b>CMP_MODE_HIGHSP EED</b>	high speed mode
<b>CMP_MODE_MIDDLE SPEED</b>	medium speed mode
<b>CMP_MODE_LOWSPE ED</b>	low speed mode
<b>CMP_MODE_VERYLO</b>	very-low speed mode

WSPEED	
Input parameter{in}	
inverting_input	inverting input select
CMP_INVERTING_INP UT_1_4VREFINT	VREFINT *1/4 input
CMP_INVERTING_INP UT_1_2VREFINT	VREFINT *1/2 input
CMP_INVERTING_INP UT_3_4VREFINT	VREFINT *3/4 input
CMP_INVERTING_INP UT_VREFINT	VREFINT input
CMP_INVERTING_INP UT_DAC0_OUT0	PA4 (DAC0_OUT0) input
CMP_INVERTING_INP UT_PA5	PA5 input
CMP_INVERTING_INP UT_PA0_PA2	PA0 input for CMP0, PA2 input for CMP1
Input parameter{in}	
output_hysteresis	hysteresis level
CMP_HYSTERESIS_N O	output no hysteresis
CMP_HYSTERESIS_L OW	output low hysteresis
CMP_HYSTERESIS_M IDDLE	output middle hysteresis
CMP_HYSTERESIS_HI GH	output high hysteresis
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

### cmp\_output\_init

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

**Table 3-66. Function cmp\_output\_init**

Function name	cmp_output_init
---------------	-----------------



Function prototype	void cmp_output_init(cmp_enum cmp_periph, uint32_t output_selection, uint32_t output_polarity);	
Function descriptions	CMP output init	
Precondition	-	
The called functions	-	
Input parameter{in}		
cmp_periph	refer to enum <a href="#">Table 3-63. Enum cmp_enum</a>	
output_selection	CMP output selection	
CMP_OUTPUT_NONE	CMP output none	
CMP_OUTPUT_TIMER0_BRKIN	CMP output TIMER0 break input	
CMP_OUTPUT_TIMER0_IC0	CMP output TIMER0_CH0 input capture	
CMP_OUTPUT_TIMER0_OCPRECLR	CMP output TIMER0 OCPRE_CLR input	
CMP_OUTPUT_TIMER1_IC3	CMP output TIMER1_CH3 input capture	
CMP_OUTPUT_TIMER1_OCPRECLR	CMP output TIMER1 OCPRE_CLR input	
CMP_OUTPUT_TIMER2_IC0	CMP output TIMER2_CH0 input capture	
CMP_OUTPUT_TIMER2_OCPRECLR	CMP output TIMER2 OCPRE_CLR input	
Input parameter{in}		
output_polarity	CMP output polarity	
CMP_OUTPUT_POLARITY_INVERTED	output is inverted	
CMP_OUTPUT_POLARITY_NONINVERTED	output is not inverted	
Output parameter{out}		
-	-	
Return value		
-	-	

Example:

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

```
cmp_output_init(CMP0, CMP_OUTPUT_TIMER0_IC0, CMP_OUTPUT_POLARITY_NONINVERTED);
```

## cmp\_enable

The description of cmp\_enable is shown as below:

**Table 3-67. Function cmp\_enable**

<b>Function name</b>	cmp_enable
<b>Function prototype</b>	void cmp_enable(cmp_enum cmp_periph);
<b>Function descriptions</b>	enable CMP
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-63. Enum cmp_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## cmp\_disable

The description of cmp\_disable is shown as below:

**Table 3-68. Function cmp\_disable**

<b>Function name</b>	cmp_disable
<b>Function prototype</b>	void cmp_disable(cmp_enum cmp_periph);
<b>Function descriptions</b>	disable CMP
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-63. Enum cmp_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## cmp\_switch\_enable

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

**Table 3-69. Function cmp\_switch\_enable**

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

Example:

```
/* enable the switch mode */
cmp_switch_enable();
```

## cmp\_switch\_disable

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

**Table 3-70. Function cmp\_switch\_disable**

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

Example:

```
/* disable the switch mode */
cmp_switch_disable();
```

## cmp\_window\_enable

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

**Table 3-71. Function cmp\_window\_enable**

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

Example:

```
/* enable the window mode */
cmp_window_enable();
```

## cmp\_window\_disable

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

**Table 3-72. Function cmp\_window\_disable**

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

Example:

```
/* disable the window mode */
cmp_window_disable();
```

## cmp\_lock\_enable

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

**Table 3-73. Function cmp\_lock\_enable**

<b>Function name</b>	cmp_lock_enable
<b>Function prototype</b>	void cmp_lock_enable(cmp_enum cmp_periph);
<b>Function descriptions</b>	lock the CMP
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-63. Enum cmp_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## cmp\_output\_level\_get

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

**Table 3-74. Function cmp\_output\_level\_get**

<b>Function name</b>	cmp_output_level_get
<b>Function prototype</b>	uint32_t cmp_output_level_get(cmp_enum cmp_periph);
<b>Function descriptions</b>	get output level
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-63. Enum cmp_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	the output level
<b>CMP_OUTPUTLEVEL_HIGH</b>	comparator output high
<b>CMP_OUTPUTLEVEL_LOW</b>	comparator output low

Example:

```
uint32_t level;

/* get CMP0 output level */

level = cmp_output_level_get(CMP0);
```

## 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 [0](#).

### 3.5.1. Descriptions of Peripheral registers

CRC registers are listed in the table shown as below:

**Table 3-75. CRC Registers**

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

### 3.5.2. Descriptions of Peripheral functions

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

**Table 3-76. CRC firmware function**

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

#### **crc\_deinit**

The description of crc\_deinit is shown as below:

Table 3-77. Function `crc_deinit`

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

Example:

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

### `crc_reverse_output_data_enable`

The description of `crc_reverse_output_data_enable` is shown as below:

Table 3-78. Function `crc_reverse_output_data_enable`

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

Example:

```
/* enable CRC reverse operation of output data */
crc_reverse_output_data_enable ();
```

### `crc_reverse_output_data_disable`

The description of `crc_reverse_output_data_disable` is shown as below:

**Table 3-79. Function crc\_reverse\_output\_data\_disable**

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

Example:

```
/* disable crc reverse operation of output data */
crc_reverse_output_data_disable ();
```

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

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

**Table 3-80. Function crc\_data\_register\_reset**

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

Example:

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

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

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



Table 3-81. 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 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 3-82. 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 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 3-83. Function crc\_free\_data\_register\_write**

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

Example:

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

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

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

**Table 3-84. Function crc\_init\_data\_register\_write**

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

Example:

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

## crc\_input\_data\_reverse\_config

The description of `crc_input_data_reverse_config` is shown as below:

**Table 3-85. Function `crc_input_data_reverse_config`**

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

Example:

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

```
crc_input_data_reverse_config (CRC_INPUT_DATA_WORD);
```

## crc\_single\_data\_calculate

The description of `crc_single_data_calculate` is shown as below:

**Table 3-86. Function `crc_single_data_calculate`**

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

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 3-87. Function `crc_block_data_calculate`**

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

Example:

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

#define BUFFER_SIZE      6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {

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

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 3-88. DAC Registers**

Register	Descriptions
DAC_CTL0	DACx control register 0
DAC_SWT	DACx software trigger register
DAC_OUT0_R12DH	DACx_OUT0 12-bit right-aligned data holding register
DAC_OUT0_L12DH	DACx_OUT0 12-bit left-aligned data holding register
DAC_OUT0_R8DH	DACx_OUT0 8-bit right-aligned data holding register
DAC_OUT0_DO	DACx_OUT0 data output register
DAC_STAT0	DACx status register 0

### 3.6.2. Descriptions of Peripheral functions

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

**Table 3-89. DAC firmware functions**

Function name	Function description
dac_deinit	deinitialize DAC
dac_enable	enable DAC
dac_disable	disable DAC
dac_dma_enable	enable DAC DMA function
dac_dma_disable	disable DAC DMA function
dac_output_buffer_enable	enable DAC output buffer
dac_output_buffer_disable	disable DAC output buffer
dac_output_value_get	get DAC output value
dac_data_set	set DAC data holding register value
dac_trigger_enable	enable DAC trigger
dac_trigger_disable	disable DAC trigger
dac_trigger_source_config	configure DAC trigger source
dac_software_trigger_enable	enable DAC software trigger
dac_flag_get	get DAC flag
dac_flag_clear	clear DAC flag
dac_interrupt_enable	enable DAC interrupt

Function name	Function description
<code>dac_interrupt_disable</code>	disable DAC interrupt
<code>dac_interrupt_flag_get</code>	get DAC interrupt flag
<code>dac_interrupt_flag_clear</code>	clear DAC interrupt flag

## **dac\_deinit**

The description of `dac_deinit` is shown as below:

**Table 3-90. 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 3-91. 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><code>dac_periph</code></b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b><code>dac_out</code></b>	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0)

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### **dac\_disable**

The description of dac\_disable is shown as below:

**Table 3-92. Function dac\_disable**

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

Example:

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

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

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

**Table 3-93. Function dac\_dma\_enable**

Function name	dac_dma_enable
Function prototype	void dac_dma_enable(uint32_t dac_periph, uint8_t dac_out);
Function descriptions	enable DAC DMA function
Precondition	-

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

Example:

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

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

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

**Table 3-94. Function dac\_dma\_disable**

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

Example:

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



## dac\_output\_buffer\_enable

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

**Table 3-95. Function dac\_output\_buffer\_enable**

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

Example:

```
/* enable DAC0_OUT0 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 3-96. Function dac\_output\_buffer\_disable**

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

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 3-97. Function dac\_output\_value\_get**

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

Example:

```
/* get the DAC0_OUT0 last data output value */
Uin16_t data;
data = dac_output_value_get(DAC0, DAC_OUT0);
```

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

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

**Table 3-98. Function dac\_data\_set**

Function name	dac_data_set
Function prototype	void dac_data_set(uint32_t dac_periph, uint8_t dac_out, uint32_t dac_align, uint16_t data);
Function descriptions	set DAC data holding register value
Precondition	-

The called functions	-
Input parameter{in}	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
<b>dac_out</b>	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0)
Input parameter{in}	
<b>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
Input parameter{in}	
<b>data</b>	data to be loaded (0~4095)
Output parameter{out}	
-	-
Return value	
-	-

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 3-99. Function `dac_trigger_enable`**

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

-	-
---	---

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 3-100. Function `dac_trigger_disable`**

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

Example:

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

```
dac_trigger_disable(DAC0, DAC_OUT0);
```

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

The description of `dac_trigger_source_config` is shown as below:

**Table 3-101. 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)
<b>Input parameter{in}</b>	
<b>triggersource</b>	external trigger of DAC
<i>DAC_TRIGGER_T5_TRGO</i>	TIMER5 TRGO
<i>DAC_TRIGGER_T14_TRGO</i>	TIMER14 TRGO
<i>DAC_TRIGGER_T2_TRGO</i>	TIMER2 TRGO
<i>DAC_TRIGGER_T1_TRGO</i>	TIMER1 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 3-102. Function `dac_software_trigger_enable`**

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

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

Example:

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

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

The description of `dac_flag_get` is shown as below:

**Table 3-103. Function `dac_flag_get`**

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

Example:

```
/* get DAC0 flag */
FlagStatus flag;
flag = dac_flag_get(DAC0, DAC_FLAG_DDUDR0);
```

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

The description of `dac_flag_clear` is shown as below:

**Table 3-104. Function `dac_flag_clear`**

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

<b>Function descriptions</b>	clear DAC flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b>flag</b>	DAC flag
<i>DAC_FLAG_DDUDR0</i>	DACx_OUT0 DMA underrun flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

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

**Table 3-105. Function dac\_interrupt\_enable**

<b>Function name</b>	dac_interrupt_enable
<b>Function prototype</b>	void dac_interrupt_enable(uint32_t dac_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable DAC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b>interrupt</b>	the DAC interrupt
<i>DAC_INT_DDUDR0</i>	DACx_OUT0 DMA underrun interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
dac_interrupt_enable (DAC0, DAC_INT_DDUDRIE0);
```

## dac\_interrupt\_disable

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

**Table 3-106. Function dac\_interrupt\_disable**

<b>Function name</b>	dac_interrupt_disable
<b>Function prototype</b>	void dac_interrupt_disable(uint32_t dac_periph, uint32_t interrupt);
<b>Function descriptions</b>	disable DAC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b>interrupt</b>	the DAC interrupt
<i>DAC_INT_DDUDR0</i>	DACx_OUT0 DMA underrun interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DAC0 interrupt */
dac_interrupt_disable (DAC0, DAC_INT_DDUDRIE0);
```

## dac\_interrupt\_flag\_get

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

**Table 3-107. Function dac\_interrupt\_flag\_get**

<b>Function name</b>	dac_interrupt_flag_get
<b>Function prototype</b>	FlagStatus dac_interrupt_flag_get(uint32_t dac_periph, uint32_t int_flag);
<b>Function descriptions</b>	get DAC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b>int_flag</b>	DAC interrupt flag
<i>DAC_INT_FLAG_DDUDR0</i>	DACx_OUT0 DMA underrun interrupt flag
<b>Output parameter{out}</b>	



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

Example:

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

```
FlagStatus flag;
```

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

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

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

**Table 3-108. Function dac\_interrupt\_flag\_clear**

<b>Function name</b>	dac_interrupt_flag_clear
<b>Function prototype</b>	void dac_interrupt_flag_clear(uint32_t dac_periph, uint32_t int_flag);
<b>Function descriptions</b>	clear DAC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b>int_flag</b>	DAC interrupt flag
<i>DAC_INT_FLAG_DDUDR0</i>	DACx_OUT0 DMA underrun interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

## **3.7. DBG**

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

### 3.7.1. Descriptions of Peripheral registers

DBG registers are listed in the table shown as below:

**Table 3-109. 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 3-110. 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 3-111. Enum dbg\_periph\_enum**

Member name	Function description
DBG_FWDGT_HOLD	debug FWDGT kept when core is halted
DBG_WWDGT_HOLD	debug WWDGT kept when core is halted
DBG_TIMER0_HOLD	hold TIMER0 counter when core is halted
DBG_TIMER1_HOLD	hold TIMER1 counter when core is halted
DBG_TIMER2_HOLD	hold TIMER2 counter when core is halted
DBG_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_I2C2_HOLD	hold I2C2 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 3-112. 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 3-113. 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 3-114. Function dbg\_low\_power\_enable**

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

Example:

```
/* enable low power behavior when the mcu is in debug mode */
dbg_low_power_enable(DBG_LOW_POWER_SLEEP);
```

## dbg\_low\_power\_disable

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

**Table 3-115. 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_SLEEP</i> <i>TANDBY</i>	keep debugger connection during standby mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

### dbg\_periph\_enable

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

**Table 3-116. 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 3-111. 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=0,1,2,5,13,14,15,16, 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:

```
/* 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 3-117. 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 3-111. Enum dbg_periph_enum</a>
<b>DBG_FWDGT_HOLD</b>	debug FWDGT kept when core is halted
<b>DBG_WWDGT_HOLD</b>	debug WWDGT kept when core is halted
<b>DBG_TIMERx_HOLD</b>	x=0,1,2,5,13,14,15,16, hold TIMERx counter when core is halted
<b>DBG_I2Cx_HOLD</b>	x=0,1, hold I2Cx smbus when core is halted
<b>DBG_RTC_HOLD</b>	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 3-118. DMA Registers**

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

### 3.8.2. Descriptions of Peripheral functions

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

**Table 3-119. DMA firmware function**

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

#### `dma_channel_enum`

**Table 3-120. Enum `dma_channel_enum`**

Member name	Function description
<code>DMA_CH0</code>	DMA Channel0
<code>DMA_CH1</code>	DMA Channel1
<code>DMA_CH2</code>	DMA Channel2

DMA_CH3	DMA Channel3
DMA_CH4	DMA Channel4
DMA_CH5	DMA Channel5
DMA_CH6	DMA Channel6

### Structure dma\_parameter\_struct

**Table 3-121. Structure dma\_parameter\_struct**

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

### dma\_deinit

The description of dma\_deinit is shown as below:

**Table 3-122. Function dma\_deinit**

<b>Function name</b>	dma_deinit
<b>Function prototype</b>	void dma_deinit(dma_channel_enum channelx);
<b>Function descriptions</b>	deinitialize DMA a channel registers
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
<i>DMA_CHx (x=0..6)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* DMA channel0 initialize */

dma_deinit(DMA_CH0);

```



## dma\_struct\_para\_init

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

**Table 3-123. Function dma\_struct\_para\_init**

<b>Function name</b>	dma_struct_para_init
<b>Function prototype</b>	void dma_struct_para_init(dma_parameter_struct* init_struct);
<b>Function descriptions</b>	initialize the parameters of DMA struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-121. Structure dma_parameter_struct.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## dma\_init

The description of dma\_init is shown as below:

**Table 3-124. Function dma\_init**

<b>Function name</b>	dma_init
<b>Function prototype</b>	void dma_init(dma_channel_enum channelx, dma_parameter_struct init_struct);
<b>Function descriptions</b>	initialize DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum.</a>
<i>DMA_CHx (x=0..6)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-121. Structure dma_parameter_struct.</a>
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* DMA 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(DMA_CH0, dma_init_struct);
```

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

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

**Table 3-125. Function dma\_circulation\_enable**

Function name	dma_circulation_enable
Function prototype	void dma_circulation_enable(dma_channel_enum channelx);
Function descriptions	enable DMA circulation mode
Precondition	-
The called functions	-
Input parameter{in}	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
DMA_CHx (x=0..6)	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMA channel0 circulation mode */
```

```
dma_circulation_enable(DMA_CH0);
```

### dma\_circulation\_disable

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

**Table 3-126. Function dma\_circulation\_disable**

<b>Function name</b>	dma_circulation_disable
<b>Function prototype</b>	void dma_circulation_disable(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>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
<i>DMA_CHx (x=0..6)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA channel0 circulation mode */
```

```
dma_circulation_disable(DMA_CH0);
```

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

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

**Table 3-127. 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(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>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
<i>DMA_CHx (x=0..6)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMA channel0 memory to memory mode */
```

```
dma_memory_to_memory_enable(DMA_CH0);
```

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

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

**Table 3-128. Function dma\_memory\_to\_memory\_disable**

Function name	dma_memory_to_memory_disable
Function prototype	void dma_memory_to_memory_disable(dma_channel_enum channelx);
Function descriptions	disable memory to memory mode
Precondition	-
The called functions	-
Input parameter{in}	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
DMA_CHx (x=0..6)	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*disable DMA channel0 memory to memory mode */
```

```
dma_memory_to_memory_disable(DMA_CH0);
```

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

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

**Table 3-129. Function dma\_channel\_enable**

Function name	dma_channel_enable
Function prototype	void dma_channel_enable(dma_channel_enum channelx);
Function descriptions	enable DMA channel
Precondition	-
The called functions	-
Input parameter{in}	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
DMA_CHx (x=0..6)	DMA channel selection
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* enable DMA channel0 */
```

```
dma_channel_enable(DMA_CH0);
```

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

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

**Table 3-130. Function dma\_channel\_disable**

Function name	dma_channel_disable
Function prototype	void dma_channel_disable(dma_channel_enum channelx);
Function descriptions	disable DMA channel
Precondition	-
The called functions	-
Input parameter{in}	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
DMA_CHx (x=0..6)	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMA channel0 */
```

```
dma_channel_disable(DMA_CH0);
```

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

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

**Table 3-131. Function dma\_periph\_address\_config**

Function name	dma_periph_address_config
Function prototype	void dma_periph_address_config(dma_channel_enum channelx, uint32_t address);
Function descriptions	set DMA peripheral base address
Precondition	-
The called functions	-
Input parameter{in}	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum</a>

	<a href="#"><u><b><i>dma_channel_enum</i></b></u></a>
<i>DMA_CHx (x=0..6)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>address</b>	peripheral base address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA channel0 periph address */

#define BANK0_WRITE_START_ADDR          ((uint32_t)0x08004000)

dma_periph_address_config(DMA_CH0, BANK0_WRITE_START_ADDR);
```

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

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

**Table 3-132. Function dma\_memory\_address\_config**

<b>Function name</b>	dma_memory_address_config
<b>Function prototype</b>	void dma_memory_address_config(dma_channel_enum channelx, uint32_t address);
<b>Function descriptions</b>	set DMA memory base address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#"><u><b>Table 3-120. Enum dma_channel_enum</b></u></a>
<i>DMA_CHx (x=0..6)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>address</b>	memory base address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA channel0 memory address */

uint8_t g_destbuf[TRANSFER_NUM];

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

## dma\_transfer\_number\_config

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

**Table 3-133. Function dma\_transfer\_number\_config**

<b>Function name</b>	dma_transfer_number_config
<b>Function prototype</b>	void dma_transfer_number_config( dma_channel_enum channelx, uint32_t number);
<b>Function descriptions</b>	set the number of remaining data to be transferred by the DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
<i>DMA_CHx (x=0..6)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>number</b>	data transfer number(0x00000000-0x0000FFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA channel0 transfer number */
#define TRANSFER_NUM                0x400
dma_transfer_number_config(DMA_CH0, TRANSFER_NUM);
```

## dma\_transfer\_number\_get

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

**Table 3-134. Function dma\_transfer\_number\_get**

<b>Function name</b>	dma_transfer_number_get
<b>Function prototype</b>	uint32_t dma_transfer_number_get(dma_channel_enum channelx);
<b>Function descriptions</b>	get the number of remaining data to be transferred by the DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
<i>DMA_CHx (x=0..6)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-

Return value	
uint32_t	DMA data transmission remaining quantity (0x00000000-0x0000FFFF)

Example:

```
/* get DMA 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 3-135. Function dma\_priority\_config**

Function name	dma_priority_config
Function prototype	void dma_priority_config(dma_channel_enum channelx, uint32_t priority);
Function descriptions	configure priority level of DMA channel
Precondition	-
The called functions	-
Input parameter{in}	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
DMA_CHx (x=0..6)	DMA channel selection
Input parameter{in}	
priority	priority Level of this channel
DMA_PRIORITY_LOW	low priority
DMA_PRIORITY_MEDIUM	medium priority
DMA_PRIORITY_HIGH	high priority
DMA_PRIORITY_ULTRA_HIGH	ultra high priority
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 priority */

dma_priority_config(DMA_CH0, DMA_PRIORITY_ULTRA_HIGH);
```

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

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



Table 3-136. Function dma\_memory\_width\_config

Function name	dma_memory_width_config
Function prototype	void dma_memory_width_config( dma_channel_enum channelx, uint32_t mwidth);
Function descriptions	configure transfer data size of memory
Precondition	-
The called functions	-
Input parameter{in}	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
DMA_CHx (x=0..6)	DMA channel selection
Input parameter{in}	
mwidth	transfer data width of memory
DMA_MEMORY_WIDTH_8BIT	transfer data width of memory is 8-bit
DMA_MEMORY_WIDTH_16BIT	transfer data width of memory is 16-bit
DMA_MEMORY_WIDTH_32BIT	transfer data width of memory is 32-bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 memory width */
```

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

### dma\_periph\_width\_config

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

Table 3-137. Function dma\_periph\_width\_config

Function name	dma_periph_width_config
Function prototype	void dma_periph_width_config(dma_channel_enum channelx, uint32_t pwidth);
Function descriptions	configure transfer data width of peripheral
Precondition	-
The called functions	-
Input parameter{in}	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
DMA_CHx (x=0..6)	DMA channel selection

Input parameter{in}	
<b>pwidth</b>	transfer data width of peripheral
<i>DMA_PERIPHERAL_WIDTH_8BIT</i>	transfer data width of peripheral is 8-bit
<i>DMA_PERIPHERAL_WIDTH_16BIT</i>	transfer data width of peripheral is 16-bit
<i>DMA_PERIPHERAL_WIDTH_32BIT</i>	transfer data width of peripheral is 32-bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 periph width */
dma_periph_width_config(DMA_CH0, DMA_PERIPHERAL_WIDTH_8BIT);
```

### **dma\_memory\_increase\_enable**

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

**Table 3-138. Function dma\_memory\_increase\_enable**

<b>Function name</b>	dma_memory_increase_enable
<b>Function prototype</b>	void dma_memory_increase_enable(dma_channel_enum channelx);
<b>Function descriptions</b>	enable next address increasement algorithm of memory
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
<i>DMA_CHx (x=0..6)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

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

**Table 3-139. Function dma\_memory\_increase\_disable**

<b>Function name</b>	dma_memory_increase_disable
<b>Function prototype</b>	void dma_memory_increase_disable(dma_channel_enum channelx);
<b>Function descriptions</b>	disable next address increasement algorithm of memory
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
<i>DMA_CHx (x=0..6)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA channel0 memory increase */
dma_memory_increase_disable(DMA_CH0);
```

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

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

**Table 3-140. Function dma\_periph\_increase\_enable**

<b>Function name</b>	dma_periph_increase_enable
<b>Function prototype</b>	void dma_periph_increase_enable(dma_channel_enum channelx);
<b>Function descriptions</b>	enable next address increasement algorithm of peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
<i>DMA_CHx (x=0..6)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMA channel0 periph increase */
dma_periph_increase_enable(DMA_CH0);
```

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

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

**Table 3-141. Function dma\_periph\_increase\_disable**

<b>Function name</b>	dma_periph_increase_disable
<b>Function prototype</b>	void dma_periph_increase_disable(dma_channel_enum channelx);
<b>Function descriptions</b>	disable next address increasement algorithm of peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
DMA_CHx (x=0..6)	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA channel0 periph increase */
dma_periph_increase_disable(DMA_CH0);
```

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

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

**Table 3-142. Function dma\_transfer\_direction\_config**

<b>Function name</b>	dma_transfer_direction_config
<b>Function prototype</b>	void dma_transfer_direction_config(dma_channel_enum channelx, uint8_t direction);
<b>Function descriptions</b>	configure the direction of data transfer on the channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
DMA_CHx (x=0..6)	DMA channel selection
<b>Input parameter{in}</b>	
<b>direction</b>	specify the direction of data transfer
DMA_PERIPHERAL_TO_MEMORY	read from peripheral and write to memory
DMA_MEMORY_TO_PERIPHERAL	read from memory and write to peripheral

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

Example:

```
/* configure DMA channel0 transfer direction */
```

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

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

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

**Table 3-143. Function dma\_flag\_get**

<b>Function name</b>	dma_flag_get
<b>Function prototype</b>	FlagStatus dma_flag_get(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>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
<i>DMA_CHx (x=0..6)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>DMA_FLAG_G</i>	global interrupt flag of channel
<i>DMA_FLAG_FTF</i>	full transfer finish flag of channel
<i>DMA_FLAG_HTF</i>	half transfer finish flag of channel
<i>DMA_FLAG_ERR</i>	error flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get DMA channel0 flag */
```

```
FlagStatus flag = RESET;
```

```
flag = dma_flag_get(DMA_CH0, DMA_FLAG_FTF);
```

## dma\_flag\_clear

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

**Table 3-144. Function dma\_flag\_clear**

<b>Function name</b>	dma_flag_clear
<b>Function prototype</b>	void dma_flag_clear(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>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
<i>DMA_CHx (x=0..6)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>DMA_FLAG_G</i>	global interrupt flag of channel
<i>DMA_FLAG_FTF</i>	full transfer finish flag of channel
<i>DMA_FLAG_HTF</i>	half transfer finish flag of channel
<i>DMA_FLAG_ERR</i>	error flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear DMA channel0 flag */
dma_flag_clear(DMA_CH0, DMA_FLAG_FTF);
```

## dma\_interrupt\_enable

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

**Table 3-145. Function dma\_interrupt\_enable**

<b>Function name</b>	dma_interrupt_enable
<b>Function prototype</b>	void dma_interrupt_enable(dma_channel_enum channelx, uint32_t source);
<b>Function descriptions</b>	enable DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
<i>DMA_CHx (x=0..6)</i>	DMA channel selection

Input parameter{in}	
<b>source</b>	DMA interrupt source
<i>DMA_INT_FTF</i>	full transfer finish interrupt of channel
<i>DMA_INT_HTF</i>	half transfer finish interrupt of channel
<i>DMA_INT_ERR</i>	error interrupt of channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMA channel0 interrupt */
```

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

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

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

**Table 3-146. Function dma\_interrupt\_disable**

<b>Function name</b>	dma_interrupt_disable
<b>Function prototype</b>	void dma_interrupt_disable(dma_channel_enum channelx, uint32_t source);
<b>Function descriptions</b>	disable DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
<i>DMA_CHx (x=0..6)</i>	DMA channel selection
Input parameter{in}	
<b>source</b>	DMA interrupt source
<i>DMA_INT_FTF</i>	full transfer finish interrupt of channel
<i>DMA_INT_HTF</i>	half transfer finish interrupt of channel
<i>DMA_INT_ERR</i>	error interrupt of channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMA channel0 interrupt */
```

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

## dma\_interrupt\_flag\_get

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

**Table 3-147. Function dma\_interrupt\_flag\_get**

<b>Function name</b>	dma_interrupt_flag_get
<b>Function prototype</b>	FlagStatus dma_interrupt_flag_get(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>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
<i>DMA_CHx (x=0..6)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>DMA_INT_FLAG_FTF</i>	full transfer finish interrupt flag of channel
<i>DMA_INT_FLAG_HTF</i>	half transfer finish interrupt flag of channel
<i>DMA_INT_FLAG_ERR</i>	error interrupt flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get DMA interrupt_flag */
if(dma_interrupt_flag_get(DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA_CH3, DMA_INT_FLAG_G);
    dma_interrupt_flag_clear(DMA_CH3, DMA_INT_FLAG_FTF);
}
```

## dma\_interrupt\_flag\_clear

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

**Table 3-148. Function dma\_interrupt\_flag\_clear**

<b>Function name</b>	dma_interrupt_flag_clear
<b>Function prototype</b>	void dma_interrupt_flag_clear(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>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-120. Enum dma_channel_enum</a> .
<i>DMA_CHx (x=0..6)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>DMA_INT_FLAG_G</i>	global interrupt flag of channel
<i>DMA_INT_FLAG_FTF</i>	full transfer finish interrupt flag of channel
<i>DMA_INT_FLAG_HTF</i>	half transfer finish interrupt flag of channel
<i>DMA_INT_FLAG_ERR</i>	error interrupt flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* get DMA interrupt_flag */

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

## 3.9. EXTI

EXTI is the interrupt/event controller in the MCU. It contains up to 23 independent edge detectors and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter [3.9.1](#), the EXTI firmware functions are introduced in chapter [3.9.2](#).

### 3.9.1. Descriptions of Peripheral registers

EXTI registers are listed in the table shown as below:

**Table 3-149. 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

### 3.9.2. Descriptions of Peripheral functions

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

**Table 3-150. EXTI firmware function**

Function name	Function description
exti_deinit	deinitialize EXTI
exti_init	initialize the EXTI
exti_interrupt_enable	enable the interrupts from EXTI line x
exti_interrupt_disable	disable the interrupts from EXTI line x
exti_event_enable	enable the events from EXTI line x
exti_event_disable	disable the events from EXTI line x
exti_software_interrupt_enable	enable the software interrupt event from EXTI line x
exti_software_interrupt_disable	disable the software interrupt event from EXTI line x
exti_flag_get	get EXTI line x interrupt pending flag
exti_flag_clear	clear EXTI line x interrupt pending flag
exti_interrupt_flag_get	get EXTI line x interrupt pending flag
exti_interrupt_flag_clear	clear EXTI line x interrupt pending flag

#### exti\_line\_enum

**Table 3-151. 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
EXTI_19	EXTI line 19

Member name	Function description
EXTI_20	EXTI line 20
EXTI_21	EXTI line 21
EXTI_22	EXTI line 22
EXTI_23	EXTI line 23
EXTI_24	EXTI line 24
EXTI_25	EXTI line 25
EXTI_26	EXTI line 26
EXTI_27	EXTI line 27

### exti\_mode\_enum

Table 3-152. Enum exti\_mode\_enum

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

### exti\_trig\_type\_enum

Table 3-153. Enum exti\_trig\_type\_enum

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

### exti\_deinit

The description of exti\_deinit is shown as below:

Table 3-154. 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 3-155. Function exti\_init**

<b>Function name</b>	exti_init
<b>Function prototype</b>	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
<b>Function descriptions</b>	initialize the EXTI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-151. Enum exti_line_enum</a>
<b>Input parameter{in}</b>	
<b>mode</b>	EXTI mode, refer to <a href="#">Table 3-152. Enum exti_mode_enum</a>
<b>Input parameter{in}</b>	
<b>trig_type</b>	trigger type, refer to <a href="#">Table 3-153. Enum exti_trig_type_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-156. 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 3-151. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* enable the interrupts from EXTI line 0 */
```

```
exti_interrupt_enable(EXTI_0);
```

### exti\_interrupt\_disable

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

**Table 3-157. Function exti\_interrupt\_disable**

Function name	exti_interrupt_disable
Function prototype	void exti_interrupt_disable(exti_line_enum linex);
Function descriptions	disable the interrupt from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-151. Enum exti_line_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the interrupts from EXTI line 0 */
```

```
exti_interrupt_disable(EXTI_0);
```

### exti\_event\_enable

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

**Table 3-158. Function exti\_event\_enable**

Function name	exti_event_enable
Function prototype	void exti_event_enable(exti_line_enum linex);
Function descriptions	enable the events from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-151. Enum exti_line_enum</a>
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable the events from EXTI line 0 */
```

```
exti_event_enable(EXTI_0);
```

### exti\_event\_disable

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

**Table 3-159. 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 3-151. 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 3-160. Function exti\_software\_interrupt\_enable**

<b>Function name</b>	exti_software_interrupt_enable
<b>Function prototype</b>	void exti_software_interrupt_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable the software interrupt event 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 3-151. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable EXTI line 0 software interrupt */
```

```
exti_software_interrupt_enable(EXTI_0);
```

### exti\_software\_interrupt\_disable

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

**Table 3-161. Function exti\_software\_interrupt\_disable**

<b>Function name</b>	exti_software_interrupt_disable
<b>Function prototype</b>	void exti_software_interrupt_disable(exti_line_enum linex);
<b>Function descriptions</b>	disable the software interrupt event 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 3-151. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable EXTI line 0 software interrupt */
```

```
exti_software_interrupt_disable(EXTI_0);
```

### exti\_flag\_get

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

**Table 3-162. Function exti\_flag\_get**

<b>Function name</b>	exti_flag_get
<b>Function prototype</b>	FlagStatus exti_flag_get(exti_line_enum linex);
<b>Function descriptions</b>	get EXTI line x 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 3-151. 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 3-163. 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 3-151. 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 3-164. 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 3-151. 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 3-165. Function exti\_interrupt\_flag\_clear**

Function name	exti_interrupt_flag_clear
Function prototype	void exti_interrupt_flag_clear(exti_line_enum linex);
Function descriptions	clear EXTI line x interrupt pending flag
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-151. Enum exti_line_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

## 3.10. FMC

There is flash controller and option byte for GD32F1x0 series. The FMC registers are listed in chapter [3.10.1](#) the FMC firmware functions are introduced in chapter [0](#).

### 3.10.1. Descriptions of Peripheral registers

FMC registers are listed in the table shown as below:

**Table 3-166. 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_WSEN	FMC wait state enable register

Registers	Descriptions
FMC_PID	FMC product ID register

### 3.10.2. Descriptions of Peripheral functions

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

**Table 3-167. FMC firmware function**

Function name	Function description
fmc_unlock	unlock the main FMC operation
fmc_lock	lock the main FMC operation
fmc_wsnt_set	set the wait state counter value
fmc_wait_state_enable	fmc wait state enable
fmc_wait_state_disable	fmc wait state disable
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_reset	reload the option byte and generate a system reset
ob_erase	erase the option byte
ob_write_protection_enable	enable option byte write protection (OB_WP)
ob_security_protection_config	configure read out protect
ob_user_write	write the FMC option byte user
ob_data_program	write the FMC option byte data
ob_user_get	get the FMC option byte OB_USER
ob_data_get	get the FMC option byte OB_DATA
ob_write_protection_get	get the FMC option byte write protection (OB_WP) in register FMC_WP
ob_obstat_plevel_get	get the state of FMC option byte security protection level
fmc_flag_get	get flag set or reset
fmc_flag_clear	clear the FMC pending flag
fmc_interrupt_enable	enable FMC interrupt
fmc_interrupt_disable	disable FMC interrupt
fmc_interrupt_flag_get	get interrupt flag set or reset
fmc_interrupt_flag_clear	clear FMC interrupt flag state

### Enum fmc\_state\_enum

**Table 3-168. Enum fmc\_state\_enum**

Member name	Function description
FMC_READY	the operation has been completed

Member name	Function description
FMC_BUSY	the operation is in progress
FMC_PGERR	program error
FMC_WPERR	erase/program protection error
FMC_TOERR	timeout error
FMC_OB_HSPC	option byte security protection code high

### fmc\_unlock

The description of fmc\_unlock is shown as below:

**Table 3-169. Function fmc\_unlock**

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

Example:

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

```
fmc_unlock( );
```

### fmc\_lock

The description of fmc\_lock is shown as below:

**Table 3-170. Function fmc\_lock**

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

-	-
---	---

Example:

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

```
fmc_lock( );
```

### fmc\_wscnt\_set

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

**Table 3-171. 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
WS_WSCNT_1	FMC 1 wait
WS_WSCNT_2	FMC 2 wait
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the wait state counter value */
```

```
fmc_wscnt_set(WS_WSCNT_1);
```

### fmc\_wait\_state\_enable

The description of fmc\_wait\_state\_enable is shown as below:

**Table 3-172. Function fmc\_wscnt\_set**

<b>Function name</b>	fmc_wait_state_enable
<b>Function prototype</b>	void fmc_wait_state_enable(void);
<b>Function descriptions</b>	fmc wait state enable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* fmc wait state enable */
```

```
fmc_wait_state_enable( );
```

### fmc\_wait\_state\_disable

The description of fmc\_wait\_state\_disable is shown as below:

**Table 3-173. Function fmc\_wscnt\_set**

<b>Function name</b>	fmc_wait_state_disable
<b>Function prototype</b>	void fmc_wait_state_disable(void);
<b>Function descriptions</b>	fmc wait state disable
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* fmc wait state disable */
```

```
fmc_wait_state_disable( );
```

### fmc\_page\_erase

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

**Table 3-174. Function fmc\_page\_erase**

<b>Function name</b>	fmc_page_erase
<b>Function prototype</b>	fmc_state_enum fmc_page_erase(uint32_t page_address);
<b>Function descriptions</b>	erase page
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
Input parameter{in}	
<b>page_address</b>	the page address to be erased
Output parameter{out}	
-	-

Return value	
<b>fmc_state_enum</b>	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-168. Enum fmc_state_enum</a>
<i>FMC_READY</i>	the operation has been completed
<i>FMC_WPERR</i>	erase/program protection error
<i>FMC_TOERR</i>	timeout error

Example:

```
/* erase page */

fmc_unlock( );

fmc_state_enum state = fmc_page_erase(0x08004000);

fmc_lock( );
```

### fmc\_mass\_erase

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

**Table 3-175. Function fmc\_mass\_erase**

<b>Function name</b>	fmc_mass_erase
<b>Function prototype</b>	fmc_state_enum fmc_mass_erase(void );
<b>Function descriptions</b>	erase whole chip
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-168. Enum fmc_state_enum</a>
<i>FMC_READY</i>	the operation has been completed
<i>FMC_WPERR</i>	erase/program protection error
<i>FMC_TOERR</i>	timeout error

Example:

```
/* erase whole chip */

fmc_unlock( );

fmc_state_enum state = fmc_mass_erase( );

fmc_lock( );
```

## fmc\_word\_program

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

**Table 3-176. Function fmc\_word\_program**

<b>Function name</b>	fmc_word_program
<b>Function prototype</b>	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
<b>Function descriptions</b>	program a word at the corresponding address
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>address</b>	the address to program
<b>data</b>	the data to program
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-168. Enum fmc_state_enum</a>
<i>FMC_READY</i>	the operation has been completed
<i>FMC_PGERR</i>	program error
<i>FMC_WPERR</i>	erase/program protection error
<i>FMC_TOERR</i>	timeout error

Example:

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

fmc_unlock( );

fmc_state_enum fmc_state = fmc_word_program(0x08004000,0xaabbccdd);

fmc_lock( );
```

## fmc\_halfword\_program

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

**Table 3-177. Function fmc\_halfword\_program**

<b>Function name</b>	fmc_halfword_program
<b>Function prototype</b>	fmc_state_enum fmc_halfword_program(uint32_t address, uint16_t data);
<b>Function descriptions</b>	program a half word at the corresponding address
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>address</b>	the address to program
<b>data</b>	the data to program

Output parameter{out}	
-	-
Return value	
<b>fmc_state_enum</b>	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-168. Enum fmc_state_enum</a>
<i>FMC_READY</i>	the operation has been completed
<i>FMC_PGERR</i>	program error
<i>FMC_WPERR</i>	erase/program protection error
<i>FMC_TOERR</i>	timeout error

Example:

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

fmc_unlock( );

fmc_state_enum fmc_state = fmc_halfword_program(0x08004000,0xaadd);

fmc_lock( );
```

### fmc\_word\_reprogram

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

**Table 3-178. Function fmc\_word\_reprogram**

<b>Function name</b>	fmc_word_reprogram
<b>Function prototype</b>	fmc_state_enum fmc_word_reprogram(uint32_t address, uint32_t data);
<b>Function descriptions</b>	program a word at the corresponding address without erasing
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
Input parameter{in}	
<b>address</b>	the address to program
<b>data</b>	the data to program
Output parameter{out}	
-	-
Return value	
<b>fmc_state_enum</b>	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-168. Enum fmc_state_enum</a>
<i>FMC_READY</i>	the operation has been completed
<i>FMC_WPERR</i>	erase/program protection error
<i>FMC_TOERR</i>	timeout error

Example:

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

fmc_state_enum fmc_state;
```



```
fmc_unlock( );
```

```
fmc_state = fmc_word_program(0x08004000, 0x01234567);
```

```
ffmc_state = fmc_word_reprogram(0x08004000, 0xd583179b);
```

```
fmc_lock( );
```

## ob\_unlock

The description of ob\_unlock is shown as below:

**Table 3-179. 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 */
```

```
ob_unlock( );
```

## ob\_lock

The description of ob\_lock is shown as below:

**Table 3-180. 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 */
```

```
ob_lock( );
```

### ob\_reset

The description of ob\_reset is shown as below:

**Table 3-181. Function ob\_reset**

<b>Function name</b>	ob_reset
<b>Function prototype</b>	void ob_reset(void);
<b>Function descriptions</b>	reload the option byte and generate a system reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reload the option byte and generate a system reset */
```

```
ob_reset( );
```

### ob\_erase

The description of ob\_erase is shown as below:

**Table 3-182. Function ob\_erase**

<b>Function name</b>	ob_erase
<b>Function prototype</b>	void ob_erase(void);
<b>Function descriptions</b>	erase the 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>	
<b>fmc_state_enum</b>	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-168. Enum fmc_state_enum</a>
<i>FMC_READY</i>	the operation has been completed

<i>FMC_PGERR</i>	program error
<i>FMC_TOERR</i>	timeout error
<i>FMC_OB_HSPC</i>	option byte security protection code high

Example:

```
/* erase the option byte */

fmc_unlock( );

ob_unlock( );

fmc_state_enum fmc_state = ob_erase( );

ob_lock( );

fmc_lock( );
```

### ob\_write\_protection\_enable

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

**Table 3-183. Function ob\_write\_protection\_enable**

<b>Function name</b>	ob_write_protection_enable
<b>Function prototype</b>	fmc_state_enum ob_write_protection_enable(uint16_t ob_wp);
<b>Function descriptions</b>	enable option byte write protection (OB_WP)
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>ob_wp</b>	write protection configuration data
<i>OB_WP_NONE</i>	disable all write protection
<i>OB_WP_x</i>	write protect specify sector (x=0..15)
<i>OB_WP_ALL</i>	write protect all sector
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-168. Enum fmc_state_enum</a>
<i>FMC_READY</i>	the operation has been completed
<i>FMC_PGERR</i>	program error
<i>FMC_TOERR</i>	timeout error
<i>FMC_OB_HSPC</i>	option byte security protection code high

Example:

```
/* enable write protection */

fmc_unlock( );
```

```
ob_unlock( );
```

```
fmc_state_enum state = ob_write_protection_enable(OB_WP_6 | OB_WP_7);
```

```
ob_lock( );
```

```
fmc_lock( );
```

### ob\_security\_protection\_config

The description of ob\_security\_protection\_config is shown as below:

**Table 3-184. Function ob\_security\_protection\_config**

<b>Function name</b>	ob_security_protection_config
<b>Function prototype</b>	fmc_state_enum ob_security_protection_config(uint8_t ob_spc);
<b>Function descriptions</b>	configure security protection
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>ob_spc</b>	specify security protection
<i>FMC_NSPC</i>	no security protection
<i>FMC_LSPC</i>	low security protection
<i>FMC_HSPC</i>	high security protection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-168. Enum fmc_state_enum</a>
<i>FMC_READY</i>	the operation has been completed
<i>FMC_PGERR</i>	program error
<i>FMC_TOERR</i>	timeout error
<i>FMC_OB_HSPC</i>	option byte security protection code high

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 3-185. Function ob\_user\_write**

<b>Function name</b>	ob_user_write
<b>Function prototype</b>	fmc_state_enum ob_user_write(uint8_t ob_user);
<b>Function descriptions</b>	program the FMC user option byte
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>ob_user</b>	user option byte
OB_FWDGT_HW	hardware free watchdog timer
OB_DEEPSLEEP_RST	generate a reset when entering deepsleep mode
OB_STDBY_RST	generate a reset when entering standby mode
OB_BOOT1_SET_1	BOOT1 bit is 1
OB_VDDA_DISABLE	disable VDDA monitor
OB_SRAM_PARITY_ENABLE	enable SRAM parity check
OB_USER_RSET	reset option byte user byte
<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 3-168. Enum fmc_state_enum</a>
FMC_READY	the operation has been completed
FMC_PGERR	program error
FMC_TOERR	timeout error
FMC_OB_HSPC	option byte security protection code high

Example:

```

/* program the FMC user option byte */

fmc_unlock( );

ob_unlock( );

fmc_state_enum fmc_state = ob_user_write(OB_DEEPSLEEP_RST & OB_STDBY_RST);

ob_lock( );

fmc_lock( );

```

## ob\_data\_program

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

Table 3-186. Function ob\_data\_program

Function name	ob_data_program
Function prototype	fmc_state_enum ob_data_program(uint16_t ob_data);
Function descriptions	program the FMC data option byte
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
data	the byte to be programmed
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-168. Enum fmc_state_enum</a>
FMC_READY	the operation has been completed
FMC_PGERR	program error
FMC_TOERR	timeout error
FMC_OB_HSPC	option byte security protection code high

Example:

```

/* program option bytes data */

fmc_unlock( );

ob_unlock( );

fmc_state_enum fmc_state = ob_data_program(0x5678);

ob_lock( );

fmc_lock( );

```

### ob\_user\_get

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

Table 3-187. Function ob\_user\_get

Function name	ob_user_get
Function prototype	uint8_t ob_user_get(void);
Function descriptions	get OB_USER in register FMC_OBSTAT
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

<b>uint8_t</b>	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 3-188. 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(0x0000 – 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 3-189. Function ob\_write\_protection\_get**

<b>Function name</b>	ob_write_protection_get
<b>Function prototype</b>	uint16_t ob_write_protection_get(void);
<b>Function descriptions</b>	get the FMC option byte write protection (OB_WP) in register FMC_WP
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	the FMC write protection option byte value(0x0000 – 0xFFFF)

Example:

```
/* get the FMC option byte write protection */
```

```
uint16_t wp = ob_write_protection_get( );
```

### ob\_obstat\_plevel\_get

The description of ob\_obstat\_plevel\_get is shown as below:

**Table 3-190. Function ob\_obstat\_plevel\_get**

<b>Function name</b>	ob_obstat_plevel_get
<b>Function prototype</b>	uint32_t ob_obstat_plevel_get(void);
<b>Function descriptions</b>	get the state of FMC option byte security protection level
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	the value of PLEVEL

Example:

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

```
uint32_t obstat_plevel = ob_obstat_plevel_get( );
```

### fmc\_flag\_get

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

**Table 3-191. Function fmc\_flag\_get**

<b>Function name</b>	fmc_flag_get
<b>Function prototype</b>	FlagStatus fmc_flag_get(uint32_t flag);
<b>Function descriptions</b>	get FMC flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
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
<b>Output parameter{out}</b>	
-	-



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 3-192. Function fmc\_flag\_clear**

Function name	fmc_flag_clear
Function prototype	void fmc_flag_clear(uint32_t flag);
Function descriptions	clear the FMC pending flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	clear FMC flag
FMC_FLAG_PGERR	FMC operation error flag
FMC_FLAG_WPERR	FMC erase/program protection error flag
FMC_FLAG_END	FMC end of operation flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
fmc_flag_clear(FMC_FLAG_END);
```

### fmc\_interrupt\_enable

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

**Table 3-193. Function fmc\_interrupt\_enable**

Function name	fmc_interrupt_enable
Function prototype	void fmc_interrupt_enable(uint32_t interrupt);
Function descriptions	enable FMC interrupt
Precondition	fmc_unlock
The called functions	-
Input parameter{in}	
interrupt	the FMC interrupt source

<i>FMC_INT_END</i>	FMC end of program interrupt
<i>FMC_INT_ERR</i>	FMC error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* enable FMC interrupt */

fmc_unlock( );

fmc_interrupt_enable(FMC_INT_END);

fmc_lock( );

```

### fmc\_interrupt\_disable

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

**Table 3-194. Function fmc\_interrupt\_disable**

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

Example:

```

/* disable FMC interrupt */

fmc_unlock( );

fmc_interrupt_disable(FMC_INT_END);

fmc_lock( );

```

### fmc\_interrupt\_flag\_get

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

Table 3-195. Function `fmc_interrupt_flag_get`

<b>Function name</b>	<code>fmc_interrupt_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus fmc_interrupt_flag_get(uint32_t int_flag);</code>
<b>Function descriptions</b>	get FMC interrupt flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	FMC flag
<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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

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

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

### **`fmc_interrupt_flag_clear`**

The description of `fmc_interrupt_flag_clear` is shown as below:

Table 3-196. Function `fmc_interrupt_flag_clear`

<b>Function name</b>	<code>fmc_interrupt_flag_clear</code>
<b>Function prototype</b>	<code>void fmc_interrupt_flag_clear(uint32_t int_flag);</code>
<b>Function descriptions</b>	clear the FMC pending interrupt flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	clear FMC flag
<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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear FMC interrupt flag */

fmc_interrupt_flag_clear(FMC_FLAG_PGERR);
```

## 3.11. 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 [3.11.1](#) the FWDGT firmware functions are introduced in chapter [3.11.2](#).

### 3.11.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

**Table 3-197. FWDGT Registers**

Registers	Descriptions
FWDGT_CTL	Control register
FWDGT_PSC	Prescaler register
FWDGT_RLD	Reload register
FWDGT_STAT	Status register
FWDGT_WND	Window register

### 3.11.2. Descriptions of Peripheral functions

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

**Table 3-198. 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_prescaler_value_config	configure the free watchdog timer counter prescaler value
fwdgt_reload_value_config	configure the free watchdog timer counter reload value
fwdgt_window_value_config	configure the FWDGT counter window value
fwdgt_counter_reload	reload the counter of FWDGT
fwdgt_config	configure counter reload value, and prescaler divider value
fwdgt_flag_get	get flag state of FWDGT

#### fwdgt\_write\_enable

The description of fwdgt\_write\_enable is shown as below:

Table 3-199. 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 3-200. 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 3-201. Function fwdgt\_enable

Function name	fwdgt_enable
Function prototype	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\_prescaler\_value\_config**

The description of fwdgt\_prescaler\_value\_config is shown as below:

**Table 3-202. Function fwdgt\_prescaler\_value\_config**

<b>Function name</b>	fwdgt_prescaler_value_config
<b>Function prototype</b>	ErrStatus fwdgt_prescaler_value_config(uint16_t prescaler_value);
<b>Function descriptions</b>	configure the free watchdog timer counter prescaler value
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>prescaler_value</b>	specify prescaler value
<i>FWDGT_PSC_DIV4</i>	FWDGT prescaler set to 4
<i>FWDGT_PSC_DIV8</i>	FWDGT prescaler set to 8
<i>FWDGT_PSC_DIV16</i>	FWDGT prescaler set to 16
<i>FWDGT_PSC_DIV32</i>	FWDGT prescaler set to 32
<i>FWDGT_PSC_DIV64</i>	FWDGT prescaler set to 64
<i>FWDGT_PSC_DIV128</i>	FWDGT prescaler set to 128
<i>FWDGT_PSC_DIV256</i>	FWDGT prescaler set to 256
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR / SUCCESS

Example:

```
/* set FWDGT prescaler to 256 */
```

```
ErrStatus flag;
```

```
flag = fwdgt_prescaler_value_config (FWDGT_PSC_DIV256);
```

## fwdgt\_reload\_value\_config

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

**Table 3-203. Function fwdgt\_reload\_value\_config**

<b>Function name</b>	fwdgt_reload_value_config
<b>Function prototype</b>	ErrStatus fwdgt_reload_value_config(uint16_t reload_value);
<b>Function descriptions</b>	configure the free watchdog timer counter reload value
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>reload_value</b>	reload_value: specify window value(0x0000 - 0x0FFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR / SUCCESS

Example:

```
/* set FWDGT reload value to 0x0FFF */
```

```
ErrStatus flag;
```

```
flag = fwdgt_reload_value_config (0x0FFF);
```

## fwdgt\_window\_value\_config

The description of fwdgt\_window\_value\_config is shown as below:

**Table 3-204. Function fwdgt\_window\_value\_config**

<b>Function name</b>	fwdgt_window_value_config
<b>Function prototype</b>	ErrStatus fwdgt_window_value_config(uint16_t window_value);
<b>Function descriptions</b>	configure the FWDGT counter window value
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>window_value</b>	window_value: specify window value(0x0000 - 0x0FFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR / SUCCESS

Example:

```
/* set FWDGT window value to 0x0FFF */
```

```
ErrStatus flag;
```

```
flag = fwdgt_window_value_config (0x0FFF);
```

## fwdgt\_counter\_reload

The description of fwdgt\_counter\_reload is shown as below:

**Table 3-205. 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 3-206. 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:

```
/* configure FWDGT counter clock: 40KHz(IRC40K) / 64 = 0.625 KHz */
```

```
fwdgt_config(2*500, FWDGT_PSC_DIV64);
```

### fwdgt\_flag\_get

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

**Table 3-207. 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
<i>FWDGT_FLAG_WUD</i>	a write operation to FWDGT_WND register is on going
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* test if a prescaler value update is on going */
```

```
FlagStatus status;
```

```
status = fwdgt_flag_get (FWDGT_FLAG_PUD);
```

## 3.12. GPIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter [3.12.1](#), the GPIO firmware functions are introduced in chapter [3.12.2](#).

### 3.12.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

**Table 3-208. GPIO Registers**

Registers	Descriptions
GPIOx_CTL	GPIO port control register

Registers	Descriptions
GPIOx_OMODE	GPIO port output mode register
GPIOx_OSPD	GPIO port output speed register
GPIOx_PUD	GPIO port pull-up/pull-down register
GPIOx_ISTAT	GPIO port input status register
GPIOx_OCTL	GPIO port output control register
GPIOx_BOP	GPIO port bit operation register
GPIOx_LOCK	GPIO port configuration lock register
GPIO_AFSEL0	GPIO alternate function selected register 0
GPIO_AFSEL1	GPIO alternate function selected register 1
GPIO_BC	GPIO bit clear register

### 3.12.2. Descriptions of Peripheral functions

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

**Table 3-209. GPIO firmware function**

Function name	Function description
gpio_deinit	reset GPIO port
gpio_mode_set	set GPIO mode
gpio_output_options_set	set GPIO output type and speed
gpio_bit_set	set GPIO pin bit
gpio_bit_reset	reset GPIO pin bit
gpio_bit_write	write data to the specified GPIO pin
gpio_port_write	write data to the specified GPIO port
gpio_input_bit_get	get GPIO pin input status
gpio_input_port_get	get GPIO port input status
gpio_output_bit_get	get GPIO pin output status
gpio_output_port_get	get GPIO port output status
gpio_af_set	set GPIO alternate function
gpio_pin_lock	lock GPIO pin

#### gpio\_deinit

The description of gpio\_deinit is shown as below:

**Table 3-210. Function gpio\_deinit**

Function name	gpio_deinit
Function prototype	void gpio_deinit(uint32_t gpio_periph);
Function descriptions	reset GPIO port
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	

<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,F)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset GPIOA */
```

```
gpio_deinit (GPIOA);
```

### gpio\_mode\_set

The description of gpio\_mode\_set is shown as below:

**Table 3-211. Function gpio\_mode\_set**

<b>Function name</b>	gpio_mode_set
<b>Function prototype</b>	void gpio_mode_set(uint32_t gpio_periph, uint32_t mode, uint32_t pull_up_down, uint32_t pin);
<b>Function descriptions</b>	set GPIO mode
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,F)
<b>Input parameter{in}</b>	
<b>mode</b>	gpio pin mode
<i>GPIO_MODE_INPUT</i>	input mode
<i>GPIO_MODE_OUTPUT</i>	output mode
<i>GPIO_MODE_AF</i>	alternate function mode
<i>GPIO_MODE_ANALOG</i>	analog mode
<b>Input parameter{in}</b>	
<b>pull_up_down</b>	gpio pin with pull-up or pull-down resistor
<i>GPIO_PUPD_NONE</i>	floating mode, no pull-up and pull-down resistors
<i>GPIO_PUPD_PULLUP</i>	with pull-up resistor
<i>GPIO_PUPD_PULLDOWN</i>	with pull-down resistor
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config PA0 as input mode with pullup*/
```

```
gpio_mode_set (GPIOA, GPIO_MODE_INPUT, GPIO_PUPD_PULLUP, GPIO_PIN_0);
```

### gpio\_output\_options\_set

The description of gpio\_output\_options\_set is shown as below:

**Table 3-212. Function gpio\_output\_options\_set**

Function name	gpio_output_options_set
Function prototype	void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t pin);
Function descriptions	set GPIO output type and speed
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C,D,F)
Input parameter{in}	
otype	gpio pin output mode
GPIO_OTYPE_PP	push pull mode
GPIO_OTYPE_OD	open drain mode
Input parameter{in}	
speed	gpio pin output max speed
GPIO_OSPEED_2MHZ	output max speed 2MHz
GPIO_OSPEED_10MHZ	output max speed 10MHz
GPIO_OSPEED_50MHZ	output max speed 50MHz
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config PA0 as push pull mode */
```

```
gpio_output_options_set(GPIOA, GPIO_OTYPE_PP, GPIO_OSPEED_2MHZ,
GPIO_PIN_0);
```

## gpio\_bit\_set

The description of gpio\_bit\_set is shown as below:

**Table 3-213. Function gpio\_bit\_set**

<b>Function name</b>	gpio_bit_set
<b>Function prototype</b>	void gpio_bit_set(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	set GPIO pin bit
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,F)
<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 3-214. Function gpio\_bit\_reset**

<b>Function name</b>	gpio_bit_reset
<b>Function prototype</b>	void gpio_bit_reset(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	reset GPIO pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port

<i>GPIOx</i>	GPIOx(x = A,B,C,D,F)
<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 3-215. 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,F)
<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 3-216. 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,F)
<b>Input parameter{in}</b>	
<b>data</b>	specify the value to be written to the port output data register
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*write 1010 0101 1010 0101 to Port A */
```

```
gpio_port_write (GPIOA, 0xA5A5);
```

## gpio\_input\_bit\_get

The description of gpio\_input\_bit\_get is shown as below:

**Table 3-217. 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,F)
<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

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

Example:

```
/* get status of PA0 */
```

```
FlagStatus bit_state = gpio_input_bit_get (GPIOA, GPIO_PIN_0);
```

### gpio\_input\_port\_get

The description of gpio\_input\_port\_get is shown as below:

**Table 3-218. Function gpio\_input\_port\_get**

Function name	gpio_input_port_get
Function prototype	uint16_t gpio_input_port_get(uint32_t gpio_periph);
Function descriptions	get GPIO all pins input status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C,D,F)
Output parameter{out}	
-	-
Return value	
uint16_t	0x0000-0xFFFF

Example:

```
/* get input value of Port A */
```

```
uint16_t port_state;
```

```
port_state = gpio_input_bit_get (GPIOA);
```

### gpio\_output\_bit\_get

The description of gpio\_output\_bit\_get is shown as below:

**Table 3-219. Function gpio\_output\_bit\_get**

Function name	gpio_output_bit_get
Function prototype	FlagStatus gpio_output_bit_get(uint32_t gpio_periph,uint32_t pin);
Function descriptions	get GPIO pin output status
Precondition	-
The called functions	-
Input parameter{in}	



<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,F)
<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 3-220. Function gpio\_output\_port\_get**

<b>Function name</b>	gpio_output_port_get
<b>Function prototype</b>	uint16_t gpio_output_port_get(uint32_t gpio_periph);
<b>Function descriptions</b>	get GPIO all pins output status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,F)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>Uint16_t</b>	0x0000-0xFFFF

Example:

```
/* get output value of Port A */
```

```
uint16_t port_state;
```

```
port_state = gpio_output_port_get (GPIOA);
```

### gpio\_af\_set

The description of gpio\_af\_set is shown as below:

Table 3-221. Function `gpio_af_set`

<b>Function name</b>	<code>gpio_af_set</code>
<b>Function prototype</b>	<code>void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);</code>
<b>Function descriptions</b>	set GPIO alternate function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,F)
<b>Input parameter{in}</b>	
<b>alt_func_num</b>	GPIO pin af function, please refer to specific device datasheet
<i>GPIO_AF_0</i>	TIMER2, TIMER13, TIMER14, TIMER16, SPI0, I2S0, SPI1, SPI2, I2S2, CK_OUT, SWDIO, SWCLK, USART0, CEC, IFRP, I2C0, I2C1, TSI, EVENTOUT
<i>GPIO_AF_1</i>	USART0, USART1, IFRP, CEC, TIMER2, TIMER14, I2C0, I2C1, I2C2, EVENTOUT
<i>GPIO_AF_2</i>	TIMER0, TIMER1, TIMER15, TIMER16, EVENTOUT
<i>GPIO_AF_3</i>	TSI, I2C0, TIMER14, EVENTOUT
<i>GPIO_AF_4 (port A,B only)</i>	TIMER13, I2C0, I2C1, I2C2, USART1
<i>GPIO_AF_5 (port A,B only)</i>	TIMER15, TIMER16, SPI2, I2S2, I2C0, I2C1
<i>GPIO_AF_6 (port A,B only)</i>	SPI1, EVENTOUT
<i>GPIO_AF_7 (port A only)</i>	CMP0, CMP1
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*set PA0 alternate function 0*/
```

```
gpio_af_set(GPIOA, GPIO_AF_0, GPIO_PIN_0);
```

### **gpio\_pin\_lock**

The description of `gpio_pin_lock` is shown as below:

Table 3-222. 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 bit
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock PA0*/
```

```
gpio_pin_lock (GPIOA, GPIO_PIN_ 0);
```

## 3.13. 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 [3.13.1](#), the I2C firmware functions are introduced in chapter [3.13.2](#)

### 3.13.1. Descriptions of Peripheral registers

I2C registers are listed in the table shown as below:

Table 3-223. 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

Registers	Descriptions
I2C_CKCFG	Clock configure register
I2C_RT	Rise time register

### 3.13.2. Descriptions of Peripheral functions

I2C firmware functions are listed in the table shown as below:

**Table 3-224. 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	select SMBus type
i2c_ack_config	whether or not to send an ACK
i2c_ackpos_config	configure I2C POAP position
i2c_master_addressing	master sends slave address
i2c_dualaddr_enable	enable dual-address mode
i2c_dualaddr_disable	disable dual-address mode
i2c_enable	enable I2C
i2c_disable	disable I2C
i2c_start_on_bus	generate a START condition on I2C bus
i2c_stop_on_bus	generate a STOP condition on I2C bus
i2c_data_transmit	I2C transmit data function
i2c_data_receive	I2C receive data function
i2c_dma_config	configure I2C DMA mode
i2c_dma_last_transfer_config	configure whether next DMA EOT is DMA last transfer or not
i2c_stretch_scl_low_config	whether to stretch SCL low when data is not ready in slave mode
i2c_slave_response_to_gcall_config	whether or not to response to a general call
i2c_software_reset_config	configure software reset of I2C
i2c_pec_config	configure I2C PEC calculation
i2c_pec_transfer_config	configure whether to transfer PEC value
i2c_pec_value_get	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 status
i2c_interrupt_flag_clear	clear I2C interrupt status

## Enum i2c\_flag\_enum

**Table 3-225. Enum i2c\_flag\_enum**

Member name	Function description
I2C_FLAG_SBSEND	start condition sent out in master mode
I2C_FLAG_ADDSEND	address is sent in master mode or received and matches in slave mode
I2C_FLAG_BTC	byte transmission finishes
I2C_FLAG_ADD10SEND	header of 10-bit address is sent in master mode
I2C_FLAG_STPDET	stop condition detected in slave mode
I2C_FLAG_RBNE	I2C_DATA is not empty during receiving
I2C_FLAG_TBE	I2C_DATA is empty during transmitting
I2C_FLAG_BERR	a bus error occurs indication a unexpected start or stop condition on I2C bus
I2C_FLAG_LOSTARB	arbitration lost in master mode
I2C_FLAG_AERR	acknowledge error
I2C_FLAG_OUERR	over-run or under-run situation occurs in slave mode
I2C_FLAG_PECERR	PEC error when receiving data
I2C_FLAG_SMBTO	timeout signal in SMBus mode
I2C_FLAG_SMBALT	SMBus alert status
I2C_FLAG_MASTER	a flag indicating whether I2C block is in master or slave mode
I2C_FLAG_I2CBSY	busy flag
I2C_FLAG_TR	whether the I2C is a transmitter or a receiver
I2C_FLAG_RXGC	general call address (00h) received
I2C_FLAG_DEFSMB	default address of SMBus device
I2C_FLAG_HSTSMB	SMBus host header detected in slave mode
I2C_FLAG_DUMOD	dual flag in slave mode indicating which address is matched in dual-address mode

## Enum i2c\_interrupt\_flag\_enum

**Table 3-226. Enum i2c\_interrupt\_flag\_enum**

Member name	Function description
I2C_INT_FLAG_SBSEND	start condition sent out in master mode interrupt flag
I2C_INT_FLAG_ADDSEND	address is sent in master mode or received and matches in slave mode interrupt flag
I2C_INT_FLAG_BTC	byte transmission finishes interrupt flag
I2C_INT_FLAG_ADD10SEND	header of 10-bit address is sent in master mode interrupt flag
I2C_INT_FLAG_STPDET	stop condition detected in slave mode interrupt flag
I2C_INT_FLAG_RBNE	I2C_DATA is not empty during receiving interrupt flag
I2C_INT_FLAG_TBE	I2C_DATA is empty during transmitting interrupt flag
I2C_INT_FLAG_BERR	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag

Member name	Function description
I2C_INT_FLAG_LOSTARB	arbitration lost in master mode interrupt flag
I2C_INT_FLAG_AERR	acknowledge error interrupt flag
I2C_INT_FLAG_OUERR	over-run or under-run situation occurs in slave mode interrupt flag
I2C_INT_FLAG_PECERR	PEC error when receiving data interrupt flag
I2C_INT_FLAG_SMBTO	timeout signal in SMBus mode interrupt flag
I2C_INT_FLAG_SMBALT	SMBus alert status interrupt flag

## Enum i2c\_interrupt\_enum

Table 3-227. Enum i2c\_interrupt\_enum

Member name	Function description
I2C_INT_ERR	error interrupt
I2C_INT_EV	event interrupt
I2C_INT_BUF	buffer interrupt

## i2c\_deinit

The description of i2c\_deinit is shown as below:

Table 3-228. Function i2c\_deinit

Function name	i2c_deinit
Function prototype	void i2c_deinit(uint32_t i2c_periph);
Function descriptions	reset I2C
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset I2C0 */
```

```
i2c_deinit (I2C0);
```

## i2c\_clock\_config

The description of i2c\_clock\_config is shown as below:

Table 3-229. Function i2c\_clock\_config

Function name	i2c_clock_config
---------------	------------------

<b>Function prototype</b>	void i2c_clock_config(uint32_t i2c_periph, uint32_t clkspeed, uint32_t dutycyc);
<b>Function descriptions</b>	configure I2C clock
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>clkspeed</b>	i2c clock speed
<b>Input parameter{in}</b>	
<b>dutycyc</b>	duty cycle in fast mode
<i>I2C_DTCY_2</i>	T_low/T_high=2
<i>I2C_DTCY_16_9</i>	T_low/T_high=16/9
<b>Output parameter{out}</b>	
-	-
<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 3-230. Function i2c\_mode\_addr\_config**

<b>Function name</b>	i2c_mode_addr_config
<b>Function prototype</b>	void i2c_mode_addr_config(uint32_t i2c_periph, uint32_t mode, uint32_t addformat, uint32_t addr);
<b>Function descriptions</b>	configure I2C address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<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

Input parameter{in}	
<b>addformat</b>	7bits or 10bits
<i>I2C_ADDFORMAT_7BITS</i>	address format is 7 bits
<i>I2C_ADDFORMAT_10BITS</i>	address format is 10 bits
Input parameter{in}	
<b>addr</b>	I2C address
Output parameter{out}	
-	-
Return value	
-	-

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 3-231. 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>	select SMBus type
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
<b>type</b>	Device or host
<i>I2C_SMBUS_DEVICE</i>	SMBus mode device type
<i>I2C_SMBUS_HOST</i>	SMBus mode host type
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config I2C0 as SMBUS host type*/
```

```
i2c_smbus_type_config (I2C0, I2C_SMBUS_HOST);
```



## i2c\_ack\_config

The description of i2c\_ack\_config is shown as below:

**Table 3-232. Function i2c\_ack\_config**

<b>Function name</b>	i2c_ack_config
<b>Function prototype</b>	void i2c_ack_config(uint32_t i2c_periph, uint32_t ack);
<b>Function descriptions</b>	whether or not to send an ACK
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>ack</b>	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 3-233. Function i2c\_ackpos\_config**

<b>Function name</b>	i2c_ackpos_config
<b>Function prototype</b>	void i2c_ackpos_config(uint32_t i2c_periph, uint32_t pos);
<b>Function descriptions</b>	configure I2C POAP position
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>pos</b>	ACK position
<i>I2C_ACKPOS_CURRENT</i>	ACKEN bit decides whether or not to send ACK or not for the current byte

<i>I2C_ACKPOS_NEXT</i>	ACKEN bit decides whether or not to send ACK 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 3-234. Function i2c\_master\_addressing**

<b>Function name</b>	i2c_master_addressing
<b>Function prototype</b>	void i2c_master_addressing(uint32_t i2c_periph, uint32_t addr, uint32_t trandirection);
<b>Function descriptions</b>	master sends slave address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>addr</b>	slave address
<b>Input parameter{in}</b>	
<b>trandirection</b>	transmitter or receiver
<i>I2C_TRANSMITTER</i>	transmitter
<i>I2C_RECEIVER</i>	receiver
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* send slave address to I2C bus and I2C0 act as receiver */
```

```
i2c_master_addressing(I2C0, 0x82, I2C_RECEIVER);
```

### i2c\_dualaddr\_enable

The description of i2c\_dualaddr\_enable is shown as below:

Table 3-235. 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>	enable dual-address mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>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 3-236. Function i2c\_dualaddr\_disable

<b>Function name</b>	i2c_dualaddr_disable
<b>Function prototype</b>	void i2c_dualaddr_disable(uint32_t i2c_periph)
<b>Function descriptions</b>	disable dual-address mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<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 3-237. Function i2c\_enable**

<b>Function name</b>	i2c_enable
<b>Function prototype</b>	void i2c_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 */
i2c_enable (I2C0);
```

## i2c\_disable

The description of i2c\_disable is shown as below:

**Table 3-238. Function i2c\_disable**

<b>Function name</b>	i2c_disable
<b>Function prototype</b>	void i2c_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C0 */
i2c_disable (I2C0);
```

## i2c\_start\_on\_bus

The description of i2c\_start\_on\_bus is shown as below:

**Table 3-239. Function i2c\_start\_on\_bus**

<b>Function name</b>	i2c_start_on_bus
<b>Function prototype</b>	void i2c_start_on_bus(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a START condition on I2C bus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 send a start condition to I2C bus */
```

```
i2c_start_on_bus (I2C0);
```

## i2c\_stop\_on\_bus

The description of i2c\_stop\_on\_bus is shown as below:

**Table 3-240. Function i2c\_stop\_on\_bus**

<b>Function name</b>	i2c_stop_on_bus
<b>Function prototype</b>	void i2c_stop_on_bus(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a STOP condition on I2C bus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 generate a STOP condition to I2C bus */
```

```
i2c_stop_on_bus (I2C0);
```

## i2c\_data\_transmit

The description of i2c\_data\_transmit is shown as below:

**Table 3-241. Function i2c\_data\_transmit**

<b>Function name</b>	i2c_data_transmit
<b>Function prototype</b>	void i2c_data_transmit(uint32_t i2c_periph, uint8_t data);
<b>Function descriptions</b>	I2C transmit data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>data</b>	transmit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 transmit data */
```

```
i2c_data_transmit (I2C0, 0x80);
```

## i2c\_data\_receive

The description of i2c\_data\_receive is shown as below:

**Table 3-242. Function i2c\_data\_receive**

<b>Function name</b>	i2c_data_receive
<b>Function prototype</b>	uint8_t i2c_data_receive(uint32_t i2c_periph);
<b>Function descriptions</b>	I2C receive data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	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 3-243. Function i2c\_dma\_config**

<b>Function name</b>	i2c_dma_config
<b>Function prototype</b>	void i2c_dma_config(uint32_t i2c_periph, uint32_t dmastate);
<b>Function descriptions</b>	configure I2C DMA mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>dmastate</b>	on or off
<i>I2C_DMA_ON</i>	enable DMA mode
<i>I2C_DMA_OFF</i>	disable DMA mode
<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 3-244. 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>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)

Input parameter{in}	
<b>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
Output parameter{out}	
-	-
Return value	
-	-

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 3-245. Function i2c\_stretch\_scl\_low\_config**

<b>Function name</b>	i2c_stretch_scl_low_config
<b>Function prototype</b>	void i2c_stretch_scl_low_config(uint32_t i2c_periph, uint32_t stretchpara);
<b>Function descriptions</b>	whether to stretch SCL low when data is not ready in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
<b>stretchpara</b>	SCL stretching enable or disable
<i>I2C_SCLSTRETCH_ENABLE</i>	enable SCL stretching
<i>I2C_SCLSTRETCH_DISABLE</i>	disable SCL stretching
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_config (I2C0, I2C_SCLSTRETCH_ENABLE);
```



## i2c\_slave\_response\_to\_gcall\_config

The description of i2c\_slave\_response\_to\_gcall\_config is shown as below:

**Table 3-246. Function i2c\_slave\_response\_to\_gcall\_config**

<b>Function name</b>	i2c_slave_response_to_gcall_config
<b>Function prototype</b>	void i2c_slave_response_to_gcall_config(uint32_t i2c_periph, uint32_t gcallpara);
<b>Function descriptions</b>	whether or not to response to a general call
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<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 3-247. Function i2c\_software\_reset\_config**

<b>Function name</b>	i2c_software_reset_config
<b>Function prototype</b>	void i2c_software_reset_config(uint32_t i2c_periph, uint32_t sreset);
<b>Function descriptions</b>	configure software reset of I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>sreset</b>	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 3-248. 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 or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>pecstate</b>	on or off
<i>I2C_PEC_ENABLE</i>	PEC calculation on
<i>I2C_PEC_DISABLE</i>	PEC calculation off
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* Enable I2C PEC calculation */
```

```
i2c_pec_config (I2C0, I2C_PEC_ENABLE);
```

### i2c\_pec\_transfer\_config

The description of i2c\_pec\_transfer\_enable is shown as below:

**Table 3-249. Function i2c\_pec\_transfer\_config**

<b>Function name</b>	i2c_pec_transfer_config
<b>Function prototype</b>	void i2c_pec_transfer_config(uint32_t i2c_periph, uint32_t pecpara);

<b>Function descriptions</b>	configure whether to transfer PEC value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>pecpara</b>	Transfer PEC or not
<i>I2C_PECTRANS_ENABLE</i>	transfer PEC
<i>I2C_PECTRANS_DISABLE</i>	not transfer PEC
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-250. Function i2c\_pec\_value\_get**

<b>Function name</b>	i2c_pec_value_get
<b>Function prototype</b>	uint8_t i2c_pec_value_get(uint32_t i2c_periph);
<b>Function descriptions</b>	get packet error checking value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	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 3-251. Function i2c\_smbus\_alert\_config**

<b>Function name</b>	i2c_smbus_alert_config
<b>Function prototype</b>	void i2c_smbus_alert_config (uint32_t i2c_periph, uint32_t smbuspara);
<b>Function descriptions</b>	configure I2C alert through SMBA pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>smbuspara</b>	issue alert through SMBA pin or not
<i>I2C_SALTSEND_ENABLE</i>	issue alert through SMBA pin
<i>I2C_SALTSEND_DISABLE</i>	not issue alert through SMBA pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 issue alert through SMBA pin enable*/
```

```
i2c_smbus_alert_config (I2C0, I2C_SALTSEND_ENABLE);
```

### i2c\_smbus\_arp\_config

The description of i2c\_smbus\_arp\_config is shown as below:

**Table 3-252. Function i2c\_smbus\_arp\_config**

<b>Function name</b>	i2c_smbus_arp_config
<b>Function prototype</b>	void i2c_smbus_arp_config(uint32_t i2c_periph, uint32_t arpstate);
<b>Function descriptions</b>	configure I2C ARP protocol in SMBus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<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 3-253. 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>	check I2C flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>I2C_FLAG_SBSEND</i>	start condition sent out in master mode
<i>I2C_FLAG_ADDSEND</i>	address is sent in master mode or received and matches in slave mode
<i>I2C_FLAG_BTCT</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>	over-run or under-run 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_SBSSEND);
```

### i2c\_flag\_clear

The description of i2c\_flag\_clear is shown as below:

**Table 3-254. Function i2c\_flag\_clear**

<b>Function name</b>	i2c_flag_clear
<b>Function prototype</b>	void i2c_flag_clear(uint32_t i2c_periph, i2c_flag_enum flag)
<b>Function descriptions</b>	clear I2C flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<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 occurs indication a unexpected start or stop condition on I2C bus

<i>I2C_FLAG_ADDSEND</i>	address is sent in master mode or received and matches in slave mode
<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 3-255. Function i2c\_interrupt\_enable**

<b>Function name</b>	i2c_interrupt_enable
<b>Function prototype</b>	void i2c_interrupt_enable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);
<b>Function descriptions</b>	enable I2C interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<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 3-256. Function i2c\_interrupt\_disable**

<b>Function name</b>	i2c_interrupt_disable
----------------------	-----------------------

<b>Function prototype</b>	void i2c_interrupt_disable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);
<b>Function descriptions</b>	disable I2C interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<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 3-257. Function i2c\_interrupt\_flag\_get**

<b>Function name</b>	i2c_interrupt_flag_get
<b>Function prototype</b>	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag)
<b>Function descriptions</b>	get I2C interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt flag
<i>I2C_INT_FLAG_SBSE</i> <i>ND</i>	start condition sent out in master mode interrupt flag
<i>I2C_INT_FLAG_ADDS</i> <i>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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* check the byte transmission finishes interrupt flag is set or not*/
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_interrupt_flag_get (I2C0, I2C_INT_FLAG_BTC);
```

### **i2c\_interrupt\_flag\_clear**

The description of i2c\_interrupt\_flag\_clear is shown as below:

**Table 3-258. 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>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral

<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt flag
<i>I2C_INT_FLAG_ADDS</i> <i>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</i> <i>RB</i>	arbitration lost in master mode interrupt flag
<i>I2C_INT_FLAG_AERR</i>	acknowledge error interrupt flag
<i>I2C_INT_FLAG_OUER</i> <i>R</i>	over-run or under-run situation occurs in slave mode interrupt flag
<i>I2C_INT_FLAG_PECER</i> <i>RR</i>	PEC error when receiving data interrupt flag
<i>I2C_INT_FLAG_SMBT</i> <i>O</i>	timeout signal in SMBus mode interrupt flag
<i>I2C_INT_FLAG_SMBA</i> <i>LT</i>	SMBus Alert status 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);
```

## 3.14. MISC

MISC is a software package that provide the interfaces for NVIC and SysTick. The NVIC and SysTick registers are listed in chapter [3.14.1](#), the MISC firmware functions are introduced in chapter [3.14.2](#).

### 3.14.1. Descriptions of Peripheral registers

**Table 3-259. NVIC Registers**

Registers	Descriptions
ISER <sup>(1)</sup>	Interrupt Set Enable Register
ICER <sup>(1)</sup>	Interrupt Clear Enable Register
ISPR <sup>(1)</sup>	Interrupt Set Pending Register
ICPR <sup>(1)</sup>	Interrupt Clear Pending Register

Registers	Descriptions
IABR <sup>(1)</sup>	Interrupt Active bit Register
IP <sup>(1)</sup>	Interrupt Priority Register
STIR <sup>(1)</sup>	Software Trigger Interrupt Register
CPUID <sup>(2)</sup>	CPUID Base Register
ICSR <sup>(2)</sup>	Interrupt Control and State Register
VTOR <sup>(2)</sup>	Vector Table Offset Register
AIRCR <sup>(2)</sup>	Application Interrupt and Reset Control Register
SCR <sup>(2)</sup>	System Control Register
CCR <sup>(2)</sup>	Configuration Control Register
SHP <sup>(2)</sup>	System Handlers Priority Registers
SHCSR <sup>(2)</sup>	System Handler Control and State Register
CFSR <sup>(2)</sup>	Configurable Fault Status Register
HFSR <sup>(2)</sup>	HardFault Status Register
DFSR <sup>(2)</sup>	Debug Fault Status Register
MMFAR <sup>(2)</sup>	MemManage Fault Address Register
BFAR <sup>(2)</sup>	BusFault Address Register
AFSR <sup>(2)</sup>	Auxiliary Fault Status Register
PFR <sup>(2)</sup>	Processor Feature Register
DFR <sup>(2)</sup>	Debug Feature Register
ADR <sup>(2)</sup>	Auxiliary Feature Register
MMFR <sup>(2)</sup>	Memory Model Feature Register
ISAR <sup>(2)</sup>	Instruction Set Attributes Register
CPACR <sup>(2)</sup>	Coprocessor Access Control Register

1. refer to the structure NVIC\_Type, is defined in the core\_cm3.h file

2. refer to the structure SCB\_Type, is defined in the core\_cm3.h file

**Table 3-260. SysTick Registers**

Registers	Descriptions
CTRL <sup>(1)</sup>	SysTick Control and Status Register
LOAD <sup>(1)</sup>	SysTick Reload Value Register
VAL <sup>(1)</sup>	SysTick Current Value Register
CALIB <sup>(1)</sup>	SysTick Calibration Register

1. refer to the structure SysTick\_Type, is defined in the core\_cm3.h file

### 3.14.2. Descriptions of Peripheral functions

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

**Table 3-261. MISC firmware function**

Function name	Function description
nvic_priority_group_set	set the priority group
nvic_irq_enable	enable NVIC interrupt request

Function name	Function description
<code>nvic_irq_disable</code>	disable NVIC interrupt request
<code>nvic_vector_table_set</code>	set the NVIC vector table address
<code>system_lowpower_set</code>	set the state of the low power mode
<code>system_lowpower_reset</code>	reset the state of the low power mode
<code>systick_clksource_set</code>	set the systick clock source

## Enum IRQn\_Type

**Table 3-262. Enum IRQn\_Type**

Member name	Function description
<code>WWDGT_IRQn</code>	WWDGT interrupt
<code>LVD_IRQn</code>	LVD from EXTI line interrupt
<code>RTC_IRQn</code>	RTC global interrupt
<code>FMC_IRQn</code>	FMC interrupt
<code>RCU_IRQn</code>	RCU interrupt
<code>EXTI0_1_IRQn</code>	EXTI line 0 and 1 interrupts
<code>EXTI2_3_IRQn</code>	EXTI line 2 and 3 interrupts
<code>EXTI4_15_IRQn</code>	EXTI line 4 to 15 interrupts
<code>TSI_IRQn</code>	TSI interrupt
<code>DMA_Channel0_IRQn</code>	DMA channel 0 global interrupt
<code>DMA_Channel1_2_IRQn</code>	DMA channel 1 and channel 2 interrupts
<code>DMA_Channel3_4_IRQn</code>	DMA channel 3 and channel 4 interrupts
<code>ADC_CMP_IRQn</code>	ADC, CMP0 and CMP1 interrupts
<code>TIMER0_BRK_UP_TRG_COM_IRQn</code>	TIMER0 break, update, trigger and commutation interrupts
<code>TIMER0_Channel_IRQn</code>	TIMER0 channel capture compare interrupts
<code>TIMER1_IRQn</code>	TIMER1 interrupt
<code>TIMER2_IRQn</code>	TIMER2 interrupt
<code>TIMER5_DAC_IRQn</code>	TIMER5 and DAC interrupts
<code>TIMER13_IRQn</code>	TIMER13 interrupt
<code>TIMER14_IRQn</code>	TIMER14 interrupt
<code>TIMER15_IRQn</code>	TIMER15 interrupt
<code>TIMER16_IRQn</code>	TIMER16 interrupt
<code>I2C0_EV_IRQn</code>	I2C0 event interrupt
<code>I2C1_EV_IRQn</code>	I2C1 event interrupt
<code>SPI0_IRQn</code>	SPI0 interrupt
<code>SPI1_IRQn</code>	SPI1 interrupt
<code>USART0_IRQn</code>	USART0 interrupt
<code>USART1_IRQn</code>	USART1 interrupt
<code>CEC_IRQn</code>	CEC interrupt
<code>I2C0_ER_IRQn</code>	I2C0 error interrupt
<code>I2C1_ER_IRQn</code>	I2C1 error interrupt

Member name	Function description
I2C2_EV_IRQn	I2C2 event interrupt
I2C2_ER_IRQn	I2C2 error interrupt
USBD_LP_IRQn	USBD_LP interrupt
USBD_HP_IRQn	USBD_HP interrupt
USBDWakeUp_IRQChannel	USBD_WKUP interrupt
DMA_Channel5_6_IRQn	DMA channel 5 and channel 6 interrupts
SPI2_IRQn	SPI2 global interrupt

## nvic\_priority\_group\_set

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

**Table 3-263. Function nvic\_priority\_group\_set**

Function name	nvic_priority_group_set
Function prototype	void nvic_priority_group_set(uint32_t nvic_prigroup);
Function descriptions	set the priority group
Precondition	-
The called functions	-
Input parameter{in}	
nvic_prigroup	priority group
NVIC_PRIGROUP_PRE0_SUB4	0 bits for pre-emption priority 4 bits for subpriority
NVIC_PRIGROUP_PRE1_SUB3	1 bits for pre-emption priority 3 bits for subpriority
NVIC_PRIGROUP_PRE2_SUB2	2 bits for pre-emption priority 2 bits for subpriority
NVIC_PRIGROUP_PRE3_SUB1	3 bits for pre-emption priority 1 bits for subpriority
NVIC_PRIGROUP_PRE4_SUB0	4 bits for pre-emption priority 0 bits for subpriority
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* priority group configuration, 0 bits for pre-emption priority 4 bits for subpriority */
nvic_priority_group_set(NVIC_PRIGROUP_PRE0_SUB4);
```

## nvic\_irq\_enable

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

Table 3-264. Function nvic\_irq\_enable

Function name	nvic_irq_enable
Function prototype	void nvic_irq_enable(IRQn_Type nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority);
Function descriptions	enable NVIC request
Precondition	-
The called functions	nvic_priority_group_set
Input parameter{in}	
nvic_irq	NVIC interrupt, refer to enum <a href="#">Table 3-262. Enum IRQn_Type</a>
Input parameter{in}	
nvic_irq_pre_priority	the pre-emption priority needed to set
Input parameter{in}	
nvic_irq_sub_priority	the subpriority needed to set
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable EXTI line 0 and 1 interrupts, nvic_irq_pre_priority is 2, nvic_irq_sub_priority is 0 */
nvic_irq_enable(EXTI0_1_IRQn, 2U, 0U);
```

### nvic\_irq\_disable

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

Table 3-265. Function nvic\_irq\_disable

Function name	nvic_irq_disable
Function prototype	void nvic_irq_disable(IRQn_Type nvic_irq);
Function descriptions	disable NVIC request
Precondition	-
The called functions	-
Input parameter{in}	
nvic_irq	NVIC interrupt, refer to enum <a href="#">Table 3-262. Enum IRQn_Type</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable EXTI line 0 and 1 interrupts */
nvic_irq_disable(EXTI0_1_IRQn);
```

## nvic\_vector\_table\_set

The description of nvic\_vector\_table\_set is shown as below:

**Table 3-266. Function nvic\_vector\_table\_set**

<b>Function name</b>	nvic_vector_table_set
<b>Function prototype</b>	void nvic_vector_table_set(uint32_t nvic_vect_tab, uint32_t offset);
<b>Function descriptions</b>	set the NVIC vector table address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>nvic_vect_tab</b>	the RAM or FLASH base address
<i>NVIC_VECTTAB_RAM</i>	RAM base address
<i>NVIC_VECTTAB_FLASH</i>	Flash base address
<i>H</i>	
<b>Input parameter{in}</b>	
<b>Offset</b>	Vector Table offset (vector table start address= base address+offset)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set vector table address = NVIC_VECTTAB_FLASH + 0x200 */
nvic_vector_table_set(NVIC_VECTTAB_FLASH, 0x200);
```

## system\_lowpower\_set

The description of system\_lowpower\_set is shown as below:

**Table 3-267. Function system\_lowpower\_set**

<b>Function name</b>	system_lowpower_set
<b>Function prototype</b>	void system_lowpower_set(uint8_t lowpower_mode);
<b>Function descriptions</b>	set the state of the low power mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lowpower_mode</b>	the low power mode state
<i>SCB_LPM_SLEEP_EXIT_ISR</i>	if chose this para, the system always enter low power mode by exiting from ISR
<i>SCB_LPM_DEEPSLEEP</i>	if chose this para, the system will enter the DEEPSLEEP mode
<i>P</i>	
<i>SCB_LPM_WAKE_BY_ALL_INT</i>	if chose this para, the lowpower mode can be woke up by all the enable and disable interrupts

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* the system always enter low power mode by exiting from ISR */
system_lowpower_set(SCB_LPM_SLEEP_EXIT_ISR);
```

### system\_lowpower\_reset

The description of system\_lowpower\_reset is shown as below:

**Table 3-268. Function system\_lowpower\_reset**

Function name	system_lowpower_reset
Function prototype	void system_lowpower_reset(uint8_t lowpower_mode);
Function descriptions	reset the state of the low power mode
Precondition	-
The called functions	-
Input parameter{in}	
lowpower_mode	the low power mode state
SCB_LPM_SLEEP_EXIT_ISR	if chose this para, the system will exit low power mode by exiting from ISR
SCB_LPM_DEEPSLEEP	if chose this para, the system will enter the SLEEP mode
SCB_LPM_WAKE_BY_ALL_INT	if chose this para, the lowpower mode only can be woke up by the enable interrupts
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* the system will exit low power mode by exiting from ISR */
system_lowpower_reset(SCB_LPM_SLEEP_EXIT_ISR);
```

### systick\_clksource\_set

The description of systick\_clksource\_set is shown as below:

**Table 3-269. Function systick\_clksource\_set**

Function name	systick_clksource_set
Function prototype	void systick_clksource_set(uint32_t systick_clksource);
Function descriptions	set the systick clock source



<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>systick_clksource</b>	the systick clock source needed to choose
SYSTICK_CLKSOURC E_HCLK	systick clock source is from HCLK
SYSTICK_CLKSOURC E_HCLK_DIV8	systick clock source is from HCLK/8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* systick clock source is HCLK/8 */
```

```
systick_clksource_set(SYSTICK_CLKSOURCE_HCLK_DIV8);
```

## 3.15. 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 [3.15.1](#), the PMU firmware functions are introduced in chapter [3.15.2](#).

### 3.15.1. Descriptions of Peripheral registers

PMU registers are listed in the table shown as below:

**Table 3-270. PMU Registers**

Registers	Descriptions
PMU_CTL	PMU control register
PMU_CS	PMU control and status register

### 3.15.2. Descriptions of Peripheral functions

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

**Table 3-271. PMU firmware function**

Function name	Function description
pmu_deinit	reset PMU registers
pmu_lvd_select	select low voltage detector threshold
pmu_lvd_disable	disable PMU lvd
pmu_to_sleepmode	PMU work in sleep mode
pmu_to_deepsleepmode	PMU work at deepsleep mode

Function name	Function description
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 3-272. 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 3-273. 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

<i>PMU_LVDT_1</i>	voltage threshold is 2.3V
<i>PMU_LVDT_2</i>	voltage threshold is 2.4V
<i>PMU_LVDT_3</i>	voltage threshold is 2.5V
<i>PMU_LVDT_4</i>	voltage threshold is 2.6V
<i>PMU_LVDT_5</i>	voltage threshold is 2.7V
<i>PMU_LVDT_6</i>	voltage threshold is 2.8V
<i>PMU_LVDT_7</i>	voltage threshold is 2.9V
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select low voltage detector threshold as 2.9V */
```

```
pmu_lvd_select (PMU_LVDT_7);
```

### **pmu\_lvd\_disable**

The description of pmu\_lvd\_disable is shown as below:

**Table 3-274. Function pmu\_lvd\_disable**

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

Example:

```
/* disable PMU lvd */
```

```
pmu_lvd_disable ();
```

### **pmu\_to\_sleepmode**

The description of pmu\_to\_sleepmode is shown as below:

**Table 3-275. Function pmu\_to\_sleepmode**

<b>Function name</b>	pmu_to_sleepmode
----------------------	------------------

<b>Function prototype</b>	void pmu_to_sleepmode(uint8_t sleepmodecmd);
<b>Function descriptions</b>	PMU work at sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sleepmodecmd</b>	command to enter sleep mode
<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 sleep mode */
pmu_to_sleepmode (WFI_CMD);
```

### pmu\_to\_deepsleepmode

The description of pmu\_to\_deepsleepmode is shown as below:

**Table 3-276. Function pmu\_to\_deepsleepmode**

<b>Function name</b>	pmu_to_deepsleepmode
<b>Function prototype</b>	void pmu_to_deepsleepmode(uint32_t ldo,uint8_t deepsleepmodecmd);
<b>Function descriptions</b>	PMU work at deepsleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ldo</b>	ldo work mode
<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 3-277. 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 3-278. Function pmu\_wakeup\_pin\_enable**

<b>Function name</b>	pmu_wakeup_pin_enable
<b>Function prototype</b>	void pmu_wakeup_pin_enable(uint32_t wakeup_pin);
<b>Function descriptions</b>	enable wakeup pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_pin</b>	Wakeup pin
<i>PMU_WAKEUP_PIN0</i>	WKUP Pin 0
<i>PMU_WAKEUP_PIN1</i>	WKUP Pin 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable wakeup pin0 */

pmu_wakeup_pin_enable (PMU_WAKEUP_PIN0);
```

### pmu\_wakeup\_pin\_disable

The description of pmu\_wakeup\_pin\_disable is shown as below:

**Table 3-279. Function pmu\_wakeup\_pin\_disable**

<b>Function name</b>	pmu_wakeup_pin_disable
<b>Function prototype</b>	void pmu_wakeup_pin_disable(uint32_t wakeup_pin);
<b>Function descriptions</b>	disable wakeup pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_pin</b>	Wakeup pin
<i>PMU_WAKEUP_PIN0</i>	WKUP Pin 0
<i>PMU_WAKEUP_PIN1</i>	WKUP Pin 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable wakeup pin0 */

pmu_wakeup_pin_disable (PMU_WAKEUP_PIN0);
```

### pmu\_backup\_write\_enable

The description of pmu\_backup\_write\_enable is shown as below:

**Table 3-280. 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 3-281. 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 3-282. 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
<i>PMU_FLAG_RESET_WAKEUP</i>	reset wakeup flag
<i>PMU_FLAG_RESET_STANDBY</i>	reset standby flag
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* clear flag bit */
```

```
pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

### pmu\_flag\_get

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

**Table 3-283. 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);
```

## 3.16. 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 [3.16.1](#), the RCU firmware functions are introduced in chapter [3.16.2](#).



### 3.16.1. Descriptions of Peripheral registers

**Table 3-284. RCU Registers**

Registers	Descriptions
RCU_CTL0	Control register 0
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_CFG2	Clock configuration register 2
RCU_CTL1	Control register 1
RCU_ADDAPB1EN	APB1 additional enable register
RCU_ADDAPB1RST	APB1 additional reset register
RCU_VKEY	Unlock voltage register
RCU_DSV	Deep-sleep mode voltage register
RCU_PDVSEL	Power down voltage select register

### 3.16.2. Descriptions of Peripheral functions

**Table 3-285. RCU firmware function**

Function name	Function description
rcu_deinit	deinitialize the RCU
rcu_periph_clock_enable	enable the peripherals clock
rcu_periph_clock_disable	disable the peripherals clock
rcu_periph_clock_sleep_enable	enable the peripherals clock when in sleep mode
rcu_periph_clock_sleep_disable	disable the peripherals clock when in sleep mode
rcu_periph_reset_enable	enable the peripherals reset
rcu_periph_reset_disable	disable the peripheral reset
rcu_bkp_reset_enable	enable the BKP domain reset
rcu_bkp_reset_disable	disable the BKP domain reset
rcu_system_clock_source_config	configure the system clock source
rcu_system_clock_source_get	get the system clock source
rcu_ahb_clock_config	configure the AHB clock prescaler selection
rcu_apb1_clock_config	configure the APB1 clock prescaler selection
rcu_apb2_clock_config	configure the APB2 clock prescaler selection

Function name	Function description
rcu_adc_clock_config	configure the ADC clock source and prescaler selection
rcu_usbd_clock_config	configure the USB D prescaler selection
rcu_ckout_config	configure the CK_OUT clock source and divider
rcu_pll_config	configure the PLL clock source selection and PLL multiply factor
rcu_usart_clock_config	configure the USART clock source selection
rcu_cec_clock_config	configure the CEC clock source selection
rcu_rtc_clock_config	configure the RTC clock source selection
rcu_hxtal_prediv_config	configure the HXTAL divider used as input of PLL
rcu_lxtal_drive_capability_config	configure the LXTAL drive capability
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_irc14m_adjust_value_set	set the IRC14M adjust value
rcu_voltage_key_unlock	unlock Deep-sleep mode voltage register
rcu_deepsleep_voltage_set	set the deep-sleep mode voltage value
rcu_power_down_voltage_set	set the power down voltage
rcu_clock_freq_get	get the system clock, bus and peripheral clock frequency

## reg\_idx

**Table 3-286. Enum reg\_idx**

enum name	Function description
IDX_AHBEN	index of AHB enable register
IDX_APB2EN	index of APB2 enable register
IDX_APB1EN	index of APB1 enable register
IDX_ADDAPB1EN	index of APB1 additional enable register
IDX_AHBRST	index of AHB reset register
IDX_APB2RST	index of APB2 reset register

enum name	Function description
IDX_APB1RST	index of APB1 reset register
IDX_ADDAPB1RST	index of APB1 additional reset register
IDX_CTL0	index of control register0
IDX_BDCTL	index of backup domain control register
IDX_CTL1	index of control register1
IDX_RSTSCK	index of reset source / clock register
IDX_INT	index of interrupt register
IDX_CFG0	index of configuration register0
IDX_CFG2	index of configuration register2

### rcu\_periph\_enum

Table 3-287. Enum rcu\_periph\_enum

enum name	Function description
RCU_DMA	DMA clock
RCU_CRC	CRC clock
RCU_GPIOA	GPIOA clock
RCU_GPIOB	GPIOB clock
RCU_GPIOC	GPIOC clock
RCU_GPIOD	GPIOD clock
RCU_GPIOF	GPIOF clock
RCU_TSI	TSI clock
RCU_CFGCMP	CFGCMP clock
RCU_ADC	ADC clock
RCU_TIMER0	TIMER0 clock
RCU_SPI0	SPI0 clock
RCU_USART0	USART0 clock
RCU_TIMER14	TIMER14 clock
RCU_TIMER15	TIMER15 clock
RCU_TIMER16	TIMER16 clock
RCU_TIMER1	TIMER1 clock
RCU_TIMER2	TIMER2 clock
RCU_TIMER5	TIMER5 clock
RCU_TIMER13	TIMER13 clock
RCU_WWDGT	WWDGT clock
RCU_SPI1	SPI1 clock
RCU_SPI2	SPI2 clock
RCU_USART1	USART1 clock
RCU_I2C0	I2C0 clock
RCU_I2C1	I2C1 clock
RCU_USBD	USBD clock

enum name	Function description
RCU_PMU	PMU clock
RCU_DAC	DAC clock
RCU_CEC	CEC clock
RCU_RTC	RTC clock
RCU_I2C2	I2C2 clock

### rcu\_periph\_sleep\_enum

**Table 3-288. Enum rcu\_periph\_sleep\_enum**

enum name	Function description
RCU_SRAM_SLP	SRAM clock when sleep mode
RCU_FMC_SLP	FMC clock when sleep mode

### rcu\_periph\_reset\_enum

**Table 3-289. Enum rcu\_periph\_reset\_enum**

enum name	Function description
RCU_GPIOARST	GPIOA reset
RCU_GPIOBRST	GPIOB reset
RCU_GPIOCRST	GPIOC reset
RCU_GPIODRST	GPIOD reset
RCU_GPIOFRST	GPIOF reset
RCU_TSIrst	TSI reset
RCU_CFGCMRST	CFGCMP reset
RCU_ADCRST	ADC reset
RCU_TIMER0RST	TIMER0 reset
RCU_SPI0RST	SPI0 reset
RCU_USART0RST	USART0 reset
RCU_TIMER14RST	TIMER14 reset
RCU_TIMER15RST	TIMER15 reset
RCU_TIMER16RST	TIMER16 reset
RCU_TIMER1RST	TIMER1 reset
RCU_TIMER2RST	TIMER2 reset
RCU_TIMER5RST	TIMER5 reset
RCU_TIMER13RST	TIMER13 reset
RCU_WWDGTRST	WWDGT reset
RCU_SPI1RST	SPI1 reset
RCU_SPI2RST	SPI2 reset
RCU_USART1RST	USART1 reset
RCU_I2C0RST	I2C0 reset
RCU_I2C1RST	I2C1 reset
RCU_USBD RST	USBD reset

enum name	Function description
RCU_PMURST	PMU reset
RCU_DACRST	DAC reset
RCU_CECRST	CEC reset
RCU_I2C2RST	I2C2 reset

### rcu\_flag\_enum

**Table 3-290. Enum rcu\_flag\_enum**

enum name	Function description
RCU_FLAG_IRC40KST B	IRC40K stabilization flags
RCU_FLAG_LXTALST B	LXTAL stabilization flags
RCU_FLAG_IRC8MST B	IRC8M stabilization flags
RCU_FLAG_HXTALST B	HXTAL stabilization flags
RCU_FLAG_PLLSTB	PLL stabilization flags
RCU_FLAG_IRC14MS TB	IRC14M stabilization flags
RCU_FLAG_V12RST	1.2V domain Power reset flags
RCU_FLAG_OBLRST	Option byte loader reset flags
RCU_FLAG_EPRST	External PIN reset flags
RCU_FLAG_PORRST	Power reset flags
RCU_FLAG_SWRST	Software reset flags
RCU_FLAG_FWDGTR ST	Free watchdog timer reset flags
RCU_FLAG_WWDGTR ST	Window watchdog timer reset flags
RCU_FLAG_LPRST	Low-power reset flags

### rcu\_int\_flag\_enum

**Table 3-291. Enum rcu\_int\_flag\_enum**

enum name	Function description
RCU_INT_FLAG_IRC4 0KSTB	IRC40K stabilization interrupt flag
RCU_INT_FLAG_LXTA LSTB	LXTAL stabilization interrupt flag
RCU_INT_FLAG_IRC8 MSTB	IRC8M stabilization interrupt flag
RCU_INT_FLAG_HXTA	HXTAL stabilization interrupt flag

enum name	Function description
LSTB	
RCU_INT_FLAG_PLLS TB	PLL stabilization interrupt flag
RCU_INT_FLAG_IRC1 4MSTB	IRC14M stabilization interrupt flag
RCU_INT_FLAG_CKM	CKM interrupt flag

### rcu\_int\_flag\_clear\_enum

**Table 3-292. Enum rcu\_int\_flag\_clear\_enum**

enum name	Function description
RCU_INT_FLAG_IRC4 0KSTB_CLR	IRC40K stabilization interrupt flags clear
RCU_INT_FLAG_LXTA LSTB_CLR	LXTAL stabilization interrupt flags clear
RCU_INT_FLAG_IRC8 MSTB_CLR	IRC8M stabilization interrupt flags clear
RCU_INT_FLAG_HXTA LSTB_CLR	HXTAL stabilization interrupt flags clear
RCU_INT_FLAG_PLLS TB_CLR	PLL stabilization interrupt flags clear
RCU_INT_FLAG_IRC1 4MSTB_CLR	IRC14M stabilization interrupt flags clear
RCU_INT_FLAG_CKM _CLR	CKM interrupt flags clear

### rcu\_int\_enum

**Table 3-293. Enum rcu\_int\_enum**

enum name	Function description
RCU_INT_IRC40KSTB	IRC40K stabilization interrupt
RCU_INT_LXTALSTB	LXTAL stabilization interrupt
RCU_INT_IRC8MSTB	IRC8M stabilization interrupt
RCU_INT_HXTALSTB	HXTAL stabilization interrupt
RCU_INT_PLLSTB	PLL stabilization interrupt
RCU_INT_IRC14MSTB	IRC14M stabilization interrupt

### rcu\_adc\_clock\_enum

**Table 3-294. Enum rcu\_adc\_clock\_enum**

enum name	Function description
RCU_ADCCCK_IRC14M	ADC clock source select IRC14

enum name	Function description
RCU_ADCCCK_APB2_D IV2	ADC clock source select APB2/2
RCU_ADCCCK_APB2_D IV4	ADC clock source select APB2/4
RCU_ADCCCK_APB2_D IV6	ADC clock source select APB2/6
RCU_ADCCCK_APB2_D IV8	ADC clock source select APB2/8

### rcu\_osci\_type\_enum

**Table 3-295. Enum rcu\_osci\_type\_enum**

enum name	Function description
RCU_HXTAL	HXTAL
RCU_LXTAL	LXTAL
RCU_IRC8M	IRC8M
RCU_IRC14M	IRC14M
RCU_IRC40K	IRC40K
RCU_PLL_CK	PLL

### rcu\_clock\_freq\_enum

**Table 3-296. Enum rcu\_clock\_freq\_enum**

enum name	Function description
CK_SYS	system clock
CK_AHB	AHB clock
CK_APB1	APB1 clock
CK_APB2	APB2 clock
CK_ADC	ADC clock
CK_CEC	CEC clock
CK_USART	USART clock

### rcu\_deinit

The description of rcu\_deinit is shown as below:

**Table 3-297. Function rcu\_deinit**

<b>Function name</b>	rcu_deinit
<b>Function prototype</b>	void rcu_deinit(void);
<b>Function descriptions</b>	deinitialize the RCU, reset the value of all RCU registers into initial values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

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

Example:

```
/* reset RCU */
```

```
rcu_deinit();
```

### rcu\_periph\_clock\_enable

The description of rcu\_periph\_clock\_enable is shown as below:

**Table 3-298. 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>	-
Input parameter{in}	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-287. Enum rcu_periph_enum.</a>
<i>RCU_GPIOx</i>	GPIO ports clock (x=A,B,C,D,F)
<i>RCU_DMA</i>	DMA clock
<i>RCU_CRC</i>	CRC clock
<i>RCU_TSI</i>	TSI clock
<i>RCU_CFGCMP</i>	CFGCMP clock
<i>RCU_ADC</i>	ADC clock
<i>RCU_TIMERx</i>	TIMERx clock(x=0,1,2,5,13,14,15,16)
<i>RCU_SPIx</i>	SPIx clock (x=0,1,2)
<i>RCU_USARTx</i>	USARTx clock (x=0,1)
<i>RCU_WWDGT</i>	WWDGT clock
<i>RCU_I2Cx</i>	I2Cx clock (x=0,1,2)
<i>RCU_USBD</i>	USBD clock
<i>RCU_PMU</i>	PMU clock
<i>RCU_DAC</i>	DAC clock
<i>RCU_CEC</i>	CEC clock
<i>RCU_RTC</i>	RTC clock
Output parameter{out}	
-	-
Return value	
-	-



Example:

```
/* enable the USART0 clock */

rcu_periph_clock_enable(RCU_USART0);
```

### rcu\_periph\_clock\_disable

The description of rcu\_periph\_clock\_disable is shown as below:

**Table 3-299. Function rcu\_periph\_clock\_disable**

Function name	rcu_periph_clock_disable
Function prototype	void rcu_periph_clock_disable(rcu_periph_enum periph);
Function descriptions	disable the peripherals clock
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to <a href="#">Table 3-287. Enum rcu_periph_enum</a> .
RCU_GPIOx	GPIO ports clock (x=A,B,C,D,F)
RCU_DMA	DMA clock
RCU_CRC	CRC clock
RCU_TSI	TSI clock
RCU_CFGCMP	CFGCMP clock
RCU_ADC	ADC clock
RCU_TIMERx	TIMERx clock(x=0,1,2,5,13,14,15,16)
RCU_SPIx	SPIx clock (x=0,1,2)
RCU_USARTx	USARTx clock (x=0,1)
RCU_WWDGT	WWDGT clock
RCU_I2Cx	I2Cx clock (x=0,1,2)
RCU_USBD	USBD clock
RCU_PMU	PMU clock
RCU_DAC	DAC clock
RCU_CEC	CEC clock
RCU_RTC	RTC clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the USART0 clock */

rcu_periph_clock_disable(RCU_USART0);
```

## rcu\_periph\_clock\_sleep\_enable

The description of rcu\_periph\_clock\_sleep\_enable is shown as below:

**Table 3-300. Function rcu\_periph\_clock\_sleep\_enable**

<b>Function name</b>	rcu_periph_clock_sleep_enable
<b>Function prototype</b>	void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);
<b>Function descriptions</b>	enable the peripherals clock when in sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-288. Enum rcu_periph_sleep_enum</a> .
<i>RCU_FMC_SLP</i>	FMC clock when sleep mode
<i>RCU_SRAM_SLP</i>	SRAM clock when sleep mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the FMC clock when in sleep mode */
rcu_periph_clock_sleep_enable(RCU_FMC_SLP);
```

## rcu\_periph\_clock\_sleep\_disable

The description of rcu\_periph\_clock\_sleep\_disable is shown as below:

**Table 3-301. Function rcu\_periph\_clock\_sleep\_disable**

<b>Function name</b>	rcu_periph_clock_sleep_disable
<b>Function prototype</b>	void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph);
<b>Function descriptions</b>	disable the peripherals clock when in sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-288. Enum rcu_periph_sleep_enum</a> .
<i>RCU_FMC_SLP</i>	FMC clock when sleep mode
<i>RCU_SRAM_SLP</i>	SRAM clock when sleep mode
<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 3-302. Function rcu\_periph\_reset\_enable**

<b>Function name</b>	rcu_periph_reset_enable
<b>Function prototype</b>	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);
<b>Function descriptions</b>	enable the peripherals reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph_reset</b>	RCU peripherals reset, refer to <a href="#">Table 3-289. Enum rcu_periph_reset_enum</a> .
<i>RCU_GPIOxRST</i>	reset GPIO ports clock (x=A,B,C,D,F)
<i>RCU_TSIrst</i>	reset TSI clock
<i>RCU_CFGCMPrst</i>	reset CFGCMP clock
<i>RCU_ADCRST</i>	reset ADC clock
<i>RCU_TIMERxRST</i>	reset TIMERx clock (x=0,1,2,5,13,14,15,16)
<i>RCU_SPIxRST</i>	reset SPIx clock (x=0,1,2)
<i>RCU_USARTxRST</i>	reset USARTx clock (x=0,1)
<i>RCU_WWDGTRST</i>	reset WWDGT clock
<i>RCU_I2CxRST</i>	reset I2Cx clock (x=0,1,2)
<i>RCU_USBD RST</i>	reset USB D clock
<i>RCU_PMuRST</i>	reset PMU clock
<i>RCU_DACRST</i>	reset DAC
<i>RCU_CEC RST</i>	reset CEC
<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 3-303. Function rcu\_periph\_reset\_disable

<b>Function name</b>	rcu_periph_reset_disable
<b>Function prototype</b>	void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);
<b>Function descriptions</b>	disable the peripheral reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph_reset</b>	RCU peripherals reset, refer to <a href="#">Table 3-289. Enum rcu_periph_reset_enum</a> .
<i>RCU_GPIOxRST</i>	reset GPIO ports clock (x=A,B,C,D,F)
<i>RCU_TSIrst</i>	reset TSI clock
<i>RCU_CFGCMPrst</i>	reset CFGCMP clock
<i>RCU_ADcRst</i>	reset ADC clock
<i>RCU_TIMERxRst</i>	reset TIMERx clock (x=0,1,2,5,13,14,15,16)
<i>RCU_SPIxRst</i>	reset SPIx clock (x=0,1,2)
<i>RCU_USARTxRst</i>	reset USARTx clock (x=0,1)
<i>RCU_WWDGTRst</i>	reset WWDGT clock
<i>RCU_I2CxRst</i>	reset I2Cx clock (x=0,1,2)
<i>RCU_USBDRst</i>	reset USB clock
<i>RCU_PMuRst</i>	reset PMU clock
<i>RCU_DAcRst</i>	reset DAC
<i>RCU_CECRst</i>	reset CEC
<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 3-304. 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>	

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

Example:

```
/* reset the BKP domain */
rcu_bkp_reset_enable();
```

### rcu\_bkp\_reset\_disable

The description of rcu\_bkp\_reset\_disable is shown as below:

**Table 3-305. 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>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

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 3-306. 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>	-
Input parameter{in}	
<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 3-307. 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 3-308. 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 3-309. 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_DIVx</i>	select (CK_AHB / x) as CK_APB1 (x=1,2,4,8,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure CK_AHB/16 as CK_APB1 */
```

```
rcu_apb1_clock_config(RCU_APB1_CKAHB_DIV16);
```

### rcu\_apb2\_clock\_config

The description of rcu\_apb2\_clock\_config is shown as below:

Table 3-310. 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_CK_AHB_D IVx	select (CK_AHB / x) as CK_APB2 clock (x=1,2,4,8,16)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure CK_AHB/8 as CK_APB2 */
rcu_apb2_clock_config(RCU_APB2_CK_AHB_DIV8);
```

### rcu\_adc\_clock\_config

The description of rcu\_adc\_clock\_config is shown as below:

Table 3-311. Function rcu\_adc\_clock\_config

Function name	rcu_adc_clock_config
Function prototype	void rcu_adc_clock_config(rcu_adc_clock_enum ck_adc);
Function descriptions	configure the ADC clock prescaler selection
Precondition	-
The called functions	-
Input parameter{in}	
ck_adc	ADC clock prescaler selection, refer to <a href="#">Table 3-294. Enum rcu_adc_clock_enum</a> .
RCU_ADCCCK_IRC14M	select CK_IRC14M as CK_ADC
RCU_ADCCCK_IRC28M	select CK_IRC28M as CK_ADC
RCU_ADCCCK_APB2_D IVx	select (CK_APB2 / x) as CK_ADC(x=2,4,6,8)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the ADC prescaler factor */
```



```
rcu_adc_clock_config(RCU_CKADC_CKAPB2_DIV8);
```

### rcu\_usbd\_clock\_config

The description of rcu\_usbd\_clock\_config is shown as below:

**Table 3-312. Function rcu\_usbd\_clock\_config**

<b>Function name</b>	rcu_usbd_clock_config
<b>Function prototype</b>	void rcu_usbd_clock_config(uint32_t ck_usbd);
<b>Function descriptions</b>	configure the USB_D clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_usbd</b>	USB_D clock prescaler selection
<i>RCU_USB_D_CKPLL_D</i> <i>IV1</i>	select CK_PLL as CK_USB_D
<i>RCU_USB_D_CKPLL_D</i> <i>IV1_5</i>	select CK_PLL / 1.5 as CK_USB_D
<i>RCU_USB_D_CKPLL_D</i> <i>IV2</i>	select CK_PLL / 2 as CK_USB_D
<i>RCU_USB_D_CKPLL_D</i> <i>IV2_5</i>	select CK_PLL / 2.5 as CK_USB_D
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the CK_PLL as CK_USB_D */
```

```
rcu_usbd_clock_config(RCU_USB_D_CKPLL_DIV1);
```

### rcu\_ckout\_config

The description of rcu\_ckout\_config is shown as below:

**Table 3-313. Function rcu\_ckout\_config**

<b>Function name</b>	rcu_ckout_config
<b>Function prototype</b>	void rcu_ckout_config(uint32_t ckout_src, uint32_t ckout_div);
<b>Function descriptions</b>	configure the CK_OUT clock source and division factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ckout_src</b>	CK_OUT clock source selection
<i>RCU_CKOUTSRC_NO</i>	no clock selected

<i>NE</i>	
<i>RCU_CKOUTSRC_IRC</i> <i>14M</i>	select high speed 14M internal oscillator clock
<i>RCU_CKOUTSRC_IRC</i> <i>40K</i>	select high speed 40K internal oscillator clock
<i>RCU_CKOUTSRC_LX</i> <i>TAL</i>	select LXTAL clock
<i>RCU_CKOUTSRC_CK</i> <i>SYS</i>	select system clock CK_SYS
<i>RCU_CKOUTSRC_IRC</i> <i>8M</i>	select high speed 8M internal oscillator clock
<i>RCU_CKOUTSRC_HX</i> <i>TAL</i>	select HXTAL clock
<i>RCU_CKOUTSRC_CK</i> <i>PLL_DIV1</i>	select CK_PLL clock
<i>RCU_CKOUTSRC_CK</i> <i>PLL_DIV2</i>	Select (CK_PLL / 2)clock
<b>Input parameter{in}</b>	
<b>ckout_div</b>	CK_OUT divider
<i>RCU_CKOUT_DIVx</i>	CK_OUT is divided by x(x=1,2,4,8,16,32,64,128)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the HXTAL as CK_OUT clock source */
```

```
rcu_ckout_config(RCU_CKOUTSRC_HXTAL, RCU_CKOUT_DIV1);
```

### rcu\_ckout0\_config

The description of rcu\_ckout0\_config is shown as below:

**Table 3-314. Function rcu\_ckout0\_config**

<b>Function name</b>	rcu_ckout0_config
<b>Function prototype</b>	void rcu_ckout0_config(uint32_t ckout0_src, uint32_t ckout0_div);
<b>Function descriptions</b>	configure the CK_OUT0 clock source and divoision factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ckout0_src</b>	CK_OUT0 clock source selection
<i>RCU_CKOUT0SRC_N</i> <i>ONE</i>	no clock selected

<i>RCU_CKOUT0SRC_IR</i> <i>C28M</i>	select high speed 28M internal oscillator clock
<i>RCU_CKOUT0SRC_IR</i> <i>C40K</i>	select high speed 40K internal oscillator clock
<i>RCU_CKOUT0SRC_LX</i> <i>TAL</i>	select LXTAL clock
<i>RCU_CKOUT0SRC_C</i> <i>KSYS</i>	select system clock CK_SYS
<i>RCU_CKOUT0SRC_IR</i> <i>C8M</i>	select high speed 8M internal oscillator clock
<i>RCU_CKOUT0SRC_H</i> <i>XTAL</i>	select HXTAL clock
<i>RCU_CKOUT0SRC_C</i> <i>KPLL_DIV1</i>	select CK_PLL clock
<i>RCU_CKOUT0SRC_C</i> <i>KPLL_DIV2</i>	Select (CK_PLL / 2) clock
<b>Input parameter{in}</b>	
<b>ckout0_div</b>	CK_OUT0 divider
<i>RCU_CKOUT0_DIVx</i>	CK_OUT0 is divided by x(x=1,2,4,8,16,32,64,128)
<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_CKOUT0_DIV1);
```

### rcu\_ckout1\_config

The description of rcu\_ckout1\_config is shown as below:

**Table 3-315. Function rcu\_ckout1\_config**

<b>Function name</b>	rcu_ckout1_config
<b>Function prototype</b>	void rcu_ckout1_config(uint32_t ckout1_src, uint32_t ckout1_div);
<b>Function descriptions</b>	configure the CK_OUT1 clock source and division factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ckout1_src</b>	CK_OUT1 clock source selection
<i>RCU_CKOUT1SRC_N</i> <i>ONE</i>	no clock selected
<i>RCU_CKOUT1SRC_IR</i>	select high speed 28M internal oscillator clock

<i>C28M</i>	
<i>RCU_CKOUT1SRC_IRC40K</i>	select high speed 40K internal oscillator clock
<i>RCU_CKOUT1SRC_LXTAL</i>	select LXTAL clock
<i>RCU_CKOUT1SRC_CKSYS</i>	select system clock CK_SYS
<i>RCU_CKOUT1SRC_IRC8M</i>	select high speed 8M internal oscillator clock
<i>RCU_CKOUT1SRC_HXTAL</i>	select HXTAL clock
<i>RCU_CKOUT1SRC_CKPLL_DIV1</i>	select CK_PLL clock
<i>RCU_CKOUT1SRC_CKPLL_DIV2</i>	Select (CK_PLL / 2) clock
<b>Input parameter{in}</b>	
<b>ckout1_div</b>	CK_OUT1 divider
<i>RCU_CKOUT1_DIVx</i>	CK_OUT1 is divided by x(x=1,2,...,64)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the HXTAL as CK_OUT1 clock source */
```

```
rcu_ckout1_config(RCU_CKOUT1SRC_HXTAL, RCU_CKOUT1_DIV1);
```

### rcu\_pll\_config

The description of rcu\_pll\_config is shown as below:

**Table 3-316. 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 PLL clock source selection and PLL multiply factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pll_src</b>	PLL clock source selection
<i>RCU_PLLSRC_IRC8M_DIV2</i>	select CK_IRC8M/2 as PLL source clock
<i>RCU_PLLSRC_HXTAL</i>	select HXTAL as PLL source clock
<b>Input parameter{in}</b>	

<b>pll_mul</b>	PLL clock multiplication factor
<i>RCU_PLL_MULx</i>	PLL source clock * x (x = 2..32)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL */
```

```
rcu_pll_config(RCU_PLLSRC_IRC8M_DIV2, RCU_PLL_MUL10);
```

### rcu\_usart\_clock\_config

The description of rcu\_usart\_clock\_config is shown as below:

**Table 3-317. Function rcu\_usart\_clock\_config**

<b>Function name</b>	rcu_usart_clock_config
<b>Function prototype</b>	void rcu_usart_clock_config(uint32_t ck_usart);
<b>Function descriptions</b>	configure the USART clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_usart</b>	USART clock source selection
<i>RCU_USART0SRC_CKAPB2</i>	CK_USART0 select CK_APB2
<i>RCU_USART0SRC_CKSYS</i>	CK_USART0 select CK_SYS
<i>RCU_USART0SRC_CKLXTAL</i>	CK_USART0 select CK_LXTAL
<i>RCU_USART0SRC_CKIRC8M</i>	CK_USART0 select CK_IRC8M
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the LXTAL as USART0 clock */
```

```
rcu_usart_clock_config(RCU_USART0SRC_LXTAL);
```

### rcu\_cec\_clock\_config

The description of rcu\_cec\_clock\_config is shown as below:

Table 3-318. Function rcu\_cec\_clock\_config

<b>Function name</b>	rcu_cec_clock_config
<b>Function prototype</b>	void rcu_cec_clock_config(uint32_t ck_cec);
<b>Function descriptions</b>	configure the CEC clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_cec</b>	CEC clock source selection
<i>RCU_CECSRC_IRC8M_DIV244</i>	CK_CEC select CK_IRC8M/244
<i>RCU_CECSRC_LXTAL</i>	CK_CEC select CK_LXTAL
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the CEC clock source selection */
rcu_cec_clock_config(RCU_CECSRC_LXTAL);
```

### rcu\_rtc\_clock\_config

The description of rcu\_rtc\_clock\_config is shown as below:

Table 3-319. 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_DIV32</i>	select (CK_HXTAL / 32) 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\_slcd\_clock\_config

The description of rcu\_slcd\_clock\_config is shown as below:

**Table 3-320. Function rcu\_slcd\_clock\_config**

Function name	rcu_slcd_clock_config
Function prototype	void rcu_slcd_clock_config(uint32_t slcd_clock_source);
Function descriptions	configure the SLCD clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
slcd_clock_source	SLCD clock source selection
RCU_SLCDSRC_NONE	no clock selected
RCU_SLCDSRC_LXTAL	select CK_LXTAL as SLCD source clock
RCU_SLCDSRC_IRC40K	select CK_IRC40K as SLCD source clock
RCU_SLCDSRC_HXTAL_DIV32	select (CK_HXTAL / 32) as SLCD source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the CK_SLCD clock source selection */

rcu_slcd_clock_config(RCU_SLCDSRC_IRC40K);
```

### rcu\_hxtal\_prediv\_config

The description of rcu\_hxtal\_prediv\_config is shown as below:

**Table 3-321. Function rcu\_hxtal\_prediv\_config**

Function name	rcu_hxtal_prediv_config
Function prototype	void rcu_hxtal_prediv_config(uint32_t hxtal_prediv);
Function descriptions	configure the HXTAL divider used as input of PLL
Precondition	-
The called functions	-

Input parameter{in}	
<b>hxtal_prediv</b>	HXTAL divider used as input of PLL
<i>RCU_PLL_PREDVx</i>	HXTAL divided x used as input of PLL (x=1..16)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the HXTAL divider */
```

```
rcu_hxtal_prediv_config(RCU_PLL_PREDV2);
```

### rcu\_lxtal\_drive\_capability\_config

The description of rcu\_lxtal\_drive\_capability\_config is shown as below:

**Table 3-322. Function rcu\_lxtal\_drive\_capability\_config**

<b>Function name</b>	rcu_lxtal_drive_capability_config
<b>Function prototype</b>	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);
<b>Function descriptions</b>	configure the LXTAL drive capability
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>lxtal_dricap</b>	drive capability of LXTAL
<i>RCU_LXTAL_LOWDRI</i>	lower driving capability
<i>RCU_LXTAL_MED_LOWDRI</i>	medium low driving capability
<i>RCU_LXTAL_MED_HIGHDRI</i>	medium high driving capability
<i>RCU_LXTAL_HIGHDRI</i>	higher driving capability
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the LXTAL drive capability */
```

```
rcu_lxtal_drive_capability_config (RCU_LXTAL_LOWDRI);
```

### rcu\_flag\_get

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



Table 3-323. Function rcu\_flag\_get

<b>Function name</b>	rcu_flag_get
<b>Function prototype</b>	FlagStatus rcu_flag_get(rcu_flag_enum flag);
<b>Function descriptions</b>	get the clock stabilization and peripheral reset flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	the clock stabilization and peripheral reset flags, refer to <a href="#">Table 3-290. Enum rcu_flag_enum</a> .
<i>RCU_FLAG_IRC40KST</i> <i>B</i>	IRC40K stabilization flag
<i>RCU_FLAG_LXTALST</i> <i>B</i>	LXTAL stabilization flag
<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_IRC14MS</i> <i>TB</i>	IRC14M stabilization flag
<i>RCU_FLAG_V12RST</i>	1.2V domain power reset flag
<i>RCU_FLAG_OBLRST</i>	option byte loader reset 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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the clock stabilization flag */
```

```
if(RESET != rcu_flag_get(RCU_FLAG_LXTALSTB)){
```

```
}
```

## rcu\_all\_reset\_flag\_clear

The description of rcu\_all\_reset\_flag\_clear is shown as below:

**Table 3-324. Function rcu\_all\_reset\_flag\_clear**

<b>Function name</b>	rcu_all_reset_flag_clear
<b>Function prototype</b>	void rcu_all_reset_flag_clear(void);
<b>Function descriptions</b>	clear all the reset flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear all the reset flag */
rcu_all_reset_flag_clear();
```

## rcu\_interrupt\_flag\_get

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

**Table 3-325. Function rcu\_interrupt\_flag\_get**

<b>Function name</b>	rcu_interrupt_flag_get
<b>Function prototype</b>	FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag);
<b>Function descriptions</b>	get the clock stabilization interrupt and ckm flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt and ckm flags, refer to <a href="#">Table 3-291. Enum rcu_int_flag_enum</a> .
<i>RCU_INT_FLAG_IRC4OKSTB</i>	IRC40K stabilization interrupt flag
<i>RCU_INT_FLAG_LXTALLSTB</i>	LXTAL stabilization interrupt flag
<i>RCU_INT_FLAG_IRC8MSTB</i>	IRC8M stabilization interrupt flag
<i>RCU_INT_FLAG_HXTALALSTB</i>	HXTAL stabilization interrupt flag
<i>RCU_INT_FLAG_PLLSTB</i>	PLL stabilization interrupt flag

<i>RCU_INT_FLAG_IRC14MSTB</i>	IRC14M stabilization interrupt flag
<i>RCU_INT_FLAG_CKM</i>	HXTAL clock stuck interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the clock stabilization interrupt flag */
if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){
}

```

### rcu\_interrupt\_flag\_clear

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

**Table 3-326. Function rcu\_interrupt\_flag\_clear**

<b>Function name</b>	rcu_interrupt_flag_clear
<b>Function prototype</b>	void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag_clear)
<b>Function descriptions</b>	clear the interrupt flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag_clear</b>	clock stabilization and stuck interrupt flags clear, refer to <a href="#">Table 3-292. Enum rcu_int_flag_clear_enum</a> .
<i>RCU_INT_FLAG_IRC40KSTB_CLR</i>	IRC40K stabilization interrupt flag clear
<i>RCU_INT_FLAG_LXTALSTB_CLR</i>	LXTAL stabilization interrupt flag clear
<i>RCU_INT_FLAG_IRC8MSTB_CLR</i>	IRC8M stabilization interrupt flag clear
<i>RCU_INT_FLAG_HXTALSTB_CLR</i>	HXTAL stabilization interrupt flag clear
<i>RCU_INT_FLAG_PLLSTB_CLR</i>	PLL stabilization interrupt flag clear
<i>RCU_INT_FLAG_IRC14MSTB_CLR</i>	IRC14M stabilization interrupt flag clear
<i>RCU_INT_FLAG_CKM_CLR</i>	clock stuck interrupt flag clear
<b>Output parameter{out}</b>	
-	-

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 3-327. Function rcu\_interrupt\_enable**

Function name	rcu_interrupt_enable
Function prototype	void rcu_interrupt_enable(rcu_int_enum stab_int);
Function descriptions	enable the stabilization interrupt
Precondition	-
The called functions	-
Input parameter{in}	
stab_int	clock stabilization interrupt, refer to <a href="#">Table 3-293. Enum rcu_int_enum</a> .
RCU_INT_IRC40KSTB	IRC40K stabilization interrupt enable
RCU_INT_LXTALSTB	LXTAL stabilization interrupt enable
RCU_INT_IRC8MSTB	IRC8M stabilization interrupt enable
RCU_INT_HXTALSTB	HXTAL stabilization interrupt enable
RCU_INT_PLLSTB	PLL stabilization interrupt enable
RCU_INT_IRC14MSTB	IRC14M stabilization interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the HXTAL stabilization interrupt */
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

### rcu\_interrupt\_disable

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

**Table 3-328. Function rcu\_interrupt\_disable**

Function name	rcu_interrupt_disable
Function prototype	void rcu_interrupt_disable(rcu_int_enum stab_int);
Function descriptions	disable the stabilization interrupt
Precondition	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>stab_int</b>	clock stabilization interrupt, refer to <a href="#">Table 3-293. Enum rcu_int_enum.</a>
<i>RCU_INT_IRC40KSTB</i>	IRC40K stabilization interrupt disable
<i>RCU_INT_LXTALSTB</i>	LXTAL stabilization interrupt disable
<i>RCU_INT_IRC8MSTB</i>	IRC8M stabilization interrupt disable
<i>RCU_INT_HXTALSTB</i>	HXTAL stabilization interrupt disable
<i>RCU_INT_PLLSTB</i>	PLL stabilization interrupt disable
<i>RCU_INT_IRC14MSTB</i>	IRC14M stabilization interrupt disable
<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 3-329. 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 <a href="#">Table 3-295. Enum rcu_osci_type_enum.</a>
<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_IRC14M</i>	internal 14M RC oscillators(IRC14M)
<i>RCU_IRC40K</i>	internal 40K RC oscillator(IRC40K)
<i>RCU_PLL_CK</i>	phase locked loop(PLL)
<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 3-330. 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 <a href="#">Table 3-295. Enum rcu_osci_type_enum</a> .
<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_IRC14M</i>	internal 14M RC oscillators(IRC14M)
<i>RCU_IRC40K</i>	internal 40K RC oscillator(IRC40K)
<i>RCU_PLL_CK</i>	phase locked loop(PLL)
<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 3-331. Function rcu\_osci\_off**

<b>Function name</b>	rcu_osci_off
<b>Function prototype</b>	void rcu_osci_off(rcu_osci_type_enum osci);
<b>Function descriptions</b>	turn off the oscillator
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-295. Enum rcu_osci_type_enum</a> .
<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_IRC14M</i>	internal 14M RC oscillators(IRC14M)
<i>RCU_IRC40K</i>	internal 40K RC oscillator(IRC40K)
<i>RCU_PLL_CK</i>	phase locked loop(PLL)
<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 3-332. 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 <a href="#">Table 3-295. Enum rcu_osci_type_enum</a> .
<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 3-333. 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 <a href="#">Table 3-295. Enum rcu_osci_type_enum</a> .
<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 3-334. 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 3-335. Function rcu\_hxtal\_clock\_monitor\_disable

Function name	rcu_hxtal_clock_monitor_disable
Function prototype	void rcu_hxtal_clock_monitor_disable(void);
Function descriptions	disable the HXTAL clock monitor
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the HXTAL clock monitor */
rcu_hxtal_clock_monitor_disable();
```

### rcu\_irc8m\_adjust\_value\_set

The description of rcu\_irc8m\_adjust\_value\_set is shown as below:

Table 3-336. Function rcu\_irc8m\_adjust\_value\_set

Function name	rcu_irc8m_adjust_value_set
Function prototype	void rcu_irc8m_adjust_value_set(uint32_t irc8m_adjval);
Function descriptions	set the IRC8M adjust value
Precondition	-
The called functions	-
Input parameter{in}	
irc8m_adjval	IRC8M adjust value, must be between 0 and 0x1F
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the IRC8M adjust value */
rcu_irc8m_adjust_value_set(0x10);
```

### rcu\_irc14m\_adjust\_value\_set

The description of rcu\_irc14m\_adjust\_value\_set is shown as below:

Table 3-337. Function rcu\_irc14m\_adjust\_value\_set

Function name	rcu_irc14m_adjust_value_set
Function prototype	void rcu_irc14m_adjust_value_set(uint32_t irc14m_adjval);
Function descriptions	set the IRC14M adjust value
Precondition	-
The called functions	-
Input parameter{in}	
irc14m_adjval	IRC14M adjust value, must be between 0 and 0x1F
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the IRC14M adjust value */
rcu_irc14m_adjust_value_set(0x10);
```

### rcu\_irc28m\_adjust\_value\_set

The description of rcu\_irc28m\_adjust\_value\_set is shown as below:

Table 3-338. Function rcu\_irc28m\_adjust\_value\_set

Function name	rcu_irc28m_adjust_value_set
Function prototype	void rcu_irc28m_adjust_value_set(uint32_t irc28m_adjval);
Function descriptions	set the IRC28M adjust value
Precondition	-
The called functions	-
Input parameter{in}	
irc28m_adjval	IRC28M adjust value, must be between 0 and 0x1F
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the IRC28M adjust value */
rcu_irc28m_adjust_value_set(0x10);
```

### rcu\_voltage\_key\_unlock

The description of rcu\_voltage\_key\_unlock is shown as below:

Table 3-339. Function rcu\_voltage\_key\_unlock

Function name	rcu_voltage_key_unlock
Function prototype	void rcu_voltage_key_unlock (void);
Function descriptions	unlock the voltage key
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* unlock the voltage key */
rcu_voltage_key_unlock();
```

### rcu\_deepsleep\_voltage\_set

The description of rcu\_deepsleep\_voltage\_set is shown as below:

Table 3-340. Function rcu\_deepsleep\_voltage\_set

Function name	rcu_deepsleep_voltage_set
Function prototype	void rcu_deepsleep_voltage_set(uint32_t dsvol);
Function descriptions	set voltage in deep sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
dsvol	deep sleep mode voltage
RCU_DEEPSLEEP_V_1_2	the core voltage is 1.2V in deep-sleep mode
RCU_DEEPSLEEP_V_1_1	the core voltage is 1.1V in deep-sleep mode
RCU_DEEPSLEEP_V_1_0	the core voltage is 1.0V in deep-sleep mode
RCU_DEEPSLEEP_V_0_9	the core voltage is 0.9V in deep-sleep mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the deep-sleep mode voltage */
```

```
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1_0);
```

### rcu\_power\_down\_voltage\_set

The description of rcu\_power\_down\_voltage\_set is shown as below:

**Table 3-341. Function rcu\_power\_down\_voltage\_set**

<b>Function name</b>	rcu_power_down_voltage_set
<b>Function prototype</b>	void rcu_power_down_voltage_set(uint32_t pdvol);
<b>Function descriptions</b>	set the power down voltage
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pdvol</b>	power down voltage select
<i>RCU_PDR_V_2_6</i>	power down voltage is 2.6V
<i>RCU_PDR_V_1_8</i>	power down voltage is 1.8V
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure power down voltage */
```

```
rcu_power_down_voltage_set(RCU_PDR_V_2_6);
```

### rcu\_clock\_freq\_get

The description of rcu\_clock\_freq\_get is shown as below:

**Table 3-342. 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 and peripheral clock frequency
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>clock</b>	the clock frequency which to get
<i>CK_SYS</i>	system clock frequency
<i>CK_AHB</i>	AHB clock frequency
<i>CK_APB1</i>	APB1 clock frequency
<i>CK_APB2</i>	APB2 clock frequency
<i>CK_ADC</i>	ADC clock frequency

<i>CK_CEC</i>	CEC clock frequency
<i>CK_USART</i>	USART clock frequency
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	clock frequency of system, AHB, APB1, APB2, ADC or USART

Example:

```
uint32_t temp_freq;

/* get the system clock frequency */

temp_freq = rcu_clock_freq_get(CK_SYS);
```

## 3.17. 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 [3.17.1](#), the FWDGT firmware functions are introduced in chapter [3.17.2](#).

### 3.17.1. Descriptions of Peripheral registers

RTC registers are listed in the table shown as below:

**Table 3-343. RTC Registers**

Registers	Descriptions
RTC_TIME	RTC time of day register
RTC_DATE	RTC date register
RTC_CTL	RTC control register
RTC_STAT	RTC status register
RTC_PSC	RTC time prescaler register
RTC_ALRM0TD	RTC alarm 0 time and date register
RTC_WPK	RTC write protection key register
RTC_SS	RTC sub second register
RTC_SHIFTCTL	RTC shift function control register
RTC_TTS	RTC time of timestamp register
RTC_DTS	RTC date of timestamp register
RTC_SSTS	RTC sub second of timestamp register
RTC_HRFC	RTC high resolution frequency compensation register
RTC_TAMP	RTC tamper register
RTC_ALRM0SS	RTC alarm 0 sub second register
RTC_BKP0	RTC backup 0 register

Registers	Descriptions
RTC_BKP1	RTC backup 1 register
RTC_BKP2	RTC backup 2 register
RTC_BKP3	RTC backup 3 register
RTC_BKP4	RTC backup 4 register

### 3.17.2. Descriptions of Peripheral functions

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

**Table 3-344. RTC firmware function**

Function name	Function description
rtc_deinit	reset most of the RTC registers
rtc_init	initialize RTC registers
rtc_init_mode_enter	enter RTC init mode
rtc_init_mode_exit	exit RTC init mode
rtc_register_sync_wait	wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated
rtc_current_time_get	get current time and date
rtc_subsecond_get	get current subsecond value
rtc_alarm_config	configure RTC alarm
rtc_alarm_subsecond_config	configure subsecond of RTC alarm
rtc_alarm_get	get RTC alarm
rtc_alarm_subsecond_get	get RTC alarm subsecond
rtc_alarm_enable	enable RTC alarm
rtc_alarm_disable	disable RTC alarm
rtc_timestamp_enable	enable RTC time-stamp
rtc_timestamp_disable	disable RTC time-stamp
rtc_timestamp_get	get RTC timestamp time and date
rtc_timestamp_subsecond_get	get RTC time-stamp subsecond
rtc_tamper_enable	enable RTC tamper
rtc_tamper_disable	disable RTC tamper
rtc_interrupt_enable	enable specified RTC interrupt
rtc_interrupt_disable	disable specified RTC interrupt
rtc_flag_get	check specified flag
rtc_flag_clear	clear specified flag
rtc_alter_output_config	configure RTC alternate output source
rtc_calibration_config	configure RTC calibration register
rtc_hour_adjust	adjust the daylight saving time by adding or subtracting one hour from the current time
rtc_second_adjust	adjust RTC second or subsecond value of current time

Function name	Function description
rtc_bypass_shadow_enable	enable RTC bypass shadow registers function
rtc_bypass_shadow_disable	disable RTC bypass shadow registers function
rtc_refclock_detection_enable	enable RTC reference clock detection function
rtc_refclock_detection_disable	disable RTC reference clock detection function

### Structure rtc\_parameter\_struct

**Table 3-345. Structure rtc\_parameter\_struct**

Member name	Function description
rtc_year	RTC year value: 0x0 - 0x99(BCD format)
rtc_month	RTC month value
rtc_date	RTC date value: 0x1 - 0x31(BCD format)
rtc_day_of_week	RTC weekday value
rtc_hour	RTC hour value
rtc_minute	RTC minute value: 0x0 - 0x59(BCD format)
rtc_second	RTC second value: 0x0 - 0x59(BCD format)
rtc_factor_asyn	RTC asynchronous prescaler value: 0x0 - 0x7F
rtc_factor_syn	RTC synchronous prescaler value: 0x0 - 0x7FFF
rtc_am_pm	RTC AM/PM value
rtc_display_format	RTC time notation

### Structure rtc\_alarm\_struct

**Table 3-346. Structure rtc\_alarm\_struct**

Member name	Function description
rtc_alarm_mask	RTC alarm mask
rtc_weekday_or_date	specify RTC alarm is on date or weekday
rtc_alarm_day	RTC alarm date or weekday value
rtc_alarm_hour	RTC alarm hour value
rtc_alarm_minute	RTC alarm minute value: 0x0 - 0x59(BCD format)
rtc_alarm_second	RTC alarm second value: 0x0 - 0x59(BCD format)
rtc_am_pm	RTC alarm AM/PM value

### Structure rtc\_timestamp\_struct

**Table 3-347. Structure rtc\_timestamp\_struct**

Member name	Function description
rtc_timestamp_month	RTC time-stamp month value
rtc_timestamp_date	RTC time-stamp date value: 0x1 - 0x31(BCD format)
rtc_timestamp_day	RTC time-stamp weekday value
rtc_timestamp_hour	RTC time-stamp hour value
rtc_timestamp_minute	RTC time-stamp minute value: 0x0 - 0x59(BCD format)

rtc_timestamp_second	RTC time-stamp second value: 0x0 - 0x59(BCD format)
rtc_am_pm	RTC time-stamp AM/PM value

## Structure rtc\_tamper\_struct

**Table 3-348. Structure rtc\_tamper\_struct**

Member name	Function description
rtc_tamper_source	RTC tamper source
rtc_tamper_trigger	RTC tamper trigger
rtc_tamper_filter	RTC tamper consecutive samples needed during a voltage level detection
rtc_tamper_sample_frequency	RTC tamper sampling frequency during a voltage level detection
rtc_tamper_precharge_enable	RTC tamper precharge feature during a voltage level detection
rtc_tamper_precharge_time	RTC tamper precharge duration if precharge feature is enabled
rtc_tamper_with_timestamp	RTC tamper time-stamp feature

## rtc\_deinit

The description of rtc\_deinit is shown as below:

**Table 3-349. Function rtc\_deinit**

Function name	rtc_deinit
Function prototype	ErrStatus rtc_deinit(void);
Function descriptions	reset most of the RTC registers
Precondition	-
The called functions	rcu_periph_reset_enable/ rcu_periph_reset_disable -
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```

/* reset most of the RTC registers*/

ErrStatus error_status = ERROR;

error_status = rtc_deinit();

```



## rtc\_init

The description of rtc\_init is shown as below:

**Table 3-350. Function rtc\_init**

<b>Function name</b>	rtc_init
<b>Function prototype</b>	ErrStatus rtc_init(rtc_parameter_struct* rtc_initpara_struct);
<b>Function descriptions</b>	initialize RTC registers
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>rtc_initpara_struct</b>	pointer to a rtc_parameter_struct structure which contains parameters for initialization of the rtc peripheral, the structure members can refer to members of the structure <a href="#">Table 3-345. Structure rtc_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* reset most of the RTC registers*/
```

```
ErrStatus error_status = ERROR;
```

```
error_status = rtc_init ();
```

## rtc\_init\_mode\_enter

The description of rtc\_init\_mode\_enter is shown as below:

**Table 3-351. Function rtc\_init\_mode\_enter**

<b>Function name</b>	rtc_init_mode_enter
<b>Function prototype</b>	ErrStatus rtc_init_mode_enter(void);
<b>Function descriptions</b>	enter RTC init mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/*enter RTC init mode*/
```

```
ErrStatus error_status = ERROR;
```

```
error_status = rtc_init_mode_enter ();
```

### rtc\_init\_mode\_exit

The description of rtc\_init\_mode\_exit is shown as below:

**Table 3-352. Function rtc\_init\_mode\_exit**

<b>Function name</b>	rtc_init_mode_exit
<b>Function prototype</b>	void rtc_init_mode_exit(void);
<b>Function descriptions</b>	exit RTC init mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*exit RTC init mode*/
```

```
rtc_init_mode_exit ();
```

### rtc\_register\_sync\_wait

The description of rtc\_register\_sync\_wait is shown as below:

**Table 3-353. Function rtc\_register\_sync\_wait**

<b>Function name</b>	rtc_register_sync_wait
<b>Function prototype</b>	ErrStatus rtc_register_sync_wait(void);
<b>Function descriptions</b>	wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/*wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated*/
```

```
ErrStatus error_status = ERROR;
```

```
error_status = rtc_register_sync_wait ();
```

### rtc\_current\_time\_get

The description of rtc\_current\_time\_get is shown as below:

**Table 3-354. Function rtc\_current\_time\_get**

<b>Function name</b>	rtc_current_time_get
<b>Function prototype</b>	void rtc_current_time_get(rtc_parameter_struct* rtc_initpara_struct);
<b>Function descriptions</b>	get current time and date
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>rtc_initpara_struct</b>	pointer to a rtc_parameter_struct structure which contains parameters for initialization of the rtc peripheral, the structure members can refer to members of the structure <a href="#">Table 3-345. Structure rtc_parameter_struct</a> .
<b>Return value</b>	
-	-

Example:

```
/*get current time and date*/
rtc_parameter_struct rtc_initpara_struct;
rtc_current_time_get (&rtc_initpara_struct);
```

### rtc\_subsecond\_get

The description of rtc\_subsecond\_get is shown as below:

**Table 3-355. Function rtc\_subsecond\_get**

<b>Function name</b>	rtc_subsecond_get
<b>Function prototype</b>	uint32_t rtc_subsecond_get(void);
<b>Function descriptions</b>	get current subsecond value
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	current subsecond value(0x00-0xFFFF)

Example:

```
/*get current subsecond value*/
```

```
uint32_t sub_second = 0;

sub_second = rtc_subsecond_get();
```

### rtc\_alarm\_config

The description of rtc\_alarm\_config is shown as below:

**Table 3-356. Function rtc\_alarm\_config**

<b>Function name</b>	rtc_alarm_config
<b>Function prototype</b>	void rtc_alarm_config(rtc_alarm_struct* rtc_alarm_time)
<b>Function descriptions</b>	configure RTC alarm
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>rtc_alarm_time</b>	pointer to a rtc_alarm_struct structure which contains parameters for RTC alarm configuration, the structure members can refer to members of the structure <a href="#">Table 3-346. Structure rtc_alarm_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*rtc_alarm_config*/

rtc_alarm_struct rtc_alarm_time;

rtc_alarm_config(&rtc_alarm_time);
```

### rtc\_alarm\_subsecond\_config

The description of rtc\_alarm\_subsecond\_config is shown as below:

**Table 3-357. Function rtc\_alarm\_subsecond\_config**

<b>Function name</b>	rtc_alarm_subsecond_config
<b>Function prototype</b>	void rtc_alarm_subsecond_config(uint32_t mask_subsecond, uint32_t subsecond);
<b>Function descriptions</b>	configure subsecond of RTC alarm
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>mask_subsecond</b>	alarm subsecond mask
<i>RTC_MASKSSC_0_14</i>	mask alarm subsecond configuration
<i>RTC_MASKSSC_1_14</i>	mask RTC_ALRM0SS_SSC[14:1], and RTC_ALRM0SS_SSC[0] is to be compared
<i>RTC_MASKSSC_2_14</i>	mask RTC_ALRM0SS_SSC[14:2], and RTC_ALRM0SS_SSC[1:0] is to be compared

<i>RTC_MASKSSC_3_14</i>	mask RTC_ALARM0SS_SSC[14:3], and RTC_ALARM0SS_SSC[2:0] is to be compared
<i>RTC_MASKSSC_4_14</i>	mask RTC_ALARM0SS_SSC[14:4], and RTC_ALARM0SS_SSC[3:0] is to be compared
<i>RTC_MASKSSC_5_14</i>	mask RTC_ALARM0SS_SSC[14:5], and RTC_ALARM0SS_SSC[4:0] is to be compared
<i>RTC_MASKSSC_6_14</i>	mask RTC_ALARM0SS_SSC[14:6], and RTC_ALARM0SS_SSC[5:0] is to be compared
<i>RTC_MASKSSC_7_14</i>	mask RTC_ALARM0SS_SSC[14:7], and RTC_ALARM0SS_SSC[6:0] is to be compared
<i>RTC_MASKSSC_8_14</i>	mask RTC_ALARM0SS_SSC[14:8], and RTC_ALARM0SS_SSC[7:0] is to be compared
<i>RTC_MASKSSC_9_14</i>	mask RTC_ALARM0SS_SSC[14:9], and RTC_ALARM0SS_SSC[8:0] is to be compared
<i>RTC_MASKSSC_10_14</i>	mask RTC_ALARM0SS_SSC[14:10], and RTC_ALARM0SS_SSC[9:0] is to be compared
<i>RTC_MASKSSC_11_14</i>	mask RTC_ALARM0SS_SSC[14:11], and RTC_ALARM0SS_SSC[10:0] is to be compared
<i>RTC_MASKSSC_12_14</i>	mask RTC_ALARM0SS_SSC[14:12], and RTC_ALARM0SS_SSC[11:0] is to be compared
<i>RTC_MASKSSC_13_14</i>	mask RTC_ALARM0SS_SSC[14:13], and RTC_ALARM0SS_SSC[12:0] is to be compared
<i>RTC_MASKSSC_14</i>	mask RTC_ALARM0SS_SSC[14], and RTC_ALARM0SS_SSC[13:0] is to be compared
<i>RTC_MASKSSC_NONE</i>	mask none, and RTC_ALARM0SS_SSC[14:0] is to be compared
<b>Input parameter{in}</b>	
<b>subsecond</b>	alarm subsecond value(0x000 - 0x7FFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*configure subsecond of RTC alarm*/
```

```
rtc_subsecond_config (RTC_MASKSSC_9_14, 0x7FFF);
```

### rtc\_alarm\_get

The description of rtc\_alarm\_get is shown as below:

**Table 3-358. Function rtc\_alarm\_get**

<b>Function name</b>	rtc_alarm_get
----------------------	---------------

<b>Function prototype</b>	void rtc_alarm_get(rtc_alarm_struct* rtc_alarm_time);
<b>Function descriptions</b>	get RTC alarm
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>rtc_alarm_time</b>	pointer to a rtc_alarm_struct structure which contains parameters for RTC alarm configuration, the structure members can refer to members of the structure <a href="#">Table 3-346. Structure rtc_alarm_struct</a> .
<b>Return value</b>	
-	-

Example:

```
rtc_alarm_struct rtc_alarm_time;

rtc_alarm_get (&rtc_alarm_time);
```

### rtc\_alarm\_subsecond\_get

The description of rtc\_alarm\_subsecond\_get is shown as below:

**Table 3-359. Function rtc\_alarm\_subsecond\_get**

<b>Function name</b>	rtc_alarm_subsecond_get
<b>Function prototype</b>	uint32_t rtc_alarm_subsecond_get(void);
<b>Function descriptions</b>	get RTC alarm subsecond
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	RTC alarm subsecond value(0x0-0x7FFF)

Example:

```
/*get RTC alarm subsecond*/

uint32_t subsecond = 0;

subsecond = rtc_alarm_subsecond_get();
```

### rtc\_alarm\_enable

The description of rtc\_alarm\_enable is shown as below:

**Table 3-360. Function rtc\_alarm\_enable**

<b>Function name</b>	rtc_alarm_enable
----------------------	------------------

<b>Function prototype</b>	void rtc_alarm_enable(void);
<b>Function descriptions</b>	enable RTC alarm
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*enable RTC alarm*/
```

```
rtc_alarm_enable();
```

### rtc\_alarm\_disable

The description of rtc\_alarm\_disable is shown as below:

**Table 3-361. Function rtc\_alarm\_disable**

<b>Function name</b>	rtc_alarm_disable
<b>Function prototype</b>	ErrStatus rtc_alarm_disable(void);
<b>Function descriptions</b>	disable RTC alarm
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/*disable RTC alarm*/
```

```
ErrStatus error_status = ERROR;
```

```
error_status = rtc_alarm_disable();
```

### rtc\_timestamp\_enable

The description of rtc\_timestamp\_enable is shown as below:

**Table 3-362. Function rtc\_timestamp\_enable**

<b>Function name</b>	rtc_timestamp_enable
<b>Function prototype</b>	void rtc_timestamp_enable(uint32_t edge);
<b>Function descriptions</b>	enable RTC time-stamp

<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>edge</b>	specify which edge to detect of time-stamp
<i>RTC_TIMESTAMP_RISING_EDGE</i>	rising edge is valid event edge for timestamp event
<i>RTC_TIMESTAMP_FALLING_EDGE</i>	falling edge is valid event edge for timestamp event
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*enable RTC time-stamp*/
```

```
rtc_timestamp_enable (RTC_TIMESTAMP_RISING_EDGE);
```

### rtc\_timestamp\_disable

The description of rtc\_timestamp\_disable is shown as below:

**Table 3-363. Function rtc\_timestamp\_disable**

<b>Function name</b>	rtc_timestamp_disable
<b>Function prototype</b>	void rtc_timestamp_disable(void);
<b>Function descriptions</b>	disable RTC time-stamp
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*disable RTC time-stamp*/
```

```
rtc_timestamp_disable ();
```

### rtc\_timestamp\_get

The description of rtc\_timestamp\_get is shown as below:

**Table 3-364. Function rtc\_timestamp\_get**

<b>Function name</b>	rtc_timestamp_get
<b>Function prototype</b>	void rtc_timestamp_get(rtc_timestamp_struct* rtc_timestamp);



<b>Function descriptions</b>	get RTC timestamp time and date
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
rtc_timestamp	Pointer to a rtc_timestamp_struct structure which contains parameters for RTC time-stamp configuration, the structure members can refer to members of the structure <a href="#">Table 3-348. Structure rtc_tamper_struct.</a>
<b>Return value</b>	
-	-

Example:

```
/* get RTC timestamp time and date */
rtc_timestamp_struct rtc_timestamp;
rtc_timestamp_get(& rtc_timestamp);
```

### rtc\_timestamp\_subsecond\_get

The description of rtc\_timestamp\_subsecond\_get is shown as below:

**Table 3-365. Function rtc\_timestamp\_subsecond\_get**

<b>Function name</b>	rtc_timestamp_subsecond_get
<b>Function prototype</b>	uint32_t rtc_timestamp_subsecond_get(void);
<b>Function descriptions</b>	get RTC time-stamp subsecond
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	RTC time-stamp subsecond value

Example:

```
/* get RTC time-stamp subsecond */
uint32_t subsecond = 0;
subsecond = rtc_timestamp_subsecond_get();
```

### rtc\_tamper\_enable

The description of rtc\_tamper\_enable is shown as below:

Table 3-366. Function rtc\_tamper\_enable

Function name	rtc_tamper_enable
Function prototype	void rtc_tamper_enable(rtc_tamper_struct* rtc_tamper);
Function descriptions	enable RTC tamper
Precondition	-
Input parameter{in}	
rtc_tamper	pointer to a rtc_tamper_struct structure which contains parameters for RTC tamper configuration, the structure members can refer to members of the structure <a href="#">Table 3-348. Structure rtc_tamper_struct</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable RTC tamper */
rtc_tamper_struct rtc_tamper
rtc_tamper_enable(& rtc_tamper);
```

### rtc\_tamper\_disable

The description of rtc\_tamper\_disable is shown as below:

Table 3-367. Function rtc\_tamper\_disable

Function name	rtc_tamper_disable
Function prototype	void rtc_tamper_disable(uint32_t source);
Function descriptions	disable RTC tamper
Precondition	-
Input parameter{in}	
source	specify which tamper source to be disabled
RTC_TAMPER0	RTC tamper0
RTC_TAMPER1	RTC tamper1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable RTC tamper */
rtc_tamper_disable(RTC_TAMPER0);
```

## rtc\_interrupt\_enable

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

**Table 3-368. Function rtc\_interrupt\_enable**

<b>Function name</b>	rtc_interrupt_enable
<b>Function prototype</b>	void rtc_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable specified RTC interrupt
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which interrupt source to be enabled
<i>RTC_INT_TIMESTAMP</i>	timestamp interrupt
<i>RTC_INT_ALARM</i>	alarm interrupt
<i>RTC_INT_TAMP</i>	tamp interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable specified RTC interrupt*/
rtc_interrupt_enable(RTC_INT_TAMP);
```

## rtc\_interrupt\_disable

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

**Table 3-369. Function rtc\_interrupt\_disable**

<b>Function name</b>	rtc_interrupt_disable
<b>Function prototype</b>	void rtc_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disble specified RTC interrupt
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which RTC interrupt to disable
<i>RTC_INT_TIMESTAMP</i>	second interrupt
<i>RTC_INT_ALARM</i>	alarm interrupt
<i>RTC_INT_TAMP</i>	tamp interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable specified RTC interrupt */
```

```
rtc_interrupt_disable(RTC_INT_TAMP);
```

### rtc\_flag\_get

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

**Table 3-370. Function rtc\_flag\_get**

<b>Function name</b>	rtc_flag_get
<b>Function prototype</b>	FlagStatus rtc_flag_get(uint32_t flag);
<b>Function descriptions</b>	check specified flag
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag to check
RTC_FLAG_RECALIBRATION	recalibration pending flag
RTC_FLAG_TAMP1	tamper 1 event flag
RTC_FLAG_TAMP0	tamper 0 event flag
RTC_FLAG_TIMESTAMP_OVERFLOW	time-stamp overflow event flag
RTC_FLAG_TIMESTAMP	time-stamp event flag
RTC_FLAG_ALARM0	alarm event flag
RTC_FLAG_INIT	init mode event flag
RTC_FLAG_RSYN	time and date registers synchronized event flag
RTC_FLAG_YCM	year parameter configured event flag
RTC_FLAG_SHIFT	shift operation pending flag
RTC_FLAG_ALARM0_WRITTEN	alarm written available flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
FlagStatus	SET or RESET

Example:

```
/* check time-stamp event flag */
```

```
FlagStatus flag = RESET;
```

```
flag = rtc_flag_get(RTC_FLAG_TIMESTAMP);
```

### rtc\_flag\_clear

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

Table 3-371. Function `rtc_flag_clear`

<b>Function name</b>	<code>rtc_flag_clear</code>
<b>Function prototype</b>	<code>void rtc_flag_clear(uint32_t flag);</code>
<b>Function descriptions</b>	clear specified flag
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag to clear
<code>RTC_FLAG_TAMP1</code>	tamper 1 event flag
<code>RTC_FLAG_TAMP0</code>	tamper 0 event flag
<code>RTC_FLAG_TIMESTAMP_OVERFLOW</code>	time-stamp overflow event flag
<code>RTC_FLAG_TIMESTAMP_MP</code>	time-stamp event flag
<code>RTC_FLAG_ALARM0</code>	alarm event flag
<code>RTC_FLAG_RSYN</code>	time and date registers synchronized event flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear time-stamp event flag */
```

```
rtc_flag_clear (RTC_FLAG_TIMESTAMP);
```

### `rtc_alter_output_config`

The description of `rtc_alter_output_config` is shown as below:

Table 3-372. Function `rtc_alter_output_config`

<b>Function name</b>	<code>rtc_alter_output_config</code>
<b>Function prototype</b>	<code>void rtc_alter_output_config(uint32_t source, uint32_t mode);</code>
<b>Function descriptions</b>	configure rtc alternate output source
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	specify signal to output
<code>RTC_CALIBRATION_512HZ</code>	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 512Hz signal
<code>RTC_CALIBRATION_1HZ</code>	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 1Hz signal
<code>RTC_ALARM_HIGH</code>	when the alarm flag is set, the output pin is high
<code>RTC_ALARM_LOW</code>	when the Alarm flag is set, the output pin is low
<b>Input parameter{in}</b>	
<b>mode</b>	specify the output pin (PC13) mode when output alarm signal

<i>RTC_ALARM_OUTPU</i> <i>T_OD</i>	open drain mode
<i>RTC_ALARM_OUTPU</i> <i>T_PP</i>	push pull mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure rtc alternate output source */
```

```
rtc_alter_output_config(RTC_ALARM_LOW, RTC_ALARM_OUTPUT_PP);
```

### rtc\_calibration\_config

The description of rtc\_calibration\_config is shown as below:

**Table 3-373. Function rtc\_calibration\_config**

<b>Function name</b>	rtc_calibration_config
<b>Function prototype</b>	ErrStatus rtc_calibration_config(uint32_t window, uint32_t plus, uint32_t minus);
<b>Function descriptions</b>	configure RTC calibration register
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>window</b>	select calibration window
<i>RTC_CALIBRATION_WINDOW_32S</i>	2exp20 RTCCLK cycles, 32s if RTCCLK = 32768 Hz
<i>RTC_CALIBRATION_WINDOW_16S</i>	2exp19 RTCCLK cycles, 16s if RTCCLK = 32768 Hz
<i>RTC_CALIBRATION_WINDOW_8S</i>	2exp18 RTCCLK cycles, 8s if RTCCLK = 32768 Hz
<b>Input parameter{in}</b>	
<b>plus</b>	add RTC clock or not
<i>RTC_CALIBRATION_PLUS_SET</i>	add one RTC clock every 2048 rtc clock
<i>RTC_CALIBRATION_PLUS_RESET</i>	no effect
<b>Input parameter{in}</b>	
<b>minus</b>	the RTC clock to minus during the calibration window(0x0 - 0x1FF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* configure RTC calibration register*/
```

```
ErrStatus error_status = ERROR;
```

```
error_status = rtc_calibration_config(RTC_CALIBRATION_WINDOW_32S,
RTC_CALIBRATION_PLUS_SET, 0x1FF);
```

### rtc\_hour\_adjust

The description of rtc\_hour\_adjust is shown as below:

**Table 3-374. Function rtc\_hour\_adjust**

<b>Function name</b>	rtc_hour_adjust
<b>Function prototype</b>	void rtc_hour_adjust(uint32_t operation);
<b>Function descriptions</b>	adjust the daylight saving time by adding or subtracting one hour from the current time
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>operation</b>	hour adjustment operation
<i>RTC_CTL_A1H</i>	add one hour
<i>RTC_CTL_S1H</i>	subtract one hour
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* adjust the daylight saving time by adding one hour from the current time */
```

```
rtc_hour_adjust(RTC_CTL_A1H);
```

### rtc\_second\_adjust

The description of rtc\_second\_adjust is shown as below:

**Table 3-375. Function rtc\_second\_adjust**

<b>Function name</b>	rtc_second_adjust
<b>Function prototype</b>	ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);
<b>Function descriptions</b>	adjust RTC second or subsecond value of current time
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>add</b>	add 1s to current time or not
<i>RTC_SHIFT_ADD1S_R</i> <i>ESET</i>	no effect

<i>RTC_SHIFT_ADD1S_SET</i>	add 1s to current time
<b>Input parameter{in}</b>	
<b>minus</b>	number of subsecond to minus from current time(0x0 - 0x7FFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* adjust RTC second or subsecond value of current time */
```

```
ErrStatus error_status = ERROR;
```

```
error_status = rtc_second_adjust(RTC_SHIFT_ADD1S_SET, 0);
```

### rtc\_bypass\_shadow\_enable

The description of rtc\_bypass\_shadow\_enable is shown as below:

**Table 3-376. Function rtc\_bypass\_shadow\_enable**

<b>Function name</b>	rtc_bypass_shadow_enable
<b>Function prototype</b>	void rtc_bypass_shadow_enable(void);
<b>Function descriptions</b>	enable RTC bypass shadow registers function
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable RTC bypass shadow registers function*/
```

```
rtc_bypass_shadow_enable();
```

### rtc\_bypass\_shadow\_disable

The description of rtc\_bypass\_shadow\_disable is shown as below:

**Table 3-377. Function rtc\_bypass\_shadow\_disable**

<b>Function name</b>	rtc_bypass_shadow_disable
<b>Function prototype</b>	void rtc_bypass_shadow_disable (void);
<b>Function descriptions</b>	disable RTC bypass shadow registers function
<b>Precondition</b>	-



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

Example:

```
/* disable RTC bypass shadow registers function*/
```

```
rtc_bypass_shadow_disable ();
```

### rtc\_refclock\_detection\_enable

The description of rtc\_refclock\_detection\_enable shown as below:

**Table 3-378. Function rtc\_refclock\_detection\_enable**

Function name	rtc_refclock_detection_enable
Function prototype	ErrStatus rtc_refclock_detection_enable(void);
Function descriptions	enable RTC reference clock detection function
Precondition	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* enable RTC reference clock detection function*/
```

```
ErrStatus error_status = ERROR;
```

```
error_status = rtc_refclock_detection_enable();
```

### rtc\_refclock\_detection\_disable

The description of rtc\_refclock\_detection\_disable shown as below:

**Table 3-379. Function rtc\_refclock\_detection\_disable**

Function name	rtc_refclock_detection_disable
Function prototype	ErrStatus rtc_refclock_detection_disable(void);
Function descriptions	disable RTC reference clock detection function
Precondition	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* disableRTC reference clock detection function*/
```

```
ErrStatus error_status = ERROR;
```

```
error_status = rtc_refclock_detection_disable ();
```

## 3.18. SLCD

The SLCD controller directly drives LCD displays by creating the AC segment and commonvoltage signals automatically. The SLCD registers are listed in chapter [3.18.1](#), the SLCD firmware functions are introduced in chapter [3.18.2](#).

### 3.18.1. Descriptions of Peripheral registers

SLCD registers are listed in the table shown as below:

**Table 3-380. SLCD Registers**

Registers	Descriptions
SLCD_CTL	Control register
SLCD_CFG	Configuration register
SLCD_STAT	Status register
SLCD_STATC	Status flag clear register
SLCD_DATA0	Display data register 0
SLCD_DATA1	Display data register 1
SLCD_DATA2	Display data register 2
SLCD_DATA3	Display data register 3
SLCD_DATA4	Display data register 4
SLCD_DATA5	Display data register 5
SLCD_DATA6	Display data register 6
SLCD_DATA7	Display data register 7

### 3.18.2. Descriptions of Peripheral functions

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

**Table 3-381. SLCD firmware function**

Function name	Function description
slcd_deinit	SLCD reset

Function name	Function description
slcd_enable	enable SLCD interface
slcd_disable	disable SLCD interface
slcd_bias_voltage_select	SLCD bias voltage select
slcd_duty_select	SLCD duty cycle select
slcd_clock_config	config the prescaler and the divider of SLCD clock
slcd_blink_mode_config	SLCD blink mode config
slcd_contrast_ratio_config	SLCD contrast ratio config
slcd_dead_time_config	SLCD dead time duration config
slcd_pulse_on_duration_config	SLCD pulse on duration config
slcd_com_seg_remap	SLCD common/segment pad select
slcd_voltage_source_select	SLCD voltage source select
slcd_high_drive_config	enable or disable permanent high drive
slcd_data_register_write	write SLCD display data registers
slcd_data_update_request	SLCD data update request
slcd_interrupt_config	the SLCD interrupt config
slcd_flag_get	get the SLCD status flag
slcd_flag_clear	clear the SLCD flag
slcd_interrupt_flag_get	get the SLCD interrupt flag
slcd_interrupt_flag_clear	clear the SLCD interrupt flag

## Enum slcd\_data\_register\_enum

**Table 3-382. slcd\_data\_register\_enum**

Member name	Function description
SLCD_DATA_REG0	SLCD display data register 0
SLCD_DATA_REG1	SLCD display data register 1
SLCD_DATA_REG2	SLCD display data register 2
SLCD_DATA_REG3	SLCD display data register 3
SLCD_DATA_REG4	SLCD display data register 4
SLCD_DATA_REG5	SLCD display data register 5
SLCD_DATA_REG6	SLCD display data register 6
SLCD_DATA_REG7	SLCD display data register 7

## slcd\_deinit

The description of slcd\_deinit is shown as below:

**Table 3-383. Function slcd\_deinit**

Function name	slcd_deinit
Function prototype	void slcd_deinit(void);
Function descriptions	SLCD reset
Precondition	-

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

Example:

```
/* reset the SLCD */
```

```
slcd_deinit ( );
```

### slcd\_enable

The description of slcd\_enable is shown as below:

**Table 3-384. Function slcd\_enable**

Function name	slcd_enable
Function prototype	void slcd_enable(void);
Function descriptions	enable SLCD interface
Precondition	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SLCD */
```

```
slcd_enable ( );
```

### slcd\_disable

The description of slcd\_disable is shown as below:

**Table 3-385. Function slcd\_disable**

Function name	slcd_disable
Function prototype	void slcd_disable(void);
Function descriptions	disable SLCD interface
Precondition	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable SLCD */
```

```
slcd_disable ( );
```

### slcd\_bias\_voltage\_select

The description of slcd\_bias\_voltage\_select is shown as below:

**Table 3-386. Function slcd\_bias\_voltage\_select**

<b>Function name</b>	slcd_bias_voltage_select
<b>Function prototype</b>	void slcd_bias_voltage_select(uint32_t bias_voltage);
<b>Function descriptions</b>	SLCD bias voltage select
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>bias_voltage</b>	the SLCD voltage bias
SLCD_BIAS_1_4	1/4 voltage bias
SLCD_BIAS_1_2	1/2 voltage bias
SLCD_BIAS_1_3	1/3 voltage bias
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the SLCD 1/4 bias voltage */
```

```
slcd_bias_voltage_select(SLCD_BIAS_1_4);
```

### slcd\_duty\_select

The description of slcd\_duty\_select is shown as below:

**Table 3-387. Function slcd\_duty\_select**

<b>Function name</b>	slcd_duty_select
<b>Function prototype</b>	void slcd_duty_select(uint32_t duty);
<b>Function descriptions</b>	SLCD duty cycle select
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>duty</b>	duty select
SLCD_DUTY_STATIC	static duty

SLCD_DUTY_1_2	1/2 duty
SLCD_DUTY_1_3	1/3 duty
SLCD_DUTY_1_4	1/4 duty
SLCD_DUTY_1_6	1/6 duty
SLCD_DUTY_1_8	1/8 duty
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SLCD duty cycle select */
```

```
slcd_duty_select (SLCD_DUTY_1_2);
```

### slcd\_clock\_config

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

**Table 3-388. Function slcd\_clock\_config**

<b>Function name</b>	slcd_clock_config
<b>Function prototype</b>	void slcd_clock_config(uint32_t prescaler,uint32_t divider);
<b>Function descriptions</b>	config the prescaler and the divider of SLCD clock
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>prescaler</b>	the prescaler factor
SLCD_PRESCALER_1	$f_{PSC} = f_{in\_clk}$
SLCD_PRESCALER_2	$f_{PSC} = f_{in\_clk}/2$
SLCD_PRESCALER_4	$f_{PSC} = f_{in\_clk}/4$
SLCD_PRESCALER_8	$f_{PSC} = f_{in\_clk}/8$
SLCD_PRESCALER_16	$f_{PSC} = f_{in\_clk}/16$
SLCD_PRESCALER_32	$f_{PSC} = f_{in\_clk}/32$
SLCD_PRESCALER_64	$f_{PSC} = f_{in\_clk}/64$
SLCD_PRESCALER_128	$f_{PSC} = f_{in\_clk}/128$
SLCD_PRESCALER_256	$f_{PSC} = f_{in\_clk}/256$
SLCD_PRESCALER_512	$f_{PSC} = f_{in\_clk}/512$
SLCD_PRESCALER_1024	$f_{PSC} = f_{in\_clk}/1024$

SLCD_PRESCALER_2 048	$f_{PSC} = f_{in\_clk}/2048$
SLCD_PRESCALER_4 096	$f_{PSC} = f_{in\_clk}/4096$
SLCD_PRESCALER_8 192	$f_{PSC} = f_{in\_clk}/8192$
SLCD_PRESCALER_1 6384	$f_{PSC} = f_{in\_clk}/16384$
SLCD_PRESCALER_3 2768	$f_{PSC} = f_{in\_clk}/32768$
<b>Input parameter{in}</b>	
<b>divider</b>	the divider factor
SLCD_DIVIDER_16	$f_{SLCD} = f_{PSC}/16$
SLCD_DIVIDER_17	$f_{SLCD} = f_{PSC}/17$
SLCD_DIVIDER_18	$f_{SLCD} = f_{PSC}/18$
SLCD_DIVIDER_19	$f_{SLCD} = f_{PSC}/19$
SLCD_DIVIDER_20	$f_{SLCD} = f_{PSC}/20$
SLCD_DIVIDER_21	$f_{SLCD} = f_{PSC}/21$
SLCD_DIVIDER_22	$f_{SLCD} = f_{PSC}/22$
SLCD_DIVIDER_23	$f_{SLCD} = f_{PSC}/23$
SLCD_DIVIDER_24	$f_{SLCD} = f_{PSC}/24$
SLCD_DIVIDER_25	$f_{SLCD} = f_{PSC}/25$
SLCD_DIVIDER_26	$f_{SLCD} = f_{PSC}/26$
SLCD_DIVIDER_27	$f_{SLCD} = f_{PSC}/27$
SLCD_DIVIDER_28	$f_{SLCD} = f_{PSC}/28$
SLCD_DIVIDER_29	$f_{SLCD} = f_{PSC}/29$
SLCD_DIVIDER_30	$f_{SLCD} = f_{PSC}/30$
SLCD_DIVIDER_31	$f_{SLCD} = f_{PSC}/31$
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config the prescaler and the divider of SLCD clock */
slcd_clock_config(SLCD_PRESCALER_4, SLCD_DIVIDER_19);
```

### slcd\_blink\_mode\_config

The description of slcd\_blink\_mode\_config is shown as below:

**Table 3-389. Function slcd\_blink\_mode\_config**

<b>Function name</b>	slcd_blink_mode_config
----------------------	------------------------

<b>Function prototype</b>	void slcd_blink_mode_config(uint32_t mode,uint32_t blink_divider);
<b>Function descriptions</b>	SLCD blink mode config
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>mode</b>	blink mode
SLCD_BLINKMODE_OFF	blink disabled
SLCD_BLINKMODE_SEG0_COM0	blink enabled on SEG[0], COM[0]
SLCD_BLINKMODE_SEG0_ALLCOM	blink enabled on SEG[0], all COM
SLCD_BLINKMODE_ALLSEG_ALLCOM	blink enabled on all SEG and all COM
<b>Input parameter{in}</b>	
<b>divider</b>	the divider factor
SLCD_BLINK_FREQUENCY_DIV8	blink frequency = $f_{SLCD}/8$
SLCD_BLINK_FREQUENCY_DIV16	blink frequency = $f_{SLCD}/16$
SLCD_BLINK_FREQUENCY_DIV32	blink frequency = $f_{SLCD}/32$
SLCD_BLINK_FREQUENCY_DIV64	blink frequency = $f_{SLCD}/64$
SLCD_BLINK_FREQUENCY_DIV128	blink frequency = $f_{SLCD}/128$
SLCD_BLINK_FREQUENCY_DIV256	blink frequency = $f_{SLCD}/256$
SLCD_BLINK_FREQUENCY_DIV512	blink frequency = $f_{SLCD}/512$
SLCD_BLINK_FREQUENCY_DIV1024	blink frequency = $f_{SLCD}/1024$
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SLCD blink mode config */
```

```
slcd_blink_mode_config(SLCD_BLINKMODE_SEG0_COM0,
SLCD_BLINK_FREQUENCY_DIV8);
```



## slcd\_contrast\_ratio\_config

The description of slcd\_contrast\_ratio\_config is shown as below:

**Table 3-390. Function slcd\_contrast\_ratio\_config**

<b>Function name</b>	slcd_contrast_ratio_config
<b>Function prototype</b>	void slcd_contrast_ratio_config(uint32_t contrast_ratio);
<b>Function descriptions</b>	SLCD contrast ratio config
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>contrast_ratio</b>	specify the VSLCD voltage
SLCD_CONTRAST_LE VEL_0	maximum SLCD Voltage = 2.60V
SLCD_CONTRAST_LE VEL_1	maximum SLCD Voltage = 2.73V
SLCD_CONTRAST_LE VEL_2	maximum SLCD Voltage = 2.86V
SLCD_CONTRAST_LE VEL_3	maximum SLCD Voltage = 2.99V
SLCD_CONTRAST_LE VEL_4	maximum SLCD Voltage = 3.12V
SLCD_CONTRAST_LE VEL_5	maximum SLCD Voltage = 3.25V
SLCD_CONTRAST_LE VEL_6	maximum SLCD Voltage = 3.38V
SLCD_CONTRAST_LE VEL_7	maximum SLCD Voltage = 3.51V
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SLCD contrast ratio config */
slcd_contrast_ratio_config(SLCD_CONTRAST_LEVEL_0);
```

## slcd\_dead\_time\_config

The description of slcd\_dead\_time\_config is shown as below:

**Table 3-391. Function slcd\_dead\_time\_config**

<b>Function name</b>	slcd_dead_time_config
<b>Function prototype</b>	void slcd_dead_time_config(uint32_t dead_time);

Function descriptions	SLCD dead time duration config
Precondition	-
Input parameter{in}	
dead_time	the length of the dead time between frames
SLCD_DEADTIME_PE RIOD_0	no dead time
SLCD_DEADTIME_PE RIOD_1	1 phase inserted between couple of frame
SLCD_DEADTIME_PE RIOD_2	2 phase inserted between couple of frame
SLCD_DEADTIME_PE RIOD_3	3 phase inserted between couple of frame
SLCD_DEADTIME_PE RIOD_4	4 phase inserted between couple of frame
SLCD_DEADTIME_PE RIOD_5	5 phase inserted between couple of frame
SLCD_DEADTIME_PE RIOD_6	6 phase inserted between couple of frame
SLCD_DEADTIME_PE RIOD_7	7 phase inserted between couple of frame
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* slcd_dead_time_config */
slcd_dead_time_config (SLCD_DEADTIME_PERIOD_1);
```

### slcd\_pulse\_on\_duration\_config

The description of slcd\_pulse\_on\_duration\_config is shown as below:

**Table 3-392. Function slcd\_pulse\_on\_duration\_config**

Function name	slcd_pulse_on_duration_config
Function prototype	void slcd_pulse_on_duration_config(uint32_t duration);
Function descriptions	SLCD pulse on duration config
Precondition	-
Input parameter{in}	
duration	the pulse duration in terms of PSC pulses
SLCD_PULSEON_DURATION_0	pulse on duration = 0
SLCD_PULSEON_DURATION_1	pulse on duration = $1/f_{psc}$

RATION_1	
SLCD_PULSEON_DU RATION_2	pulse on duration = $2/f_{psc}$
SLCD_PULSEON_DU RATION_3	pulse on duration = $3/f_{psc}$
SLCD_PULSEON_DU RATION_4	pulse on duration = $4/f_{psc}$
SLCD_PULSEON_DU RATION_5	pulse on duration = $5/f_{psc}$
SLCD_PULSEON_DU RATION_6	pulse on duration = $6/f_{psc}$
SLCD_PULSEON_DU RATION_7	pulse on duration = $7/f_{psc}$
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SLCD pulse on duration config */
```

```
slcd_pulse_on_duration_config(SLCD_PULSEON_DURATION_7);
```

### slcd\_com\_seg\_remap

The description of slcd\_com\_seg\_remap is shown as below:

**Table 3-393. Function slcd\_com\_seg\_remap**

<b>Function name</b>	slcd_com_seg_remap
<b>Function prototype</b>	void slcd_com_seg_remap(ControlStatus newvalue);
<b>Function descriptions</b>	SLCD common/segment pad select
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>NewValue</b>	ENABLE or DISABLE
ENABLE	LCD_COM[7:4] pad select LCD_SEG[31:28]
DISABLE	LCD_COM[7:4] pad select LCD_COM[7:4]
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SLCD common/segment pad select */
```

```
slcd_com_seg_remap(ENABLE);
```

## slcd\_voltage\_source\_select

The description of slcd\_voltage\_source\_select is shown as below:

**Table 3-394. Function slcd\_voltage\_source\_select**

<b>Function name</b>	slcd_voltage_source_select
<b>Function prototype</b>	void slcd_voltage_source_select(uint8_t voltage_source);
<b>Function descriptions</b>	SLCD voltage source select
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>voltage_source</b>	the SLCD voltage source
SLCD_VOLTAGE_INTERNAL	internal source
SLCD_VOLTAGE_EXTERNAL	external source (VSLCD pin)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SLCD voltage source select */
```

```
slcd_voltage_source_select(SLCD_VOLTAGE_EXTERNAL);
```

## slcd\_high\_drive\_config

The description of slcd\_high\_drive\_config is shown as below:

**Table 3-395. Function slcd\_high\_drive\_config**

<b>Function name</b>	slcd_high_drive_config
<b>Function prototype</b>	void slcd_high_drive_config(ControlStatus newvalue);
<b>Function descriptions</b>	enable or disable permanent high drive
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>NewValue</b>	ENABLE or DISABLE
ENABLE	Permanent high drive enabled
DISABLE	Permanent high drive disabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable permanent high drive */
slcd_high_drive_config(ENABLE);
```

### slcd\_data\_register\_write

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

**Table 3-396. Function slcd\_high\_drive\_config**

<b>Function name</b>	slcd_data_register_write
<b>Function prototype</b>	void slcd_data_register_write(slcd_data_register_enum register_number, uint32_t data);
<b>Function descriptions</b>	write slcd display data registers
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>register_number</b>	refer to <a href="#">Table 3-382. slcd_data_register_enum</a> .
SLCD_DATA_REGx(x=0, 1, ..., 7)	SLCD_DATAx寄存器
<b>Input parameter{in}</b>	
<b>data</b>	the data write to the register
0-0xffffffff	data value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* write slcd display data registers */
slcd_data_register_write (SLCD_DATA_REG0, 0xffff);
```

### slcd\_data\_update\_request

The description of slcd\_data\_update\_request is shown as below:

**Table 3-397. Function slcd\_data\_update\_request**

<b>Function name</b>	slcd_data_update_request
<b>Function prototype</b>	void slcd_data_update_request(void);
<b>Function descriptions</b>	SLCD data update request
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* SLCD data update request */
slcd_data_update_request();
```

### slcd\_interrupt\_config

The description of slcd\_interrupt\_config is shown as below:

**Table 3-398. Function slcd\_interrupt\_config**

Function name	slcd_interrupt_config
Function prototype	void slcd_interrupt_config(uint8_t slcd_interrupt, ControlStatus newvalue);
Function descriptions	the SLCD interrupt config
Precondition	-
Input parameter{in}	
slcd_interrupt	interrupt source
SLCD_INT_SOF	start of frame interrupt
SLCD_INT_UPD	SLCD update done interrupt
Input parameter{in}	
newvalue	ENABLE or DISABLE interrupt
ENABLE	interrupt enabled
DISABLE	interrupt disabled
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* the SLCD interrupt config */
slcd_interrupt_config(SLCD_INT_SOF, ENABLE);
```

### slcd\_flag\_get

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

**Table 3-399. Function slcd\_flag\_get**

Function name	slcd_flag_get
Function prototype	FlagStatus slcd_flag_get(uint8_t slcd_flag);
Function descriptions	get the SLCD status flag
Precondition	-
Input parameter{in}	

<b>slcd_flag</b>	status flag
SLCD_FLAG_ON	SLCD controller on flag
SLCD_FLAG_SOF	start of frame flag
SLCD_FLAG_UPR	SLCD data update request flag
SLCD_FLAG_UPD	update SLCD data done flag
SLCD_FLAG_VRDY	SLCD voltage ready flag
SLCD_FLAG_SYN	SLCD CFG register synchronization flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the SLCD status flag */
```

```
slcd_flag_get(SLCD_FLAG_ON);
```

### slcd\_flag\_clear

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

**Table 3-400. Function slcd\_flag\_clear**

<b>Function name</b>	slcd_flag_clear
<b>Function prototype</b>	void slcd_flag_clear (uint8_t slcd_flag);
<b>Function descriptions</b>	Clear the SLCD status flag
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>slcd_flag</b>	status flag
SLCD_FLAG_SOF	start of frame flag
SLCD_FLAG_UPD	update SLCD data done flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the SLCD status flag */
```

```
slcd_flag_clear(SLCD_FLAG_SOF);
```

### slcd\_interrupt\_flag\_get

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

Table 3-401. Function slcd\_interrupt\_flag\_get

<b>Function name</b>	slcd_interrupt_flag_get
<b>Function prototype</b>	FlagStatus slcd_interrupt_flag_get (uint8_t slcd_interrupt);
<b>Function descriptions</b>	get the SLCD interrupt flag
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>slcd_interrupt</b>	interrupt source
SLCD_INT_FLAG_SOF	start of frame interrupt
SLCD_INT_FLAG_UPD	SLCD update done interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the SLCD interrupt flag */
slcd_interrupt_flag_get(SLCD_INT_FLAG_SOF);
```

### slcd\_interrupt\_flag\_clear

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

Table 3-402. Function slcd\_interrupt\_flag\_clear

<b>Function name</b>	slcd_interrupt_flag_clear
<b>Function prototype</b>	FlagStatus slcd_interrupt_flag_clear (uint8_t slcd_interrupt);
<b>Function descriptions</b>	clear the SLCD interrupt flag
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>slcd_interrupt</b>	interrupt source
SLCD_INT_FLAG_SOF	start of frame interrupt
SLCD_INT_FLAG_UPD	SLCD update done interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the SLCD interrupt flag */
slcd_interrupt_flag_clear(SLCD_INT_FLAG_SOF);
```



## 3.19. SPI/I2S

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 3-403. 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 3-404. 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	pull NSS pin high in software mode
spi_nss_internal_low	pull NSS pin low in software mode
spi_dma_enable	enable SPI DMA function

Function name	Function description
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
i2s_format_error_clear	clear 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_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 3-405. Structure spi\_parameter\_struct**

Member name	Function description
device_mode	SPI master or slave (SPI_MASTER, SPI_SLAVE)
trans_mode	SPI transtype (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	SPI frame size (SPI_FRAME_SIZE_8BIT, SPI_FRAME_SIZE_16BIT)
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 3-406. 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
<b>SPIx</b>	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 3-407. Function spi\_struct\_para\_init**

<b>Function name</b>	spi_struct_para_init
<b>Function prototype</b>	void spi_struct_para_init(spi_parameter_struct* spi_struct);
<b>Function descriptions</b>	Initialize the parameters of SPI struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_struct</b>	SPI init parameter struct, the structure members can refer to <a href="#">Table 3-405. Structure spi parameter struct.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the parameters of SPI */
spi_parameter_struct spi_init_struct;
```

```
spi_struct_para_init(&spi_init_struct);
```

## spi\_init

The description of spi\_init is shown as below:

**Table 3-408. Function spi\_init**

<b>Function name</b>	spi_init
<b>Function prototype</b>	void spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);
<b>Function descriptions</b>	Initialize SPIx peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>spi_struct</b>	SPI parameter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-405. Structure spi_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize SPI0 */

spi_parameter_struct spi_init_struct;

spi_init_struct.trans_mode      = SPI_TRANSMODE_BDTRANSMIT;
spi_init_struct.device_mode     = SPI_MASTER;
spi_init_struct.frame_size     = SPI_FRAME_SIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_CK_PL_HIGH_PH_2EDGE;
spi_init_struct.nss             = SPI_NSS_SOFT;
spi_init_struct.prescale       = SPI_PSC_8;
spi_init_struct.endian         = SPI_ENDIAN_MSB;

spi_init(SPI0, &spi_init_struct);
```

## spi\_enable

The description of spi\_enable is shown as below:

Table 3-409. Function spi\_enable

Function name	spi_enable
Function prototype	void spi_enable(uint32_t spi_periph);
Function descriptions	Enable SPIx
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 */
spi_enable(SPI0);
```

### spi\_disable

The description of spi\_disable is shown as below:

Table 3-410. Function spi\_disable

Function name	spi_disable
Function prototype	void spi_disable(uint32_t spi_periph);
Function descriptions	Disable SPIx
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 */
spi_disable(SPI0);
```

### i2s\_init

The description of i2s\_init is shown as below:

Table 3-411. 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
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2Sx peripheral
<i>SPIx</i>	x=0,2
<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_PHILIPS</i>	I2S philips standard
<i>I2S_STD_MSB</i>	I2S MSB standard
<i>I2S_STD_LSB</i>	I2S LSB standard
<i>I2S_STD_PCMSHORT</i>	I2S PCM short standard
<i>I2S_STD_PCMLONG</i>	I2S PCM long standard
<b>Input parameter{in}</b>	
<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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize I2S0 */
```

```
i2s_init(SPI0, I2S_MODE_MASTERTX, I2S_STD_PHILIPS, I2S_CKPL_LOW);
```

### i2s\_psc\_config

The description of i2s\_psc\_config is shown as below:

Table 3-412. 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
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2Sx peripheral
<i>SPIx</i>	x=0,2
<b>Input parameter{in}</b>	
<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_96K</i>	audio sample rate is 96KHz
<i>I2S_AUDIOSAMPLE_192K</i>	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_ENABL</i> <i>E</i>	I2S master clock output enable
<i>I2S_MCKOUT_DISABL</i> <i>E</i>	I2S master clock output disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure I2S0 prescaler */
```

```
i2s_psc_config(SPI0, I2S_AUDIOSAMPLE_44K, I2S_FRAMEFORMAT_DT16B_CH16B,  
I2S_MCKOUT_DISABLE);
```

### i2s\_enable

The description of i2s\_enable is shown as below:

**Table 3-413. 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>	-
Input parameter{in}	
<b>spi_periph</b>	I2Sx peripheral
<i>SPIx</i>	x=0,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2S0 */
```

```
i2s_enable(SPI0);
```

### i2s\_disable

The description of i2s\_disable is shown as below:

**Table 3-414. Function i2s\_disable**

<b>Function name</b>	i2s_disable
<b>Function prototype</b>	void i2s_disable(uint32_t spi_periph);



<b>Function descriptions</b>	Disable I2Sx
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2Sx peripheral
<b>SPIx</b>	x=0,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2S0 */
i2s_disable(SPI0);
```

### spi\_nss\_output\_enable

The description of spi\_nss\_output\_enable is shown as below:

**Table 3-415. Function spi\_nss\_output\_enable**

<b>Function name</b>	spi_nss_output_enable
<b>Function prototype</b>	void spi_nss_output_enable(uint32_t spi_periph);
<b>Function descriptions</b>	Enable SPIx NSS output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPIx peripheral
<b>SPIx</b>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 NSS output */
spi_nss_output_enable(SPI0);
```

### spi\_nss\_output\_disable

The description of spi\_nss\_output\_disable is shown as below:

**Table 3-416. 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>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPIx peripheral
<b>SPIx</b>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 NSS output */
spi_nss_output_disable(SPI0);
```

### spi\_nss\_internal\_high

The description of spi\_nss\_internal\_high is shown as below:

**Table 3-417. 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>	pull NSS pin high in software mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 NSS pin is pulled high level in software mode */
spi_nss_internal_high(SPI0);
```

### spi\_nss\_internal\_low

The description of spi\_nss\_internal\_low is shown as below:

Table 3-418. Function `spi_nss_internal_low`

Function name	<code>spi_nss_internal_low</code>
Function prototype	<code>void spi_nss_internal_low(uint32_t spi_periph);</code>
Function descriptions	pull NSS pin low in software mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>spi_periph</code>	SPI peripheral
<code>SPIx</code>	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI0 NSS pin is pulled low level in software mode */
```

```
spi_nss_internal_low(SPI0);
```

## `spi_dma_enable`

The description of `spi_dma_enable` is shown as below:

Table 3-419. Function `spi_dma_enable`

Function name	<code>spi_dma_enable</code>
Function prototype	<code>void spi_dma_enable(uint32_t spi_periph, uint8_t dma);</code>
Function descriptions	Enable SPIx DMA function
Precondition	-
The called functions	-
Input parameter{in}	
<code>spi_periph</code>	SPI peripheral
<code>SPIx</code>	x=0,1,2
Input parameter{in}	
<code>dma</code>	SPI DMA mode
<code>SPI_DMA_TRANSMIT</code>	SPI transmit data use DMA
<code>SPI_DMA_RECEIVE</code>	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 3-420. 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>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>dma</b>	SPI DMA mode
<i>SPI_DMA_TRANSMIT</i>	SPI transmit data use DMA
<i>SPI_DMA_RECEIVE</i>	SPI receive data use DMA
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 transmit data DMA function */
```

```
spi_dma_disable(SPI0, SPI_DMA_TRANSMIT);
```

## spi\_i2s\_data\_frame\_format\_config

The description of spi\_i2s\_data\_frame\_format\_config is shown as below:

**Table 3-421. 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_8BIT</i>	SPI frame size is 8 bits
<i>SPI_FRAME_SIZE_16BIT</i>	SPI frame size is 16 bits
<i>T</i>	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SPI0/I2S0 data frame format size is 16 bits */
```

```
spi_i2s_data_frame_format_config(SPI0, SPI_FRAME_SIZE_16BIT);
```

### spi\_bidirectional\_transfer\_config

The description of spi\_bidirectional\_transfer\_config is shown as below:

**Table 3-422. 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>	-
Input parameter{in}	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

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 3-423. 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 3-424. 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);
```

### i2s\_format\_error\_clear

The description of i2s\_format\_error\_clear is shown as below:

**Table 3-425. Function i2s\_format\_error\_clear**

<b>Function name</b>	i2s_format_error_clear
<b>Function prototype</b>	void i2s_format_error_clear(uint32_t spi_periph);
<b>Function descriptions</b>	clear 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=0,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear I2S0 format error flag status */
```

```
i2s_format_error_clear(SPI0);
```

### spi\_crc\_polynomial\_set

The description of spi\_crc\_polynomial\_set is shown as below:

**Table 3-426. Function spi\_crc\_polynomial\_set**

<b>Function name</b>	spi_crc_polynomial_set
<b>Function prototype</b>	void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly);
<b>Function descriptions</b>	Set SPI CRC polynomial
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>crc_poly</b>	CRC polynomial value
<b>Output parameter{out}</b>	
-	-

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 3-427. 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 3-428. Function spi\_crc\_on**

Function name	spi_crc_on
Function prototype	void spi_crc_on(uint32_t spi_periph);
Function descriptions	Turn on CRC function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2



Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn on SPI0 CRC function */
```

```
spi_crc_on(SPI0);
```

### spi\_crc\_off

The description of spi\_crc\_off is shown as below:

**Table 3-429. Function spi\_crc\_off**

Function name	spi_crc_off
Function prototype	void spi_crc_off(uint32_t spi_periph);
Function descriptions	Turn off CRC function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn off SPI0 CRC function */
```

```
spi_crc_off(SPI0);
```

### spi\_crc\_next

The description of spi\_crc\_next is shown as below:

**Table 3-430. Function spi\_crc\_next**

Function name	spi_crc_next
Function prototype	void spi_crc_next(uint32_t spi_periph);
Function descriptions	SPI next data is CRC value
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral

<i>SPIx</i>	<i>x</i> =0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI0 next data is CRC value */
spi_crc_next(SPI0);
```

### spi\_crc\_get

The description of spi\_crc\_get is shown as below:

**Table 3-431. Function spi\_crc\_get**

Function name	spi_crc_get
Function prototype	uint16_t spi_crc_get(uint32_t spi_periph,uint8_t crc);
Function descriptions	Get SPI CRC send value or receive value
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	<i>x</i> =0,1,2
Input parameter{in}	
<i>crc</i>	SPI crc value
<i>SPI_CRC_TX</i>	get transmit crc value
<i>SPI_CRC_RX</i>	get receive crc value
Output parameter{out}	
-	-
Return value	
uint16_t	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 3-432. Function spi\_crc\_error\_clear**

Function name	spi_crc_error_clear
---------------	---------------------

<b>Function prototype</b>	void spi_crc_error_clear(uint32_t spi_periph);
<b>Function descriptions</b>	Clear SPI CRC error flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear SPI0 CRC error flag status */
spi_crc_error_clear(SPI0);
```

### spi\_i2s\_flag\_get

The description of spi\_i2s\_flag\_get is shown as below:

**Table 3-433. 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
<b>SPIx</b>	x=0,1,2
<b>Input parameter{in}</b>	
<b>flag</b>	SPI/I2S flag status
<b>SPI_FLAG_TBE</b>	transmit buffer empty flag
<b>SPI_FLAG_RBNE</b>	receive buffer not empty flag
<b>SPI_FLAG_TRANS</b>	transmit on-going flag
<b>SPI_FLAG_RXORERR</b>	receive overrun error flag
<b>SPI_FLAG_CONFERR</b>	mode config error flag
<b>SPI_FLAG_CRCERR</b>	CRC error flag
<b>SPI_FLAG_FERR</b>	SPI format error interrupt flag
<b>I2S_FLAG_TBE</b>	transmit buffer empty flag
<b>I2S_FLAG_RBNE</b>	receive buffer not empty flag
<b>I2S_FLAG_TRANS</b>	transmit on-going flag
<b>I2S_FLAG_RXORERR</b>	overrun error flag

<i>I2S_FLAG_TXURERR</i>	underrun error flag
<i>I2S_FLAG_CH</i>	channel side flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get SPI0 transmit buffer empty flag status */
while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
```

### spi\_i2s\_interrupt\_enable

The description of spi\_i2s\_interrupt\_enable is shown as below:

**Table 3-434. 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>	-
<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 and transmission underrun 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 3-435. 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 and transmission underrun 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 3-436. 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_T BE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_FLAG_R BNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_FLAG_R XORERR</i>	overrun interrupt
<i>SPI_INT_FLAG_CONF ERR</i>	config error interrupt
<i>SPI_INT_FLAG_CRCE RR</i>	CRC error interrupt
<i>I2S_INT_FLAG_TXUR ERR</i>	underrun 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. SYSCFG

The SYSCFG registers are listed in chapter [3.20.1](#), the SYSCFG firmware functions are introduced in chapter [3.20.2](#).

### 3.20.1. Descriptions of Peripheral registers

SYSCFG registers are listed in the table shown as below:

**Table 3-437. SYSCFG Registers**

Registers	Descriptions
SYSCFG_CFG0	system configuration register 0
SYSCFG_EXTISS0	EXTI sources selection register 0
SYSCFG_EXTISS1	EXTI sources selection register 1
SYSCFG_EXTISS2	EXTI sources selection register 2
SYSCFG_EXTISS3	EXTI sources selection register 3

Registers	Descriptions
SYSCFG_CFG2	system configuration register 2
SYSCFG_CPSCTL	system I/O compensation control register

### 3.20.2. Descriptions of Peripheral functions

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

**Table 3-438. SYSCFG firmware function**

Function name	Function description
syscfg_deinit	deinit syscfg module
syscfg_dma_remap_enable	enable the DMA channels remapping
syscfg_dma_remap_disable	disable the DMA channels remapping
syscfg_high_current_enable	enable PB9 high current capability
syscfg_high_current_disable	disable PB9 high current capability
syscfg_exti_line_config	configure the GPIO pin as EXTI Line
syscfg_lock_config	connect TIMER0/14/15/16 break input to the selected parameter
syscfg_flag_get	check if the specified flag in SYSCFG_CFG2 is set or not
syscfg_flag_clear	clear the flag in SYSCFG_CFG2 by writing 1
syscfg_compensation_config	configure the I/O compensation cell
syscfg_cps_rdy_flag_get	check if the I/O compensation cell ready flag is set or not

#### syscfg\_deinit

The description of syscfg\_deinit is shown as below:

**Table 3-439. Function syscfg\_deinit**

Function name	syscfg_deinit
Function prototype	void syscfg_deinit(void);
Function descriptions	reset the SYSCFG registers
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset SYSCFG registers */
syscfg_deinit();
```

## syscfg\_dma\_remap\_enable

The description of syscfg\_dma\_remap\_enable is shown as below:

**Table 3-440. Function syscfg\_dma\_remap\_enable**

<b>Function name</b>	syscfg_dma_remap_enable
<b>Function prototype</b>	void syscfg_dma_remap_enable (void);
<b>Function descriptions</b>	enable the DMA channels remapping
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_dma_remap</b>	specify the DMA channels to remap
SYSCFG_DMA_REMAP_P_TIMER16	remap TIMER16 channel0 and UP DMA requests to channel1(default channel0)
SYSCFG_DMA_REMAP_P_TIMER15	remap TIMER15 channel2 and UP DMA requests to channel3(default channel2)
SYSCFG_DMA_REMAP_P_USART0RX	remap USART0 Rx DMA request to channel4(default channel2)
SYSCFG_DMA_REMAP_P_USART0TX	remap USART0 Tx DMA request to channel3(default channel1)
SYSCFG_DMA_REMAP_P_ADC	remap ADC DMA requests from channel0 to channel1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMA channel remap*/
```

```
syscfg_dma_remap_enable(SYSCFG_DMA_REMAP_TIMER16);
```

## syscfg\_dma\_remap\_disable

The description of syscfg\_dma\_remap\_disable is shown as below:

**Table 3-441. Function syscfg\_dma\_remap\_disable**

<b>Function name</b>	syscfg_dma_remap_disable
<b>Function prototype</b>	void syscfg_dma_remap_disable (void);
<b>Function descriptions</b>	disable the DMA channels remapping
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_dma_remap</b>	specify the DMA channels to remap



SYSCFG_DMA_REMAP P_TIMER16	remap TIMER16 channel0 and UP DMA requests to channel1(default channel0)
SYSCFG_DMA_REMAP P_TIMER15	remap TIMER15 channel2 and UP DMA requests to channel3(default channel2)
SYSCFG_DMA_REMAP P_USART0RX	remap USART0 Rx DMA request to channel4(default channel2)
SYSCFG_DMA_REMAP P_USART0TX	remap USART0 Tx DMA request to channel3(default channel1)
SYSCFG_DMA_REMAP P_ADC	remap ADC DMA requests from channel0 to channel1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMA channel remap*/
```

```
syscfg_dma_remap_disable(SYSCFG_DMA_REMAP_TIMER16);
```

### syscfg\_high\_current\_enable

The description of syscfg\_high\_current\_enable is shown as below:

**Table 3-442. Function syscfg\_high\_current\_enable**

Function name	syscfg_high_current_enable
Function prototype	void syscfg_high_current_enable(void);
Function descriptions	enable PB9 high current capability
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable PB9 high current capability */
```

```
syscfg_high_current_enable();
```

### syscfg\_high\_current\_disable

The description of syscfg\_high\_current\_disable is shown as below:

Table 3-443. Function syscfg\_high\_current\_disable

Function name	syscfg_high_current_disable
Function prototype	void syscfg_high_current_disable(void);
Function descriptions	disable PB9 high current capability
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable PB9 high current capability */
syscfg_high_current_disable();
```

### syscfg\_exti\_line\_config

The description of syscfg\_exti\_line\_config is shown as below:

Table 3-444. Function syscfg\_exti\_line\_config

Function name	syscfg_exti_line_config
Function prototype	void syscfg_exti_line_config(uint8_t exti_port, uint8_t exti_pin);
Function descriptions	configure the GPIO pin as EXTI Line
Precondition	-
The called functions	-
Input parameter{in}	
exti_port	specify the GPIO port used in EXTI
EXTI_SOURCE_GPIOx	x=A,B,C,D,F
exti_pin	specify the EXTI line
EXTI_SOURCE_PINx	x=0..15(GPIOA, GPIOB, GPIOC, GPIOD, GPIOF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the GPIO pin as EXTI Line */
syscfg_exti_line_config(EXTI_SOURCE_GPIOA, EXTI_SOURCE_PIN0);
```

## syscfg\_lock\_config

The description of syscfg\_lock\_config is shown as below:

**Table 3-445. Function syscfg\_lock\_config**

<b>Function name</b>	syscfg_lock_config
<b>Function prototype</b>	void syscfg_lock_config (uint32_t syscfg_lock);
<b>Function descriptions</b>	connect TIMER0/14/15/16 break input to the selected parameter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_lock</b>	specify the parameter to be connected
SYSCFG_LOCK_LOCKUP	Cortex-M3 lockup output connected to the break input
SYSCFG_LOCK_SRAM_PARITY_ERROR	SRAM_PARITY check error connected to the break input
SYSCFG_LOCK_LVD	LVD interrupt connected to the break input
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure syscfg lock*/
syscfg_lock_config(SYSCFG_LOCK_LOCKUP);
```

## syscfg\_flag\_get

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

**Table 3-446. Function syscfg\_flag\_get**

<b>Function name</b>	syscfg_flag_get
<b>Function prototype</b>	FlagStatus syscfg_flag_get(uint32_t syscfg_flag);
<b>Function descriptions</b>	check if the specified flag in SYSCFG_CFG2 is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_flag</b>	specify the flag in SYSCFG_CFG2 to check
SYSCFG_SRAM_PCE_F	SRAM parity check error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

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

Example:

```
/* get syscfg flag */
```

```
FlagStatus status;
```

```
status = syscfg_flag_get(SYSCFG_SRAM_PCEF);
```

### syscfg\_flag\_clear

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

**Table 3-447. Function syscfg\_flag\_clear**

<b>Function name</b>	syscfg_flag_clear
<b>Function prototype</b>	void syscfg_flag_clear (uint32_t syscfg_flag);
<b>Function descriptions</b>	clear the flag in SYSCFG_CFG2 by writing 1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_flag</b>	specify the flag in SYSCFG_CFG2 to check
<i>SYSCFG_SRAM_PCE</i> <i>F</i>	SRAM parity check error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear syscfg flag */
```

```
syscfg_flag_clear(SYSCFG_SRAM_PCEF);
```

### syscfg\_compensation\_config

The description of syscfg\_compensation\_config is shown as below:

**Table 3-448. Function syscfg\_compensation\_config**

<b>Function name</b>	syscfg_compensation_config
<b>Function prototype</b>	void syscfg_compensation_config(uint32_t syscfg_compensation);
<b>Function descriptions</b>	configure the I/O compensation cell
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_compensation</b>	specifies the I/O compensation cell mode
<i>SYSCFG_COMPENSA</i>	I/O compensation cell is enabled

<i>TION_ENABLE</i>	
<i>SYSCFG_COMPENSATION_DISABLE</i>	I/O compensation cell is disabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the I/O compensation cell */
```

```
syscfg_compensation_config(SYSCFG_COMPENSATION_ENABLE);
```

### syscfg\_cps\_rdy\_flag\_get

The description of syscfg\_cps\_rdy\_flag\_get is shown as below:

**Table 3-449. Function syscfg\_cps\_rdy\_flag\_get**

<b>Function name</b>	syscfg_cps_rdy_flag_get
<b>Function prototype</b>	FlagStatus syscfg_cps_rdy_flag_get(void);
<b>Function descriptions</b>	check if the I/O compensation cell ready flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear syscfg flag */
```

```
FlagStatus ready_flag;
```

```
ready_flag = syscfg_cps_rdy_flag_get();
```

## 3.21. TIMER

The timers have a 16-bit counter that can be used as an unsigned counter and supports both input capture and output compare. Timers (TIMERx) are divided into five sorts: advanced timer (TIMER0), general level0 timer (TIMERx, x=1, 2), general level2 timer (TIMER13), general level2 timer (TIMERx, x=15, 16), Basic timer (TIMER5). The specific functions of different types of timer are different. The TIMER registers are listed in chapter [3.21.1](#), the

TIMER firmware functions are introduced in chapter [3.21.2](#).

### 3.21.1. Descriptions of Peripheral registers

TIMERx registers are listed in the table shown as below:

**Table 3-450. TIMERx Registers**

Registers	Descriptions
TIMER_CTL0 (TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	Control register 0
TIMERx_CTL1 (TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	Control register 1
TIMERx_SMCFG (TIMERx, x=0, 1, 2, 14)	Slave mode configuration register
TIMERx_DMAINTEN (TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	DMA and interrupt enable register
TIMERx_INTF (TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	Interrupt flag register
TIMERx_SWEVG (TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	Software event generation register
TIMERx_CHCTL0 (TIMERx, x=0, 1, 2, 13, 14, 15, 16)	Channel control register 0
TIMERx_CHCTL1 (TIMERx, x=0, 1, 2)	Channel control register 1
TIMERx_CHCTL2 (TIMERx, x=0, 1, 2, 13, 14, 15, 16)	Channel control register 2
TIMERx_CNT (TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	Counter register
TIMERx_PSC (TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	Prescaler register
TIMERx_CAR (TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	Counter auto reload register
TIMERx_CREP	Counter repetition register

Registers	Descriptions
(TIMERx, x=0, 5, 14, 15, 16)	
TIMERx_CH0CV (TIMERx, x=0, 1, 2, 13, 14, 15, 16)	Channel 0 capture/compare value register
TIMERx_CH1CV (TIMERx, x=0, 1, 2, 14)	Channel 1 capture/compare value register
TIMERx_CH2CV (TIMERx, x=0, 1, 2)	Channel 2 capture/compare value register
TIMERx_CH3CV (TIMERx, x=0, 1, 2)	Channel 3 capture/compare value register
TIMERx_IRMP (TIMERx, x=13)	Channel complementary protection register
TIMERx_CCHP (TIMERx, x=0, 1, 2, 14, 15, 16)	TIMER complementary channel protection register
TIMERx_DMACFG (TIMERx, x=0, 1, 2, 14, 15, 16)	DMA configuration register
TIMERx_DMATB (TIMERx, x=0, 1, 2, 14, 15, 16)	DMA transfer buffer register

### 3.21.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

**Table 3-451. TIMERx firmware function**

Function name	Function description
timer_deinit	deinit a timer
timer_struct_para_init	initialize the parameters of TIMER init parameter struct with the default values
timer_init	initialize TIMER counter
timer_enable	enable a timer
timer_disable	disable a timer
timer_auto_reload_shadow_enable	enable the auto reload shadow function
timer_auto_reload_shadow_disable	disable the auto reload shadow function
timer_update_event_enable	enable the update event
timer_update_event_disable	disable the update event
timer_counter_alignment	set TIMER counter alignment mode
timer_counter_up_direction	set TIMER counter up direction
timer_counter_down_direction	set TIMER counter down direction

Function name	Function description
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_ocpre_clear_source_config	configure TIMER OCPRE clear source selection
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
timer_flag_clear	clear TIMER flags
timer_dma_enable	enable the TIMER DMA
timer_dma_disable	disable the TIMER DMA
timer_channel_dma_request_source_select	channel DMA request source selection
timer_dma_transfer_config	configure the TIMER DMA transfer
timer_event_software_generate	software generate events
timer_break_struct_para_init	initialize the parameters of TIMER break parameter struct with the default values
timer_break_config	configure TIMER break function
timer_break_enable	enable TIMER break function
timer_break_disable	disable TIMER break function
timer_automatic_output_enable	enable TIMER output automatic function
timer_automatic_output_disable	disable TIMER output automatic function
timer_primary_output_config	configure TIMER primary output function
timer_channel_control_shadow_config	channel capture/compare control shadow register enable
timer_channel_control_shadow_update_config	configure TIMER channel control shadow register update control
timer_channel_output_struct_para_init	initialize the parameters of TIMER channel output parameter struct with the default values
timer_channel_output_config	configure TIMER channel output function
timer_channel_output_mode_config	configure TIMER channel output compare mode
timer_channel_output_pulse_value_config	configure TIMER channel output pulse value
timer_channel_output_shadow_config	configure TIMER channel output shadow function



Function name	Function description
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_channel_remap_config	configure TIMER channel remap function
timer_write_chxval_register_config	configure TIMER write CHxVAL register selection
timer_output_value_selection_config	configure TIMER output value selection

### Structure timer\_parameter\_struct

**Table 3-452. 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 3-453. 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 3-454. Structure timer\_oc\_parameter\_struct**

Member name	Function description
outputstate	channel output state (TIMER_CCX_ENABLE, TIMER_CCX_DISABLE)
outputnstate	channel complementary output state (TIMER_CCXN_ENABLE, TIMER_CCXN_DISABLE)
ocpolarity	channel output polarity (TIMER_OC_POLARITY_HIGH, TIMER_OC_POLARITY_LOW)
ocnpolarity	channel complementary output polarity (TIMER_OCN_POLARITY_HIGH, TIMER_OCN_POLARITY_LOW)

Member name	Function description
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 3-455. 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 3-456. Function timer\_deinit**

Function name	timer_deinit
Function prototype	void timer_deinit(uint32_t timer_periph);
Function descriptions	deinit a TIMER
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..2, 5, 13..16)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset TIMER0 */
timer_deinit (TIMER0);
```

### timer\_struct\_para\_init

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

Table 3-457. 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 3-452. 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 3-458. 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..2, 5, 13..16)	TIMER peripheral selection
Input parameter{in}	
initpara	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-452. Structure timer_parameter_struct.</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize TIMERO */

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 3-459. 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..2, 5, 13..16)</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 3-460. Function timer\_disable**

<b>Function name</b>	timer_disable
<b>Function prototype</b>	void timer_disable(uint32_t timer_periph);

<b>Function descriptions</b>	disable a timer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i> ( <i>x</i> =0..2, 5, 13..16)	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 */
timer_disable (TIMER0);
```

### timer\_auto\_reload\_shadow\_enable

The description of timer\_auto\_reload\_shadow\_enable is shown as below:

**Table 3-461. Function timer\_auto\_reload\_shadow\_enable**

<b>Function name</b>	timer_auto_reload_shadow_enable
<b>Function prototype</b>	void timer_auto_reload_shadow_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable the auto reload shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i> ( <i>x</i> =0..2, 5, 13..16)	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the TIMER0 auto reload shadow function */
timer_auto_reload_shadow_enable (TIMER0);
```

### timer\_auto\_reload\_shadow\_disable

The description of timer\_auto\_reload\_shadow\_disable is shown as below:

Table 3-462. Function timer\_auto\_reload\_shadow\_disable

<b>Function name</b>	timer_auto_reload_shadow_disable
<b>Function prototype</b>	void timer_auto_reload_shadow_disable (uint32_t timer_periph);
<b>Function descriptions</b>	disable the auto reload shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_disable (TIMER0);
```

### timer\_update\_event\_enable

The description of timer\_update\_event\_enable is shown as below:

Table 3-463. Function timer\_update\_event\_enable

<b>Function name</b>	timer_update_event_enable
<b>Function prototype</b>	void timer_update_event_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable the update event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 the update event */
```

```
timer_update_event_enable (TIMER0);
```

## timer\_update\_event\_disable

The description of timer\_update\_event\_disable is shown as below:

**Table 3-464. Function timer\_update\_event\_disable**

<b>Function name</b>	timer_update_event_disable
<b>Function prototype</b>	void timer_update_event_disable (uint32_t timer_periph);
<b>Function descriptions</b>	disable the update event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 the update event */
timer_update_event_disable (TIMER0);
```

## timer\_counter\_alignment

The description of timer\_counter\_alignment is shown as below:

**Table 3-465. 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..2)</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_CENTER_UP</i>	Center-aligned and counting up assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in TIMEx_CHCTL0 register). Only when the counter is counting up, compare interrupt flag of channels can be set.
<i>TIMER_COUNTER_CENTER_BOTH</i>	Center-aligned and counting up/down assert mode. The counter counts under center-aligned and channel is configured in output mode (CHxMS=00 in TIMEx_CHCTL0 register). Both when the counter is counting up and counting down, compare interrupt flag of channels can be set.
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set TIMER0 counter center-aligned and counting up assert mode */
timer_counter_alignment (TIMER0, TIMER_COUNTER_CENTER_UP);
```

### timer\_counter\_up\_direction

The description of timer\_counter\_up\_direction is shown as below:

**Table 3-466. Function timer\_counter\_up\_direction**

<b>Function name</b>	timer_counter_up_direction
<b>Function prototype</b>	void timer_counter_up_direction(uint32_t timer_periph);
<b>Function descriptions</b>	set TIMER counter up direction
<b>Precondition</b>	set TIMER counter no center-aligned mode (edge-aligned mode)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set TIMER0 counter up direction */
timer_counter_up_direction (TIMER0);
```

### timer\_counter\_down\_direction

The description of timer\_counter\_down\_direction is shown as below:

Table 3-467. Function timer\_counter\_down\_direction

<b>Function name</b>	timer_counter_down_direction
<b>Function prototype</b>	void timer_counter_down_direction(uint32_t timer_periph);
<b>Function descriptions</b>	set TIMER counter down direction
<b>Precondition</b>	set TIMER counter no center-aligned mode (edge-aligned mode)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set TIMER0 counter down direction */
```

```
timer_counter_down_direction (TIMER0);
```

### timer\_prescaler\_config

The description of timer\_prescaler\_config is shown as below:

Table 3-468. Function timer\_prescaler\_config

<b>Function name</b>	timer_prescaler_config
<b>Function prototype</b>	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint8_t pscreload);
<b>Function descriptions</b>	configure TIMER prescaler
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>prescaler</b>	prescaler value (0~65535)
<b>Input parameter{in}</b>	
<b>pscreload</b>	prescaler reload mode
<i>TIMER_PSC_RELOAD_NOW</i>	the prescaler is loaded right now
<i>TIMER_PSC_RELOAD_UPDATE</i>	the prescaler is loaded at the next update event
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* configure TIMER0 prescaler */
```

```
timer_prescaler_config (TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

### timer\_repetition\_value\_config

The description of timer\_repetition\_value\_config is shown as below:

**Table 3-469. Function timer\_repetition\_value\_config**

Function name	timer_repetition_value_config
Function prototype	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
Function descriptions	configure TIMER repetition register value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
Input parameter{in}	
repetition	the counter repetition value (0~255)
Output parameter{out}	
-	-
Return value	
-	-

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 3-470. Function timer\_autoreload\_value\_config**

Function name	timer_autoreload_value_config
Function prototype	void timer_autoreload_value_config(uint32_t timer_periph, uint16_t autoreload);
Function descriptions	configure TIMER autoreload register value
Precondition	-
The called functions	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i> ( <i>x</i> =0..2, 5, 13..16)	TIMER peripheral selection
Input parameter{in}	
<b>autoreload</b>	the counter auto-reload value (0-65535)
Output parameter{out}	
-	-
Return value	
-	-

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 3-471. Function timer\_counter\_value\_config**

<b>Function name</b>	timer_counter_value_config
<b>Function prototype</b>	void timer_counter_value_config(uint32_t timer_periph, uint16_t counter);
<b>Function descriptions</b>	configure TIMER counter register value
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i> ( <i>x</i> =0..2, 5, 13..16)	TIMER peripheral selection
Input parameter{in}	
<b>counter</b>	the counter value (0-65535)
Output parameter{out}	
-	-
Return value	
-	-

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 3-472. Function timer\_counter\_read

Function name	timer_counter_read
Function prototype	uint32_t timer_counter_read(uint32_t timer_periph);
Function descriptions	read TIMER counter value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..2, 5, 13..16)	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 3-473. 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..2, 5, 13..16)	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 3-474. Function timer\_single\_pulse\_mode\_config**

<b>Function name</b>	timer_single_pulse_mode_config
<b>Function prototype</b>	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);
<b>Function descriptions</b>	configure TIMER single pulse mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 5, 14..16)</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 3-475. 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</i> ( <i>x</i> =0..2, 5, 13..16)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>update</b>	update source
<i>TIMER_UPDATE_SRC_GLOBAL</i>	Any of the following events generate an update interrupt or DMA request: <ul style="list-style-type: none"> <li>– The UPG bit is set</li> <li>– The counter generates an overflow or underflow event</li> <li>– The slave mode controller generates an update event</li> </ul>
<i>TIMER_UPDATE_SRC_REGULAR</i>	Only counter overflow/underflow generates an update interrupt or DMA request.
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER update only by counter overflow/underflow */
```

```
timer_update_source_config (TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

### timer\_ocpre\_clear\_source\_config

The description of timer\_ocpre\_clear\_source\_config is shown as below:

**Table 3-476. Function t timer\_ocpre\_clear\_source\_config**

<b>Function name</b>	timer_ocpre_clear_source_config
<b>Function prototype</b>	void timer_ocpre_clear_source_config (uint32_t timer_periph, uint8_t ocpreclear);
<b>Function descriptions</b>	configure TIMER OCPRE clear source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i> ( <i>x</i> =0..2)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>ocpreclear</b>	clear source
<i>TIMER_OCPRE_CLEAR_SOURCE_CLR</i>	OCPRE_CLR_INT is connected to the OCPRE_CLR input
<i>TIMER_OCPRE_CLEAR_SOURCE_ETIF</i>	OCPRE_CLR_INT is connected to ETIF

Output parameter{out}	
-	-
Return value	
-	-

例如:

```
/* configure TIMER0 OCPRE_CLR_INT is connected to the OCPRE_CLR input */
timer_ocpre_clear_source_config(TIMER0, TIMER_OCPRE_CLEAR_SOURCE_CLR);
```

### timer\_interrupt\_enable

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

**Table 3-477. Function timer\_interrupt\_enable**

Function name	timer_interrupt_enable
Function prototype	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
Function descriptions	enable the TIMER interrupt
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
interrupt	timer interrupt enable source
<i>TIMER_INT_UP</i>	update interrupt enable, TIMERx (x=0..2, 5, 13..16)
<i>TIMER_INT_CH0</i>	channel 0 interrupt enable, TIMERx(x=0..2, 13..16)
<i>TIMER_INT_CH1</i>	channel 1 interrupt enable, TIMERx(x=0..2, 14)
<i>TIMER_INT_CH2</i>	channel 2 interrupt enable, TIMERx(x=0..2)
<i>TIMER_INT_CH3</i>	channel 3 interrupt enable , TIMERx(x=0..2)
<i>TIMER_INT_CMT</i>	commutation interrupt enable, TIMERx (x=0, 14..16)
<i>TIMER_INT_TRG</i>	trigger interrupt enable, TIMERx(x=0..2, 14)
<i>TIMER_INT_BRK</i>	break interrupt enable, TIMERx (x=0, 14..16)
Output parameter{out}	
-	-
Return value	
-	-

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 3-478. Function timer\_interrupt\_disable**

<b>Function name</b>	timer_interrupt_disable
<b>Function prototype</b>	void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	disable the TIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	timer interrupt disable source
<i>TIMER_INT_UP</i>	update interrupt disable, TIMERx (x=0..2, 5, 13..16)
<i>TIMER_INT_CH0</i>	channel 0 interrupt disable, TIMERx(x=0..2, 13..16)
<i>TIMER_INT_CH1</i>	channel 1 interrupt disable, TIMERx(x=0..2, 14)
<i>TIMER_INT_CH2</i>	channel 2 interrupt disable, TIMERx(x=0..2)
<i>TIMER_INT_CH3</i>	channel 3 interrupt disable, TIMERx(x=0..2)
<i>TIMER_INT_CMT</i>	commutation interrupt disable, TIMERx (x=0, 14..16)
<i>TIMER_INT_TRG</i>	trigger interrupt disable, TIMERx(x=0..2, 14)
<i>TIMER_INT_BRK</i>	break interrupt disable, TIMERx(x=0, 14..16)
<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 3-479. 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>	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>interrupt</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..2, 5, 13..16)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..2, 13..16)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, <i>TIMERx</i> <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, <i>TIMERx</i> <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_INT_FLAG_CMT</i>	channel commutation interrupt flag, <i>TIMERx</i> ( <i>x</i> =0, 14..16)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, <i>TIMERx</i> ( <i>x</i> =0, 14..16)
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TIMERO update interrupt flag */
```

```
FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = timer_interrupt_flag_get (TIMERO, TIMER_INT_FLAG_UP);
```

### timer\_interrupt\_flag\_clear

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

**Table 3-480. Function timer\_interrupt\_flag\_clear**

<b>Function name</b>	timer_interrupt_flag_clear
<b>Function prototype</b>	void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	clear TIMER interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>interrupt</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..2, 5, 13..16)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..2, 13..16)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..2, 14)

<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, <i>TIMERx</i> <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, <i>TIMERx</i> <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_INT_FLAG_CMT</i>	channel commutation interrupt flag, <i>TIMERx</i> ( <i>x</i> =0, 14..16)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, <i>TIMERx</i> ( <i>x</i> =0, 14..16)
<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 3-481. Function `timer_flag_get`**

<b>Function name</b>	<code>timer_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);</code>
<b>Function descriptions</b>	get <i>TIMER</i> flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	<i>TIMER</i> peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>flag</b>	the <i>timer</i> interrupt flags
<i>TIMER_FLAG_UP</i>	update flag, <i>TIMERx</i> ( <i>x</i> =0.. 2, 5, 13..16)
<i>TIMER_FLAG_CH0</i>	channel 0 flag, <i>TIMERx</i> ( <i>x</i> =0..2, 13..16)
<i>TIMER_FLAG_CH1</i>	channel 1 flag, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_FLAG_CH2</i>	channel 2 flag, <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_FLAG_CH3</i>	channel 3 flag, <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_FLAG_CMT</i>	channel commutation flag, <i>TIMERx</i> ( <i>x</i> =0, 14..16)
<i>TIMER_FLAG_TRG</i>	trigger flag, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_FLAG_BRK</i>	break flag, <i>TIMERx</i> ( <i>x</i> =0, 14..16)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..2, 3..16)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..2)

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

Example:

```
/* get TIMER0 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get (TIMER0, TIMER_FLAG_UP);
```

### timer\_flag\_clear

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

**Table 3-482. Function timer\_flag\_clear**

Function name	timer_flag_clear
Function prototype	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
Function descriptions	clear TIMER flags
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
flag	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag, <i>TIMERx</i> ( <i>x</i> =0..2, 5, 13..16)
<i>TIMER_FLAG_CH0</i>	channel 0 flag, <i>TIMERx</i> ( <i>x</i> =0..2, 13..16)
<i>TIMER_FLAG_CH1</i>	channel 1 flag, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_FLAG_CH2</i>	channel 2 flag, <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_FLAG_CH3</i>	channel 3 flag, <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_FLAG_CMT</i>	channel commutation flag, <i>TIMERx</i> ( <i>x</i> =0, 14..16)
<i>TIMER_FLAG_TRG</i>	trigger flag, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_FLAG_BRK</i>	break flag, <i>TIMERx</i> ( <i>x</i> =0, 14..16)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..2, 13..16)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear TIMER0 update flags */
```

```
timer_flag_clear (TIMER0, TIMER_FLAG_UP);
```

### timer\_dma\_enable

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

**Table 3-483. Function timer\_dma\_enable**

<b>Function name</b>	timer_dma_enable
<b>Function prototype</b>	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
<b>Function descriptions</b>	enable the TIMER DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>dma</b>	timer DMA source enable
<i>TIMER_DMA_UPD</i>	update DMA enable, TIMERx(x=0..2, 5, 14..16)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable, TIMERx(x=0..2, 14..16)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable, TIMERx(x=0..2, 4)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA enable, TIMERx(x=0..2)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA enable, TIMERx(x=0..2)
<i>TIMER_DMA_CMTD</i>	commutation DMA request enable, TIMERx(x=0, 14)
<i>TIMER_DMA_TRGD</i>	trigger DMA enable, TIMERx(x=0..2, 14)
<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 3-484. 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, <i>TIMERx</i> (x=0..2, 5, 14..16)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable, <i>TIMERx</i> (x=0..2, 14..16)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable, <i>TIMERx</i> (x=0..2, 14)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA enable, <i>TIMERx</i> (x=0..2)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA enable, <i>TIMERx</i> (x=0..2)
<i>TIMER_DMA_CMTD</i>	commutation DMA request enable, <i>TIMERx</i> (x=0, 14)
<i>TIMER_DMA_TRGD</i>	trigger DMA enable, <i>TIMERx</i> (x=0..2, 14)
<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 3-485. Function timer\_channel\_dma\_request\_source\_select**

<b>Function name</b>	timer_channel_dma_request_source_select
<b>Function prototype</b>	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
<b>Function descriptions</b>	channel DMA request source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i> (x=0..2, 14,..16)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>dma_request</b>	channel DMA request source selection
<i>TIMER_DMAREQUEST_CHANNELEVENT</i>	DMA request of channel n is sent when channel y event occurs
<i>TIMER_DMAREQUEST_UPDATEEVENT</i>	DMA request of channel n is sent when update event occurs

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* TIMER0 channel DMA request of channel n is sent when channel y event occurs */
```

```
timer_channel_dma_request_source_select(TIMER0,  
TIMER_DMAREQUEST_CHANNELEVENT);
```

### timer\_dma\_transfer\_config

The description of timer\_dma\_transfer\_config is shown as below:

**Table 3-486. Function timer\_dma\_transfer\_config**

Function name	timer_dma_transfer_config
Function prototype	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
Function descriptions	configure the TIMER DMA transfer
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..2, 14,..16)	TIMER peripheral selection
Input parameter{in}	
dma_baseaddr	DMA transfer access start address
TIMER_DMACFG_DMA TA_CTL0	DMA transfer address is TIMER_CTL0, TIMERx(x=0..2, 14..16)
TIMER_DMACFG_DMA TA_CTL1	DMA transfer address is TIMER_CTL1, TIMERx(x=0..2, 14..16)
TIMER_DMACFG_DMA TA_SMCFG	DMA transfer address is TIMER_SMCFG, TIMERx(x=0..2, 14)
TIMER_DMACFG_DMA TA_DMAINTEN	DMA transfer address is TIMER_DMAINTEN, TIMERx(x=0..2, 14..16)
TIMER_DMACFG_DMA TA_INTF	DMA transfer address is TIMER_INTF, TIMERx(x=0..2, 14..16)
TIMER_DMACFG_DMA TA_SWEVG	DMA transfer address is TIMER_SWEVG, TIMERx(x=0..2, 14..16)
TIMER_DMACFG_DMA TA_CHCTL0	DMA transfer address is TIMER_CHCTL0, TIMERx(x=0..2, 14..16)
TIMER_DMACFG_DMA TA_CHCTL1	DMA transfer address is TIMER_CHCTL1, TIMERx(x=0..2)
TIMER_DMACFG_DMA	DMA transfer address is TIMER_CHCTL2, TIMERx (x=0..2, 14..16)

TA_CHCTL2	
TIMER_DMACFG_DMA TA_CNT	DMA transfer address is TIMER_CNT, TIMERx (x=0..2, 14..16)
TIMER_DMACFG_DMA TA_PSC	DMA transfer address is TIMER_PSC, TIMERx (x=0..2, 14..16)
TIMER_DMACFG_DMA TA_CAR	MA transfer address is TIMER_CAR, TIMERx (x=0..2, 14..16)
TIMER_DMACFG_DMA TA_CREP	DMA transfer address is TIMER_CREP, TIMERx (x=0, 14..16)
TIMER_DMACFG_DMA TA_CH0CV	DMA transfer address is TIMER_CH0CV, TIMERx (x=0..2, 14..16)
TIMER_DMACFG_DMA TA_CH1CV	DMA transfer address is TIMER_CH1CV, TIMERx(x=0..2, 14)
TIMER_DMACFG_DMA TA_CH2CV	DMA transfer address is TIMER_CH2CV, TIMERx(x=0..2)
TIMER_DMACFG_DMA TA_CH3CV	DMA transfer address is TIMER_CH3CV, TIMERx(x=0..2)
TIMER_DMACFG_DMA TA_CCHP	DMA transfer address is TIMER_CCHP, TIMERx (x=0, 14..16)
TIMER_DMACFG_DMA TA_DMACFG	DMA transfer address is TIMER_DMACFG, TIMERx (x=0..2, 14..16)
<b>Input parameter{in}</b>	
dma_lenth	DMA transfer count
TIMER_DMACFG_DMA TC_xTRANSFER	x=1..18, DMA transfer x time
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the TIMER0 DMA transfer */
```

```
timer_dma_transfer_config(TIMER0, TIMER_DMACFG_DMATA_CTL0,  
TIMER_DMACFG_DMATC_5TRANSFER);
```

### timer\_event\_software\_generate

The description of timer\_event\_software\_generate is shown as below:

**Table 3-487. Function timer\_event\_software\_generate**

<b>Function name</b>	timer_event_software_generate
<b>Function prototype</b>	void timer_event_software_generate(uint32_t timer_periph, uint16_t event);



<b>Function descriptions</b>	software generate events
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>event</b>	the timer software event generation sources
<i>TIMER_EVENT_SRC_UPG</i>	update event, <i>TIMERx</i> ( <i>x</i> =0..2, 5, 13..16)
<i>TIMER_EVENT_SRC_CH0G</i>	channel 0 capture or compare event generation, <i>TIMERx</i> ( <i>x</i> =0..2, 13..16)
<i>TIMER_EVENT_SRC_CH1G</i>	channel 1 capture or compare event generation, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_EVENT_SRC_CH2G</i>	channel 2 capture or compare event generation, <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_EVENT_SRC_CH3G</i>	channel 3 capture or compare event generation, <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_EVENT_SRC_CHMTG</i>	channel commutation event generation, <i>TIMERx</i> ( <i>x</i> =0, 14..16)
<i>TIMER_EVENT_SRC_TRGG</i>	trigger event generation, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_EVENT_SRC_BRKG</i>	break event generation, <i>TIMERx</i> ( <i>x</i> =0, 14..16)
<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 3-488. 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

Precondition	-
The called functions	-
Input parameter{in}	
breakpara	TIMER break parameter struct, the structure members can refer to <a href="#">Table 3-453. Structure timer break parameter struct.</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER break parameter struct with a default value */

timer_break_parameter_struct timer_breakpara;

timer_break_struct_para_init(timer_breakpara);
```

### timer\_break\_config

The description of timer\_break\_config is shown as below:

**Table 3-489. Function timer\_break\_config**

Function name	timer_break_config
Function prototype	void timer_break_config(uint32_t timer_periph, timer_break_parameter_struct* breakpara);
Function descriptions	configure TIMER break function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0, 14..16)	TIMER peripheral selection
Input parameter{in}	
breakpara	TIMER break parameter struct, the structure members can refer to <a href="#">Table 3-453. Structure timer break parameter struct.</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 break function */

timer_break_parameter_struct timer_breakpara;

timer_breakpara.runoffstate = TIMER_ROS_STATE_DISABLE;
```

```

timer_breakpara.ideloffstate    = TIMER_IOS_STATE_DISABLE ;

timer_breakpara.deadtime       = 255;

timer_breakpara.breakpolarity  = TIMER_BREAK_POLARITY_LOW;

timer_breakpara.outputautostate = TIMER_OUTAUTO_ENABLE;

timer_breakpara.protectmode     = TIMER_CCHP_PROT_0;

timer_breakpara.breakstate     = TIMER_BREAK_ENABLE;

timer_break_config(TIMER0, &timer_breakpara);

```

### timer\_break\_enable

The description of timer\_break\_enable is shown as below:

**Table 3-490. 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, 14..16)</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 3-491. 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, 14..16)</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 3-492. 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-field in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 output automatic function */
timer_automatic_output_enable (TIMER0);
```

### timer\_automatic\_output\_disable

The description of timer\_automatic\_output\_disable is shown as below:

**Table 3-493. 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, 14..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 output automatic function */
```

```
timer_automatic_output_disable (TIMER0);
```

### timer\_primary\_output\_config

The description of timer\_primary\_output\_config is shown as below:

**Table 3-494. 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 3-495. 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, 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:

```
/* 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 3-496. 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, 14..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>ccuctl</b>	channel control shadow register update control

<i>TIMER_UPDATECTL_CCU</i>	the shadow registers update by when CMTG bit is set
<i>TIMER_UPDATECTL_CUTRI</i>	the shadow registers update by when CMTG bit is set or an rising edge of TRGI occurs
<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 3-497. 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 3-454. 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 3-498. Function timer\_channel\_output\_config**

<b>Function name</b>	timer_channel_output_config
<b>Function prototype</b>	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpa);
<b>Function descriptions</b>	configure TIMER channel output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..2))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..2))
<b>Input parameter{in}</b>	
<b>ocpara</b>	TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-454. 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 3-499. 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..2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx (x=0..2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx (x=0..2))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx (x=0..2))
<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 3-500. Function timer\_channel\_output\_pulse\_value\_config**

<b>Function name</b>	timer_channel_output_pulse_value_config
<b>Function prototype</b>	void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse);
<b>Function descriptions</b>	configure TIMER channel output pulse value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx (x=0..2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx (x=0..2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx (x=0..2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx (x=0..2))
<b>Input parameter{in}</b>	
<b>pulse</b>	channel output pulse value (0~65535)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 output pulse value */
timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

## timer\_channel\_output\_shadow\_config

The description of timer\_channel\_output\_shadow\_config is shown as below:

**Table 3-501. Function timer\_channel\_output\_shadow\_config**

<b>Function name</b>	timer_channel_output_shadow_config
<b>Function prototype</b>	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
<b>Function descriptions</b>	configure TIMER channel output shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx (x=0..2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx (x=0..2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx (x=0..2))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx (x=0..2))
<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 3-502. 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..2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx (x=0..2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx (x=0..2))

<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx (x=0..2))
<b>Input parameter{in}</b>	
<b>ocfast</b>	channel output fast function
<i>TIMER_OC_FAST_ENABLE</i>	channel output fast function enable
<i>TIMER_OC_FAST_DISABLE</i>	channel output fast function disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 output fast function */
```

```
timer_channel_output_fast_config (TIMER0, TIMER_CH_0, TIMER_OC_FAST_ENABLE);
```

### timer\_channel\_output\_clear\_config

The description of timer\_channel\_output\_clear\_config is shown as below:

**Table 3-503. 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..2))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx (x=0..2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx (x=0..2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx (x=0..2))
<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>	

-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output clear function */
```

```
timer_channel_output_clear_config (TIMER0, TIMER_CH_0,  
TIMER_OC_CLEAR_ENABLE);
```

### timer\_channel\_output\_polarity\_config

The description of timer\_channel\_output\_polarity\_config is shown as below:

**Table 3-504. 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..2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx (x=0..2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..2))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx (x=0..2))
<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 3-505. Function timer\_channel\_complementary\_output\_polarity\_config**

<b>Function name</b>	timer_channel_complementary_output_polarity_config
<b>Function prototype</b>	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnpolarity);
<b>Function descriptions</b>	configure TIMER channel complementary output polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx (x=0..2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx (x=0..2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx (x=0..2))
<i>TIMER_CH_3</i>	TIMER channel 3(TIMERx (x=0..2))
<b>Input parameter{in}</b>	
<b>ocpolarity</b>	channel complementary output polarity
<i>TIMER_OCN_POLARITY_HIGH</i>	channel complementary output polarity is high
<i>TIMER_OCN_POLARITY_LOW</i>	channel complementary output polarity is low
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 complementary output polarity */
```

```
timer_channel_complementary_output_polarity_config (TIMER0, TIMER_CH_0,
TIMER_OCN_POLARITY_HIGH);
```

### timer\_channel\_output\_state\_config

The description of timer\_channel\_output\_state\_config is shown as below:

Table 3-506. Function timer\_channel\_output\_state\_config

<b>Function name</b>	timer_channel_output_state_config
<b>Function prototype</b>	void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
<b>Function descriptions</b>	configure TIMER channel enable state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx (x=0..2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx (x=0..2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx (x=0..2))
<i>TIMER_CH_3</i>	TIMER channel 3(TIMERx (x=0..2))
<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 3-507. 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</i> ( <i>x</i> =0, 14,..16)	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, 14..16)
<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_DISABLE</i>	channel complementary disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 complementary output enable state */
```

```
timer_channel_complementary_output_state_config (TIMER0, TIMER_CH_0,  
TIMER_CCXN_ENABLE);
```

### timer\_channel\_input\_struct\_para\_init

The description of timer\_channel\_input\_struct\_para\_init is shown as below:

**Table 3-508. 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>	
<b>icpara</b>	TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-455. 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 3-509. 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..2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx (x=0..2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx (x=0..2))
<i>TIMER_CH_3</i>	TIMER channel 3(TIMERx (x=0..2))
<b>Input parameter{in}</b>	
<b>icpara</b>	TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-455. 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 3-510. 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..2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx (x=0..2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx (x=0..2))
<i>TIMER_CH_3</i>	TIMER channel 3(TIMERx (x=0..2))
<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 3-511. 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..2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx (x=0..2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx (x=0..2))
<i>TIMER_CH_3</i>	TIMER channel 3(TIMERx (x=0..2))
<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 3-512. 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..2, 14)</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>	

icpwm	TIMER channel input pwm parameter struct, the structure members can refer to <a href="#">Table 3-455. Structure timer_ic_parameter_struct</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 input pwm capture parameter */
timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_pwm_capture_config (TIMER0, TIMER_CH_0, &timer_icinitpara);

```

### timer\_hall\_mode\_config

The description of timer\_hall\_mode\_config is shown as below:

**Table 3-513. 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..2)	TIMER peripheral selection
Input parameter{in}	
hallmode	TIMER hall sensor mode state
TIMER_HALLINTERFA CE_ENABLE	TIMER hall sensor mode enable
TIMER_HALLINTERFA CE_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 3-514. Function timer\_input\_trigger\_source\_select**

<b>Function name</b>	timer_input_trigger_source_select
<b>Function prototype</b>	void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger);
<b>Function descriptions</b>	select TIMER input trigger source
<b>Precondition</b>	SMC[2:0] = 000
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 14)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>intrigger</b>	trigger selection
<i>TIMER_SMCFG_TRGS_EL_ITI0</i>	Internal trigger input 0(ITI0, TIMERx(x=0..2, 14))
<i>TIMER_SMCFG_TRGS_EL_ITI1</i>	Internal trigger input 0 (ITI1, TIMERx(x=0..2, 14))
<i>TIMER_SMCFG_TRGS_EL_ITI2</i>	Internal trigger input 0 (ITI2, TIMERx(ITI2, TIMERx(x=0..2))
<i>TIMER_SMCFG_TRGS_EL_ITI3</i>	Internal trigger input 0(ITI3, TIMERx(x=0..2, 14))
<i>TIMER_SMCFG_TRGS_EL_CIOF_ED</i>	CIO edge flag (CIOF_ED, TIMERx(x=0..2, 14))
<i>TIMER_SMCFG_TRGS_EL_CIOFE0</i>	channel 0 input Filtered output(CIOFE0, TIMERx(x=0..2, 14))
<i>TIMER_SMCFG_TRGS_EL_CI1FE1</i>	channel 1 input Filtered output(CI1FE1, TIMERx(x=0..2, 14))
<i>TIMER_SMCFG_TRGS_EL_ETIFP</i>	External trigger input filter output(ETIFP, TIMERx(x=0..2))
<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 3-515. Function timer\_master\_output\_trigger\_source\_select**

<b>Function name</b>	timer_master_output_trigger_source_select
<b>Function prototype</b>	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
<b>Function descriptions</b>	select TIMER master mode output trigger source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 5, 14)</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 3-516. Function timer\_slave\_mode\_select**

Function name	timer_slave_mode_select
Function prototype	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
Function descriptions	select TIMER slave mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..2, 14)	TIMER peripheral selection
Input parameter{in}	
slavemode	slave mode
TIMER_SLAVE_MODE_DISABLE	slave mode disable, TIMERx(x=0..2, 14)
TIMER_QUAD_DECODER_MODE0	quadrature decoder mode 0 TIMERx(x=0..2)
TIMER_QUAD_DECODER_MODE1	quadrature decoder mode 1, TIMERx(x=0..2)
TIMER_QUAD_DECODER_MODE2	quadrature decoder mode 2, TIMERx(x=0..2)
TIMER_SLAVE_MODE_RESTART	restart mode, TIMERx(x=0..2, 14)
TIMER_SLAVE_MODE_PAUSE	pause mode, TIMERx(x=0..2, 14)
TIMER_SLAVE_MODE_EVENT	event mode, TIMERx(x=0..2, 14)
TIMER_SLAVE_MODE_EXTERNAL0	external clock mode 0, TIMERx(x=0..2, 14)
Output parameter{out}	
-	-
Return value	
-	-

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 3-517. 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..2, 14)	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 3-518. Function timer\_external\_trigger\_config**

<b>Function name</b>	timer_external_trigger_config
<b>Function prototype</b>	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER external trigger input
<b>Precondition</b>	-
<b>The called functions</b>	-



Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2)</i>	TIMER peripheral selection
Input parameter{in}	
<b>extprescaler</b>	external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
Input parameter{in}	
<b>expolarity</b>	external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
Input parameter{in}	
<b>extfilter</b>	external trigger filter control (0~15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 external trigger input */
```

```
timer_external_trigger_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 10);
```

### timer\_quadrature\_decoder\_mode\_config

The description of timer\_quadrature\_decoder\_mode\_config is shown as below:

**Table 3-519. Function timer\_quadrature\_decoder\_mode\_config**

<b>Function name</b>	timer_quadrature_decoder_mode_config
<b>Function prototype</b>	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decompode, uint16_t ic0polarity, uint16_t ic1polarity);
<b>Function descriptions</b>	configure TIMER quadrature decoder mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2)</i>	TIMER peripheral selection

Input parameter{in}	
<b>decomode</b>	quadrature decoder mode
<i>TIMER_QUAD_DECODER_MODE0</i>	counter counts on CI0FE0 edge depending on CI1FE1 level
<i>TIMER_QUAD_DECODER_MODE1</i>	counter counts on CI1FE1 edge depending on CI0FE0 level
<i>TIMER_QUAD_DECODER_MODE2</i>	counter counts on both CI0FE0 and CI1FE1 edges depending on the level of the other input
Input parameter{in}	
<b>ic0polarity</b>	IC0 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
Input parameter{in}	
<b>ic1polarity</b>	IC1 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
Output parameter{out}	
-	-
Return value	
-	-

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 3-520. 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>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 14)</i>	TIMER peripheral selection

Output parameter{out}	
-	-
Return value	
-	-

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 3-521. Function timer\_internal\_trigger\_as\_external\_clock\_config**

Function name	timer_internal_trigger_as_external_clock_config
Function prototype	void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);
Function descriptions	configure TIMER the internal trigger as external clock input
Precondition	-
The called functions	timer_input_trigger_source_select
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i> (x=0..2, 14)	TIMER peripheral selection
Input parameter{in}	
<b>intrigger</b>	trigger selection
<i>TIMER_SMCFG_TRGS_EL_ITI0</i>	Internal trigger input 0 (ITI0), <i>TIMERx</i> (x=0..2, 14)
<i>TIMER_SMCFG_TRGS_EL_ITI1</i>	Internal trigger input 0 (ITI1) , <i>TIMERx</i> (x=0..2, 14)
<i>TIMER_SMCFG_TRGS_EL_ITI2</i>	Internal trigger input 0 (ITI2) , <i>TIMERx</i> (x=0..2)
Output parameter{out}	
-	-
Return value	
-	-

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 3-522. 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..2, 14)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extrigger</b>	external trigger selection
<i>TIMER_SMCFG_TRGS EL_CIOF_ED</i>	CIO edge flag (CIOF_ED)
<i>TIMER_SMCFG_TRGS EL_CIOFE0</i>	channel 0 input Filtered output (CIOFE0)
<i>TIMER_SMCFG_TRGS EL_C1FE1</i>	channel 1 input Filtered output (C1FE1)
<b>Input parameter{in}</b>	
<b>expolarity</b>	external trigger polarity
<i>TIMER_IC_POLARITY_ RISING</i>	active high or rising edge active
<i>TIMER_IC_POLARITY_ FALLING</i>	active low or falling edge active
<i>TIMER_IC_POLARITY_ 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 3-523. 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..2)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	ETI external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
<b>Input parameter{in}</b>	
<b>expolarity</b>	ETI external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
<b>Input parameter{in}</b>	
<b>extfilter</b>	ETI external trigger filter control (0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 the external clock mode0 */
```

```
timer_external_clock_mode0_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 0);
```

### timer\_external\_clock\_mode1\_config

The description of timer\_external\_clock\_mode1\_config is shown as below:

Table 3-524. Function timer\_external\_clock\_mode1\_config

<b>Function name</b>	timer_external_clock_mode1_config
<b>Function prototype</b>	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t

	extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external clock mode1
<b>Precondition</b>	-
<b>The called functions</b>	timer_external_trigger_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	ETI external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
<b>Input parameter{in}</b>	
<b>expolarity</b>	ETI external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
<b>Input parameter{in}</b>	
<b>extfilter</b>	ETI external trigger filter control (0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TIMER_ETP_FALLING, 0);
```

### timer\_external\_clock\_mode1\_disable

The description of timer\_external\_clock\_mode1\_disable is shown as below:

**Table 3-525. 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>	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_disable (TIMER0);
```

### timer\_channel\_remap\_config

The description of timer\_channel\_remap\_config is shown as below:

**Table 3-526. Function timer\_channel\_remap\_config**

<b>Function name</b>	timer_channel_remap_config
<b>Function prototype</b>	void timer_channel_remap_config (uint32_t timer_periph, uint32_t remap);
<b>Function descriptions</b>	configure TIMER channel remap function
<b>Precondition</b>	-
<b>The called functions</b>	timer_external_trigger_config
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=13)</i>	TIMER peripheral selection
Input parameter{in}	
<b>remap</b>	remap function selection
<i>TIMER13_CIO_RMP_GPIO</i>	timer13 channel 0 input is connected to GPIO(TIMER13_CH0)
<i>TIMER13_CIO_RMP_RTCCLK</i>	timer13 channel 0 input is connected to the RTCCLK
<i>TIMER13_CIO_RMP_HXTAL_DIV32</i>	timer13 channel 0 input is connected to HXTAL/32 clock
<i>TIMER13_CIO_RMP_CKOUTSEL</i>	timer13 channel 0 input is connected to CKOUTSEL
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER13 channel 0 input is connected to GPIO */
```

timer\_channel\_remap\_config (TIMER13, TIMER13\_CIO\_RMP\_GPIO);

## 3.22. TSI

Touch Sensing Interface (TSI) provides a convenient solution for touch keys, sliders and capacitive proximity sensing applications. The TSI registers are listed in chapter [3.22.1](#), the TSI firmware functions are introduced in chapter [3.22.2](#).

### 3.22.1. Descriptions of Peripheral registers

TSI registers are listed in the table shown as below:

**Table 3-527. TSI Registers**

Registers	Descriptions
TSI_CTL	Control register0
TSI_INTEN	Interrupt enable register
TSI_INTC	Interrupt flag clear register
TSI_INTF	Interrupt flag register
TSI_PHM	Pin hysteresis mode register
TSI_ASW	Analog switch register
TSI_SAMPCFG	Sample configuration register
TSI_CHCFG	Channel configuration register
TSI_GCTL	Group control register
TSI_GxCYCN (x= 0..5)	Group x cycle number registers

### 3.22.2. Descriptions of Peripheral functions

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

**Table 3-528. TSI firmware function**

Function name	Function description
tsi_deinit	reset TSI peripheral
tsi_init	initialize TSI plus prescaler,charge plus,transfer plus,max cycle number
tsi_enable	enable TSI module
tsi_disable	disable TSI module
tsi_sample_pin_enable	enable sample pin
tsi_sample_pin_disable	disable sample pin
tsi_channel_pin_enable	enable channel pin
tsi_channel_pin_disable	disable channel pin
tsi_software_mode_config	configure TSI triggering by software
tsi_software_start	start a charge-transfer sequence when TSI is in software



Function name	Function description
	trigger mode
tsi_software_stop	stop a charge-transfer sequence when TSI is in software trigger mode
tsi_hardware_mode_config	configure TSI triggering by hardware
tsi_pin_mode_config	configure TSI pin mode when charge-transfer sequence is IDLE
tsi_extend_charge_config	configure extend charge state
tsi_plus_config	configure charge plus and transfer plus
tsi_max_number_config	configure the max cycle number of a charge-transfer sequence
tsi_hysteresis_on	switch on hysteresis pin
tsi_hysteresis_off	switch off hysteresis pin
tsi_analog_on	switch on analog pin
tsi_analog_off	switch off analog pin
tsi_flag_get	get flag
tsi_flag_clear	clear flag
tsi_interrupt_enable	enable TSI interrupt
tsi_interrupt_disable	disable TSI interrupt
tsi_interrupt_flag_get	get TSI interrupt flag
tsi_interrupt_flag_clear	Clear TSI interrupt flag
tsi_group_enable	enable group
tsi_group_disable	disable group
tsi_group_status_get	get group complete status
tsi_group0_cycle_get	get the cycle number for group0 as soon as a charge-transfer sequence completes
tsi_group1_cycle_get	get the cycle number for group1 as soon as a charge-transfer sequence completes
tsi_group2_cycle_get	get the cycle number for group2 as soon as a charge-transfer sequence completes
tsi_group3_cycle_get	get the cycle number for group3 as soon as a charge-transfer sequence completes
tsi_group4_cycle_get	get the cycle number for group4 as soon as a charge-transfer sequence completes
tsi_group5_cycle_get	get the cycle number for group5 as soon as a charge-transfer sequence completes

## tsi\_deinit

The description of tsi\_deinit is shown as below:

**Table 3-529. Function tsi\_deinit**

Function name	tsi_deinit
---------------	------------

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

Example:

```
/* reset TSI*/
tsi_deinit();
```

## tsi\_init

The description of tsi\_init is shown as below:

**Table 3-530. Function tsi\_init**

<b>Function name</b>	tsi_init
<b>Function prototype</b>	void tsi_init(uint32_t prescaler,uint32_t charge_duration,uint32_t transfer_duration,uint32_t max_number);
<b>Function descriptions</b>	initialize TSI plus prescaler,charge plus,transfer plus,max cycle number
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>prescaler</b>	CTCLK clock division factor
<i>TSI_CTCDIV_DIV1</i>	$f_{CTCLK} = f_{HCLK}$
<i>TSI_CTCDIV_DIV2</i>	$f_{CTCLK} = f_{HCLK}/2$
<i>TSI_CTCDIV_DIV4</i>	$f_{CTCLK} = f_{HCLK} /4$
<i>TSI_CTCDIV_DIV8</i>	$f_{CTCLK} = f_{HCLK} /8$
<i>TSI_CTCDIV_DIV16</i>	$f_{CTCLK} = f_{HCLK} /16$
<i>TSI_CTCDIV_DIV32</i>	$f_{CTCLK} = f_{HCLK} /32$
<i>TSI_CTCDIV_DIV64</i>	$f_{CTCLK} = f_{HCLK} /64$
<i>TSI_CTCDIV_DIV128</i>	$f_{CTCLK} = f_{HCLK} /128$
<b>Input parameter{in}</b>	
<b>charge_duration</b>	charge state duration time
<i>TSI_CHARGE_1CTCL</i> <i>K(x=1..16)</i>	the duration time of charge state is x CTCLK
<b>Input parameter{in}</b>	
<b>transfer_duration</b>	charge transfer state duration time

<i>TSI_TRANSFER_xCTCLK</i> <i>LK(x=1..16)</i>	the duration time of transfer state is x CTCLK
<b>Input parameter{in}</b>	
<b>max_number</b>	max cycle number
<i>TSI_MAXNUM255</i>	the max cycle number of a sequence is 255
<i>TSI_MAXNUM511</i>	the max cycle number of a sequence is 511
<i>TSI_MAXNUM1023</i>	the max cycle number of a sequence is 1023
<i>TSI_MAXNUM2047</i>	the max cycle number of a sequence is 2047
<i>TSI_MAXNUM4095</i>	the max cycle number of a sequence is 4095
<i>TSI_MAXNUM8191</i>	the max cycle number of a sequence is 8191
<i>TSI_MAXNUM16383</i>	the max cycle number of a sequence is 16383
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* init TSI*/
```

```
tsi_init (TSI_CTCDIV_DIV128, TSI_CHARGE_10CTCLK, TSI_TRANSFER_8CTCLK,  
TSI_MAXNUM511);
```

### tsi\_enable

The description of tsi\_enable is shown as below:

**Table 3-531. Function tsi\_enable**

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

Example:

```
/* enable TSI module */
```

```
tsi_enable();
```

## tsi\_disable

The description of tsi\_disable is shown as below:

**Table 3-532. Function tsi\_disable**

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

Example:

```
/* disable TSI module */
tsi_disable ();
```

## tsi\_sample\_pin\_enable

The description of tsi\_sample\_pin\_enable is shown as below:

**Table 3-533. Function tsi\_sample\_pin\_enable**

<b>Function name</b>	tsi_sample_pin_enable
<b>Function prototype</b>	void tsi_sample_pin_enable(uint32_t sample);
<b>Function descriptions</b>	enable sample pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sample</b>	sample pin
<i>TSI_SAMPCFG_GxPy( x=0..5,y=0..3)</i>	pin y of group x is sample pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable G5P3 sample pin */
tsi_sample_pin_enable (TSI_SAMPCFG_G5P3);
```

## tsi\_sample\_pin\_disable

The description of tsi\_sample\_pin\_disable is shown as below:

**Table 3-534. Function tsi\_sample\_pin\_disable**

<b>Function name</b>	tsi_sample_pin_disable
<b>Function prototype</b>	void tsi_sample_pin_disable(uint32_t sample);
<b>Function descriptions</b>	disable sample pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sample</b>	sample pin
<i>TSI_SAMPCFG_GxPy(</i> <i>x=0..5,y=0..3)</i>	pin y of group x is sample pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable G5P3 sample pin */
tsi_sample_pin_disable (TSI_SAMPCFG_G5P3);
```

## tsi\_channel\_pin\_enable

The description of tsi\_channel\_pin\_enable is shown as below:

**Table 3-535. Function tsi\_channel\_pin\_enable**

<b>Function name</b>	tsi_channel_pin_enable
<b>Function prototype</b>	void tsi_channel_pin_enable(uint32_t channel);
<b>Function descriptions</b>	enable channel pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channel</b>	channel pin
<i>TSI_CHCFG_GxPy( x=</i> <i>0..5,y=0..3)</i>	pin y of group x
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable G5P3 channel pin */
```

```
tsi_channel_pin_enable (TSI_CHCFG_G5P3);
```

### tsi\_channel\_pin\_disable

The description of tsi\_channel\_pin\_disable is shown as below:

**Table 3-536. Function tsi\_channel\_pin\_disable**

<b>Function name</b>	tsi_channel_pin_disable
<b>Function prototype</b>	void tsi_channel_pin_disable(uint32_t channel);
<b>Function descriptions</b>	disable channel pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channel</b>	channel pin
<i>TSI_CHCFG_GxPy( x=0..5,y=0..3)</i>	pin y of group x
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable G5P3 channel pin */
```

```
tsi_channel_pin_disable (TSI_CHCFG_G5P3);
```

### tsi\_sofeware\_mode\_config

The description of tsi\_sofeware\_mode\_config is shown as below:

**Table 3-537. Function tsi\_sofeware\_mode\_config**

<b>Function name</b>	tsi_sofeware_mode_config
<b>Function prototype</b>	void tsi_sofeware_mode_config (void);
<b>Function descriptions</b>	configure TSI triggering by software
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TSI triggering by software */
```

```
tsi_software_mode_config ();
```

### tsi\_software\_start

The description of tsi\_software\_start is shown as below:

**Table 3-538. Function tsi\_software\_start**

<b>Function name</b>	tsi_software_start
<b>Function prototype</b>	void tsi_software_start (void);
<b>Function descriptions</b>	start a charge-transfer sequence when TSI is in software trigger mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* start a charge-transfer sequence when TSI is in software trigger mode */
```

```
tsi_software_start ();
```

### tsi\_software\_stop

The description of tsi\_software\_stop is shown as below:

**Table 3-539. Function tsi\_software\_stop**

<b>Function name</b>	tsi_software_stop
<b>Function prototype</b>	void tsi_software_stop (void);
<b>Function descriptions</b>	stop a charge-transfer sequence when TSI is in software trigger mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* stop a charge-transfer sequence when TSI is in software trigger mode */
```

```
tsi_software_stop ();
```

## tsi\_hardware\_mode\_config

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

**Table 3-540. Function tsi\_hardware\_mode\_config**

<b>Function name</b>	tsi_hardware_mode_config
<b>Function prototype</b>	void tsi_hardware_mode_config(uint8_t trigger_edge);
<b>Function descriptions</b>	configure TSI triggering by hardware
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>trigger_edge</b>	the edge type in hardware trigger mode
<i>TSI_FALLING_TRIGGER</i>	falling edge trigger TSI charge transfer sequence
<i>TSI_RISING_TRIGGER</i>	rising edge trigger TSI charge transfer sequence
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TSI triggering by hardware */
```

```
tsi_hardware_mode_config (TSI_FALLING_TRIGGER);
```

## tsi\_pin\_mode\_config

The description of tsi\_pin\_mode\_config is shown as below:

**Table 3-541. Function tsi\_pin\_mode\_config**

<b>Function name</b>	tsi_pin_mode_config
<b>Function prototype</b>	void tsi_pin_mode_config(uint8_t pin_mode);
<b>Function descriptions</b>	configure TSI pin mode when charge-transfer sequence is IDLE
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pin_mode</b>	pin mode when charge-transfer sequence is IDLE
<i>TSI_OUTPUT_LOW</i>	TSI pin will output low when IDLE
<i>TSI_INPUT_FLOATING</i>	TSI pin will keep input_floating when IDLE
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-



Example:

```
/* configure TSI pin mode when charge-transfer sequence is IDLE */
tsi_pin_mode_config (TSI_OUTPUT_LOW);
```

### tsi\_extend\_charge\_config

The description of tsi\_extend\_charge\_config is shown as below:

**Table 3-542. Function tsi\_extend\_charge\_config**

<b>Function name</b>	tsi_extend_charge_config
<b>Function prototype</b>	void tsi_extend_charge_config(ControlStatus extend,uint8_t prescaler,uint32_t max_duration);
<b>Function descriptions</b>	configure extend charge state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>extend</b>	enable or disable extend charge state
<i>ENABLE</i>	enable extend charge state
<i>DISABLE</i>	disable extend charge state
<b>Input parameter{in}</b>	
<b>prescaler</b>	ECCLK clock division factor
<i>TSI_EXTEND_DIV1</i>	$f_{ECCLK} = f_{HCLK}$
<i>TSI_EXTEND_DIV2</i>	$f_{ECCLK} = f_{HCLK} / 2$
<b>Input parameter{in}</b>	
<b>max_duration</b>	extend charge state maximum duration time
<i>value range 1...128</i>	extend charge state maximum duration time is $1 * t_{ECCLK} \sim 128 * t_{ECCLK}$
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure extend charge state */
tsi_extend_charge_config (ENABLE, TSI_EXTEND_DIV2, 10);
```

### tsi\_plus\_config

The description of tsi\_plus\_config is shown as below:

**Table 3-543. Function tsi\_plus\_config**

<b>Function name</b>	tsi_plus_config
<b>Function prototype</b>	void tsi_plus_config(uint32_t prescaler,uint32_t charge_duration,uint32_t transfer_duration);

<b>Function descriptions</b>	configure charge plus and transfer plus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>prescaler</b>	CTCLK clock division factor
<i>TSI_CTCDIV_DIV1</i>	$f_{CTCLK} = f_{HCLK}$
<i>TSI_CTCDIV_DIV2</i>	$f_{CTCLK} = f_{HCLK} / 2$
<i>TSI_CTCDIV_DIV4</i>	$f_{CTCLK} = f_{HCLK} / 4$
<i>TSI_CTCDIV_DIV8</i>	$f_{CTCLK} = f_{HCLK} / 8$
<i>TSI_CTCDIV_DIV16</i>	$f_{CTCLK} = f_{HCLK} / 16$
<i>TSI_CTCDIV_DIV32</i>	$f_{CTCLK} = f_{HCLK} / 32$
<i>TSI_CTCDIV_DIV64</i>	$f_{CTCLK} = f_{HCLK} / 64$
<i>TSI_CTCDIV_DIV128</i>	$f_{CTCLK} = f_{HCLK} / 128$
<b>Input parameter{in}</b>	
<b>charge_duration</b>	charge state duration time
<i>TSI_CHARGE_1CTCL</i> <i>K(x=1..16)</i>	the duration time of charge state is x CTCLK
<b>Input parameter{in}</b>	
<b>transfer_duration</b>	charge transfer state duration time
<i>TSI_TRANSFER_xCTC</i> <i>LK(x=1..16)</i>	the duration time of transfer state is x CTCLK
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure charge plus and transfer plus */
```

```
tsi_plus_config (TSI_CTCDIV_DIV128, TSI_CHARGE_10CTCLK,  
TSI_TRANSFER_8CTCLK);
```

### tsi\_max\_number\_config

The description of tsi\_max\_number\_config is shown as below:

**Table 3-544. Function tsi\_max\_number\_config**

<b>Function name</b>	tsi_max_number_config
<b>Function prototype</b>	void tsi_max_number_config(uint32_t max_number);
<b>Function descriptions</b>	configure the max cycle number of a charge-transfer sequence
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

max_number	max cycle number
<i>TSI_MAXNUM255</i>	the max cycle number of a sequence is 255
<i>TSI_MAXNUM511</i>	the max cycle number of a sequence is 511
<i>TSI_MAXNUM1023</i>	the max cycle number of a sequence is 1023
<i>TSI_MAXNUM2047</i>	the max cycle number of a sequence is 2047
<i>TSI_MAXNUM4095</i>	the max cycle number of a sequence is 4095
<i>TSI_MAXNUM8191</i>	the max cycle number of a sequence is 8191
<i>TSI_MAXNUM16383</i>	the max cycle number of a sequence is 16383
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the max cycle number of a charge-transfer sequence */
```

```
tsi_max_number_config (TSI_MAXNUM1023);
```

### tsi\_hysteresis\_on

The description of tsi\_hysteresis\_on is shown as below:

**Table 3-545. Function tsi\_hysteresis\_on**

Function name	tsi_hysteresis_on
Function prototype	void tsi_hysteresis_on(uint32_t group_pin);
Function descriptions	switch on hysteresis pin
Precondition	-
The called functions	-
<b>Input parameter{in}</b>	
group_pin	select pin which will be switched on hysteresis
<i>TSI_PHM_GxPy</i> ( <i>x=0..5</i> , <i>y=0..3</i> )	pin y of group x switch on hysteresis
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* switch on hysteresis pin */
```

```
tsi_hysteresis_on (TSI_PHM_G5P3);
```

### tsi\_hysteresis\_off

The description of tsi\_hysteresis\_off is shown as below:

Table 3-546. Function tsi\_hysteresis\_off

<b>Function name</b>	tsi_hysteresis_off
<b>Function prototype</b>	void tsi_hysteresis_off(uint32_t group_pin);
<b>Function descriptions</b>	switch off hysteresis pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>group_pin</b>	select pin which will be switched off hysteresis
<i>TSI_PHM_GxPy</i> ( <i>x=0..5</i> , <i>y=0..3</i> )	pin y of group x switch off hysteresis
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* switch off hysteresis pin */
tsi_hysteresis_off (TSI_PHM_G5P3);
```

### tsi\_analog\_on

The description of tsi\_analog\_on is shown as below:

Table 3-547. Function tsi\_analog\_on

<b>Function name</b>	tsi_analog_on
<b>Function prototype</b>	void tsi_analog_on(uint32_t group_pin);
<b>Function descriptions</b>	switch on analog pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>group_pin</b>	select pin which will be switched on analog
<i>TSI_ASW_GxPy</i> ( <i>x=0..5</i> , <i>y=0..3</i> )	pin y of group x switch on analog
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* switch on analog pin */
tsi_analog_on (TSI_ASW_G5P3);
```

## tsi\_analog\_off

The description of tsi\_analog\_off is shown as below:

**Table 3-548. Function tsi\_analog\_off**

<b>Function name</b>	tsi_analog_off
<b>Function prototype</b>	void tsi_analog_off(uint32_t group_pin);
<b>Function descriptions</b>	switch off analog pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>group_pin</b>	select pin which will be switched off analog
<i>TSI_ASW_GxPy</i> (x=0..5,y=0..3)	pin y of group x switch off analog
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* switch off analog pin */
tsi_analog_off (TSI_ASW_G5P3);
```

## tsi\_flag\_get

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

**Table 3-549. Function tsi\_flag\_get**

<b>Function name</b>	tsi_flag_get
<b>Function prototype</b>	FlagStatus tsi_flag_get(uint32_t flag);
<b>Function descriptions</b>	get flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag
<i>TSI_FLAG_CTCF</i>	charge-transfer complete flag
<i>TSI_FLAG_MNERR</i>	max cycle number error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TSI_FLAG_CTCF flag */
```

```
FlagStatus flag = RESET;
```

```
flag = tsi_flag_get (TSI_FLAG_CTCF);
```

### tsi\_flag\_clear

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

**Table 3-550. Function tsi\_flag\_clear**

<b>Function name</b>	tsi_flag_clear
<b>Function prototype</b>	void tsi_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	select flag which will be cleared
<i>TSI_FLAG_CTCF</i>	charge-transfer complete flag
<i>TSI_FLAG_MNERR</i>	max cycle number error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TSI_FLAG_CTCF flag */
```

```
tsi_flag_clear (TSI_FLAG_CTCF);
```

### tsi\_interrupt\_enable

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

**Table 3-551. Function tsi\_interrupt\_enable**

<b>Function name</b>	tsi_interrupt_enable
<b>Function prototype</b>	void tsi_interrupt_enable(uint32_t source);
<b>Function descriptions</b>	enable TSI interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	select interrupt which will be enabled
<i>TSI_INT_CCTCF</i>	charge-transfer complete flag interrupt enable
<i>TSI_INT_MNERR</i>	max cycle number error interrupt enable
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* enable TSI_INT_CCTCF interrupt */
tsi_interrupt_enable (TSI_INT_CCTCF);
```

### tsi\_interrupt\_disable

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

**Table 3-552. Function tsi\_interrupt\_disable**

Function name	tsi_interrupt_disable
Function prototype	void tsi_interrupt_disable(uint32_t source);
Function descriptions	disable TSI interrupt
Precondition	-
The called functions	-
Input parameter{in}	
source	select interrupt which will be disabled
<i>TSI_INT_CCTCF</i>	charge-transfer complete flag interrupt disable
<i>TSI_INT_MNERR</i>	max cycle number error interrupt disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TSI_INT_CCTCF interrupt */
tsi_interrupt_disable (TSI_INT_CCTCF);
```

### tsi\_interrupt\_flag\_get

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

**Table 3-553. Function tsi\_interrupt\_flag\_get**

Function name	tsi_interrupt_flag_get
Function prototype	FlagStatus tsi_interrupt_flag_get(uint32_t flag);
Function descriptions	get TSI interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	specify which flag
<i>TSI_INT_FLAG_CTCF</i>	charge-transfer complete interrupt flag

<i>TSI_INT_FLAG_MNER</i> <i>R</i>	max cycle number error interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TSI_INT_FLAG_CTCF interrupt flag */
```

```
FlagStatus flag = RESET;
```

```
flag = tsi_interrupt_flag_get (TSI_INT_FLAG_CTCF);
```

### tsi\_interrupt\_flag\_clear

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

**Table 3-554. Function tsi\_interrupt\_flag\_clear**

<b>Function name</b>	tsi_interrupt_flag_clear
<b>Function prototype</b>	void tsi_interrupt_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear TSI interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	select flag which will be cleared
<i>TSI_INT_FLAG_CTCF</i>	charge-transfer complete interrupt flag
<i>TSI_INT_FLAG_MNER</i> <i>R</i>	max cycle number error interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TSI_INT_FLAG_CTCF interrupt flag */
```

```
tsi_interrupt_flag_clear (TSI_INT_FLAG_CTCF);
```

### tsi\_group\_enable

The description of tsi\_group\_enable is shown as below:

**Table 3-555. Function tsi\_group\_enable**

<b>Function name</b>	tsi_group_enable
<b>Function prototype</b>	void tsi_group_enable(uint32_t group);



<b>Function descriptions</b>	enbale group
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>group</b>	select group to be enabled
<i>TSI_GCTL_GEx(x=0..5)</i>	the x group will be enabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable group 5 */
```

```
tsi_group_enable (TSI_GCTL_GE5);
```

### tsi\_group\_disable

The description of tsi\_group\_disable is shown as below:

**Table 3-556. Function tsi\_group\_disable**

<b>Function name</b>	tsi_group_disable
<b>Function prototype</b>	void tsi_group_disable(uint32_t group);
<b>Function descriptions</b>	disbale group
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>group</b>	select group to be disabled
<i>TSI_GCTL_GEx(x=0..5)</i>	the x group will be disabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable group 5 */
```

```
tsi_group_disable (TSI_GCTL_GE5);
```

### tsi\_group\_status\_get

The description of tsi\_group\_status\_get is shown as below:

**Table 3-557. Function tsi\_group\_status\_get**

<b>Function name</b>	tsi_group_status_get
----------------------	----------------------

<b>Function prototype</b>	FlagStatus tsi_group_status_get(uint32_t group);
<b>Function descriptions</b>	get group complete status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>group</b>	select group
<i>TSI_GCTL_GCx(x=0..5)</i>	get the complete status of group x
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get group complete status */

FlagStatus flag = RESET;

flag = tsi_flag_get (TSI_GCTL_GC5);
```

### tsi\_group0\_cycle\_get

The description of tsi\_group0\_cycle\_get is shown as below:

**Table 3-558. Function tsi\_group0\_cycle\_get**

<b>Function name</b>	tsi_group0_cycle_get
<b>Function prototype</b>	uint16_t tsi_group0_cycle_get(void);
<b>Function descriptions</b>	get the cycle number for group0 as soon as a charge-transfer sequence completes
<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>	0-16383

Example:

```
/* get the cycle number for group0 as soon as a charge-transfer sequence completes */

uint16_t flag = 0;

flag = tsi_group0_cycle_get ();
```

## tsi\_group1\_cycle\_get

The description of tsi\_group1\_cycle\_get is shown as below:

**Table 3-559. Function tsi\_group1\_cycle\_get**

<b>Function name</b>	tsi_group1_cycle_get
<b>Function prototype</b>	uint16_t tsi_group1_cycle_get(void);
<b>Function descriptions</b>	get the cycle number for group1 as soon as a charge-transfer sequence completes
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	0-16383

Example:

```
/* get the cycle number for group1 as soon as a charge-transfer sequence completes */
uint16_t flag = 0;
flag = tsi_group1_cycle_get ();
```

## tsi\_group2\_cycle\_get

The description of tsi\_group2\_cycle\_get is shown as below:

**Table 3-560. Function tsi\_group2\_cycle\_get**

<b>Function name</b>	tsi_group2_cycle_get
<b>Function prototype</b>	uint16_t tsi_group2_cycle_get(void);
<b>Function descriptions</b>	get the cycle number for group2 as soon as a charge-transfer sequence completes
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	0-16383

Example:

```
/* get the cycle number for group2 as soon as a charge-transfer sequence completes */
```

```
uint16_t flag = 0;

flag = tsi_group2_cycle_get ();
```

### tsi\_group3\_cycle\_get

The description of tsi\_group3\_cycle\_get is shown as below:

**Table 3-561. Function tsi\_group3\_cycle\_get**

<b>Function name</b>	tsi_group3_cycle_get
<b>Function prototype</b>	uint16_t tsi_group3_cycle_get(void);
<b>Function descriptions</b>	get the cycle number for group3 as soon as a charge-transfer sequence completes
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	0-16383

Example:

```
/* get the cycle number for group3 as soon as a charge-transfer sequence completes */
uint16_t flag = 0;

flag = tsi_group3_cycle_get ();
```

### tsi\_group4\_cycle\_get

The description of tsi\_group4\_cycle\_get is shown as below:

**Table 3-562. Function tsi\_group4\_cycle\_get**

<b>Function name</b>	tsi_group4_cycle_get
<b>Function prototype</b>	uint16_t tsi_group4_cycle_get(void);
<b>Function descriptions</b>	get the cycle number for group4 as soon as a charge-transfer sequence completes
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	0-16383

Example:

```
/* get the cycle number for group4 as soon as a charge-transfer sequence completes */
uint16_t flag = 0;

flag = tsi_group4_cycle_get ();
```

### tsi\_group5\_cycle\_get

The description of tsi\_group5\_cycle\_get is shown as below:

**Table 3-563. Function tsi\_group5\_cycle\_get**

<b>Function name</b>	tsi_group5_cycle_get
<b>Function prototype</b>	uint16_t tsi_group5_cycle_get(void);
<b>Function descriptions</b>	get the cycle number for group5 as soon as a charge-transfer sequence completes
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	0-16383

Example:

```
/* get the cycle number for group5 as soon as a charge-transfer sequence completes */
uint16_t flag = 0;

flag = tsi_group5_cycle_get ();
```

## 3.23. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter [3.23.1](#), the USART firmware functions are introduced in chapter [3.23.2](#).

### 3.23.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

**Table 3-564. USART Registers**

Registers	Descriptions
USART_CTL0	Control register 0

Registers	Descriptions
USART_CTL1	Control register 1
USART_CTL2	Control register 2
USART_BAUD	Baud rate register
USART_GP	Guard time and prescaler register
USART_RT	Receiver timeout register
USART_CMD	Command register
USART_STAT	Status register
USART_INTC	Status clear register
USART_RDATA	Receive data register
USART_TDATA	Transmit data register

### 3.23.2. Descriptions of Peripheral functions

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

**Table 3-565. 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_first_config	data is transmitted/received with the LSB/MSB first
usart_invert_config	configure USART inverted
usart_overrun_enable	enable the USART overrun function
usart_overrun_disable	disable the USART overrun function
usart_oversample_config	configure the USART oversample mode
usart_sample_bit_config	configure sample bit method
usart_receiver_timeout_enable	enable receiver timeout
usart_receiver_timeout_disable	disable receiver timeout
usart_receiver_timeout_threshold_config	configure receiver timeout threshold
usart_data_transmit	USART transmit data function
usart_data_receive	USART receive data function
usart_address_config	configure the address of the USART in wake up by address match mode
usart_address_detection_mode_config	configure address detection mode

Function name	Function description
g	
usart_mute_mode_enable	enable mute mode
usart_mute_mode_disable	disable mute mode
usart_mute_mode_wakeup_config	configure wakeup method in mute mode
usart_lin_mode_enable	enable LIN mode
usart_lin_mode_disable	disable LIN mode
usart_lin_break_dection_length_config	configure LIN break frame length
usart_halfduplex_enable	enable half duplex mode
usart_halfduplex_disable	disable half duplex mode
usart_clock_enable	enable USART clock
usart_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
usart_smartcard_mode_enable	enable smartcard mode
usart_smartcard_mode_disable	disable smartcard mode
usart_smartcard_mode_nack_enable	enable NACK in smartcard mode
usart_smartcard_mode_nack_disable	disable NACK in smartcard mode
usart_smartcard_autoretry_config	configure smartcard auto-retry number
usart_block_length_config	configure block length
usart_irda_mode_enable	enable IrDA mode
usart_irda_mode_disable	disable IrDA mode
usart_prescaler_config	configure the peripheral clock prescaler in USART IrDA low-power mode or SmartCard 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_rs485_driver_enable	enable RS485 driver
usart_rs485_driver_disable	disable RS485 driver
usart_driver_asserttime_config	configure driver enable assertion time
usart_driver_deasserttime_config	configure driver enable de-assertion time
usart_depolarity_config	configure driver enable polarity mode
usart_dma_receive_config	configure USART DMA for reception
usart_dma_transmit_config	configure USART DMA for transmission
usart_reception_error_dma_disable	disable DMA on reception error
usart_reception_error_dma_enable	enable DMA on reception error
usart_wakeup_enable	enable USART to wakeup the mcu from deep-sleep mode
usart_wakeup_disable	disable USART to wakeup the mcu from deep-sleep mode
usart_wakeup_mode_config	configure the USART wakeup mode from deep-sleep mode
usart_command_enable	enable USART command
usart_flag_get	get flag in STAT/RFCs register

Function name	Function description
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

### Enum usart\_flag\_enum

**Table 3-566. Enum usart\_flag\_enum**

Member name	Function description
USART_FLAG_REA	receive enable acknowledge flag
USART_FLAG_TEA	transmit enable acknowledge flag
USART_FLAG_WU	wakeup from deep-sleep mode flag
USART_FLAG_RWU	receiver wakeup from mute mode
USART_FLAG_SB	send break flag
USART_FLAG_AM	ADDR match flag
USART_FLAG_BSY	busy flag
USART_FLAG_EB	end of block flag
USART_FLAG_RT	receiver timeout flag
USART_FLAG_CTS	CTS level
USART_FLAG_CTSF	CTS change flag
USART_FLAG_LBD	LIN break detected flag
USART_FLAG_TBE	transmit data buffer empty
USART_FLAG_TC	transmission complete
USART_FLAG_RBNE	read data buffer not empty
USART_FLAG_IDLE	IDLE line detected flag
USART_FLAG_ORERR	overrun error
USART_FLAG_NERR	noise error flag
USART_FLAG_FERR	frame error flag
USART_FLAG_PERR	parity error flag

### Enum usart\_interrupt\_flag\_enum

**Table 3-567. Enum usart\_interrupt\_flag\_enum**

Member name	Function description
USART_INT_FLAG_EB	end of block interrupt flag
USART_INT_FLAG_RT	receiver timeout interrupt flag
USART_INT_FLAG_AM	address match interrupt flag
USART_INT_FLAG_PERR	parity error interrupt flag
USART_INT_FLAG_TBE	transmitter buffer empty interrupt flag
USART_INT_FLAG_TC	transmission complete interrupt flag
USART_INT_FLAG_RBNE	read data buffer not empty interrupt flag



Member name	Function description
USART_INT_FLAG_RBNE_ORE RR	overrun error interrupt flag
USART_INT_FLAG_IDLE	IDLE line detected interrupt flag
USART_INT_FLAG_LBD	LIN break detected interrupt flag
USART_INT_FLAG_WU	wakeup from deep-sleep mode interrupt flag
USART_INT_FLAG_CTS	CTS interrupt flag
USART_INT_FLAG_ERR_NERR	noise error interrupt flag
USART_INT_FLAG_ERR_ORER R	overrun error interrupt flag
USART_INT_FLAG_ERR_FERR	frame error interrupt flag

### Enum usart\_interrupt\_enum

**Table 3-568. Enum usart\_interrupt\_enum**

Member name	Function description
USART_INT_EB	end of block interrupt
USART_INT_RT	receiver timeout interrupt
USART_INT_AM	address match interrupt
USART_INT_PERR	parity error interrupt
USART_INT_TBE	transmitter buffer empty interrupt
USART_INT_TC	transmission complete interrupt
USART_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt
USART_INT_IDLE	IDLE line detected interrupt
USART_INT_LBD	LIN break detected interrupt
USART_INT_WU	wakeup from deep-sleep mode interrupt
USART_INT_CTS	CTS interrupt
USART_INT_ERR	error interrupt

### Enum usart\_invert\_enum

**Table 3-569. Enum usart\_invert\_enum**

Member name	Function description
USART_DINV_ENABLE	data bit level inversion
USART_DINV_DISABLE	data bit level not inversion
USART_TXPIN_ENABLE	TX pin level inversion
USART_TXPIN_DISABLE	TX pin level not inversion
USART_RXPIN_ENABLE	RX pin level inversion
USART_RXPIN_DISABLE	RX pin level not inversion
USART_SWAP_ENABLE	swap TX/RX pins
USART_SWAP_DISABLE	not swap TX/RX pins

## usart\_deinit

The description of usart\_deinit is shown as below:

**Table 3-570. 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
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<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 3-571. 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>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<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 3-572. 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>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>paritycfg</b>	configure USART parity
<i>USART_PM_NONE</i>	no parity
<i>USART_PM_ODD</i>	odd parity
<i>USART_PM_EVEN</i>	even parity
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure 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 3-573. Function usart\_word\_length\_set**

<b>Function name</b>	usart_word_length_set
<b>Function prototype</b>	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);
<b>Function descriptions</b>	configure USART word length
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>wlen</b>	USART word length configure
<i>USART_WL_8BIT</i>	8 bits
<i>USART_WL_9BIT</i>	9 bits
<b>Output parameter{out}</b>	

-	-
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 3-574. Function usart\_stop\_bit\_set**

<b>Function name</b>	usart_stop_bit_set
<b>Function prototype</b>	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);
<b>Function descriptions</b>	configure USART stop bit length
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1
<b>Input parameter{in}</b>	
<b>stblen</b>	USART stop bit configure
USART_STB_1BIT	1 bit
USART_STB_0_5BIT	0.5 bit
USART_STB_2BIT	2 bits
USART_STB_1_5BIT	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 3-575. 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>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<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 3-576. 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>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<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 3-577. 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>	-

Input parameter{in}	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
Input parameter{in}	
<b>txconfig</b>	enable or disable USART transmitter
<i>USART_TRANSMIT_ENABLE</i>	enable USART transmission
<i>USART_TRANSMIT_DISABLE</i>	disable USART transmission
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 3-578. 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>	-
Input parameter{in}	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
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\_first\_config

The description of usart\_data\_first\_config is shown as below:

**Table 3-579. Function usart\_data\_first\_config**

<b>Function name</b>	usart_data_first_config
<b>Function prototype</b>	void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);
<b>Function descriptions</b>	data is transmitted/received with the LSB/MSB first
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1
<b>Input parameter{in}</b>	
<b>msbf</b>	LSB/MSB
USART_MSBF_LSB	LSB first
USART_MSBF_MSB	MSB first
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LSB of data first */
```

```
usart_data_first_config(USART0, USART_MSBF_LSB);
```

### usart\_invert\_config

The description of usart\_invert\_config is shown as below:

**Table 3-580. Function usart\_invert\_config**

<b>Function name</b>	usart_invert_config
<b>Function prototype</b>	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);
<b>Function descriptions</b>	USART inverted configure
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1
<b>Input parameter{in}</b>	
<b>invertpara</b>	refer to usart_invert_enum
USART_DINV_ENABLER	data bit level inversion

<i>USART_DINV_DISABL</i> <i>E</i>	data bit level not inversion
<i>USART_TXPIN_ENABL</i> <i>E</i>	TX pin level inversion
<i>USART_TXPIN_DISAB</i> <i>LE</i>	TX pin level not inversion
<i>USART_RXPIN_ENAB</i> <i>LE</i>	RX pin level inversion
<i>USART_RXPIN_DISAB</i> <i>LE</i>	RX pin level not inversion
<i>USART_SWAP_ENABL</i> <i>E</i>	swap TX/RX pins
<i>USART_SWAP_DISAB</i> <i>LE</i>	not swap TX/RX pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 inversion */
usart_invert_config(USART0, USART_DINV_ENABLE);
```

### usart\_overrun\_enable

The description of usart\_overrun\_enable is shown as below:

**Table 3-581. Function usart\_overrun\_enable**

<b>Function name</b>	usart_overrun_enable
<b>Function prototype</b>	void usart_overrun_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable the USART overrun function
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 overrun */
```



```
usart_oversample_enable(USART0);
```

### usart\_oversample\_disable

The description of usart\_oversample\_disable is shown as below:

**Table 3-582. Function usart\_oversample\_disable**

<b>Function name</b>	usart_oversample_disable
<b>Function prototype</b>	void usart_oversample_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable the USART oversample function
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 overrun */
usart_oversample_disable(USART0);
```

### usart\_oversample\_config

The description of usart\_oversample\_config is shown as below:

**Table 3-583. Function usart\_oversample\_config**

<b>Function name</b>	usart_oversample_config
<b>Function prototype</b>	void usart_oversample_config(uint32_t usart_periph, uint32_t oversamp);
<b>Function descriptions</b>	configure the USART oversample mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1
<b>Input parameter{in}</b>	
<b>oversamp</b>	oversample value
USART_OVSMOD_8	oversampling by 8
USART_OVSMOD_16	oversampling by 16
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 oversampling by 8 */
usart_oversample_config(USART0,USART_OVSMOD_8);
```

### usart\_sample\_bit\_config

The description of usart\_sample\_bit\_config is shown as below:

**Table 3-584. Function usart\_sample\_bit\_config**

<b>Function name</b>	usart_sample_bit_config
<b>Function prototype</b>	void usart_sample_bit_config(uint32_t usart_periph, uint32_t osb);
<b>Function descriptions</b>	configure the sample bit method
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1
<b>Input parameter{in}</b>	
<b>osb</b>	sample bit method
USART_OSB_1BIT	One sample bit method
USART_OSB_3BIT	Three sample bit method
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config USART0 1 bit sample mode */
usart_sample_bit_config(USART0, USART_OSB_1BIT);
```

### usart\_receiver\_timeout\_enable

The description of usart\_receiver\_timeout\_enable is shown as below:

**Table 3-585. Function usart\_receiver\_timeout\_enable**

<b>Function name</b>	usart_receiver_timeout_enable
<b>Function prototype</b>	void usart_receiver_timeout_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable receiver timeout
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* enable USART0 receiver timeout */
```

```
usart_receiver_timeout_enable(USART0);
```

### usart\_receiver\_timeout\_disable

The description of usart\_receiver\_timeout\_disable is shown as below:

**Table 3-586. Function usart\_receiver\_timeout\_disable**

Function name	usart_receiver_timeout_disable
Function prototype	void usart_receiver_timeout_disable(uint32_t usart_periph);
Function descriptions	disable receiver timeout
Precondition	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 receiver timeout */
```

```
usart_receiver_timeout_disable(USART0);
```

### usart\_receiver\_timeout\_threshold\_config

The description of usart\_receiver\_timeout\_threshold\_config is shown as below:

**Table 3-587. Function usart\_receiver\_timeout\_threshold\_config**

Function name	usart_receiver_timeout_threshold_config
Function prototype	void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);
Function descriptions	configure receiver timeout threshold
Precondition	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0
Input parameter{in}	
rtimeout	receiver timeout

0x00000000- 0x00FFFFFF	receiver timeout value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the receiver timeout threshold of USART0*/
```

```
usart_receiver_timeout_threshold_config(USART0, 115200*3);
```

### usart\_data\_transmit

The description of usart\_data\_transmit is shown as below:

**Table 3-588. Function usart\_data\_transmit**

<b>Function name</b>	usart_data_transmit
<b>Function prototype</b>	void usart_data_transmit(uint32_t usart_periph, uint32_t data);
<b>Function descriptions</b>	USART transmit data function
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1
<b>Input parameter{in}</b>	
<b>data</b>	data of transmission
0-0x1FF	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 3-589. 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>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	data of received (0-0xFF)

Example:

```
/* USART0 receive data */
uint16_t temp;
temp = usart_data_receive(USART0);
```

### usart\_address\_config

The description of usart\_address\_config is shown as below:

**Table 3-590. 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>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>addr</b>	address of USART
<i>0-0xFF</i>	address of USART
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure address of the USART0 */
usart_address_config(USART0, 0x00);
```

### usart\_address\_detection\_mode\_config

The description of usart\_address\_detection\_mode\_config is shown as below:

Table 3-591. Function `usart_address_detection_mode_config`

<b>Function name</b>	<code>usart_address_detection_mode_config</code>
<b>Function prototype</b>	<code>void usart_address_detection_mode_config(uint32_t usart_periph, uint32_t addmod);</code>
<b>Function descriptions</b>	configure address detection mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>addmod</b>	address detection mode
<i>USART_ADDM_4BIT</i>	4 bits
<i>USART_ADDM_FULLBIT</i>	full bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*configure address detection mode */
usart_address_config(USART0, USART_ADDM_4BIT);
```

### **usart\_mute\_mode\_enable**

The description of `usart_mute_mode_enable` is shown as below:

Table 3-592. Function `usart_mute_mode_enable`

<b>Function name</b>	<code>usart_mute_mode_enable</code>
<b>Function prototype</b>	<code>void usart_mute_mode_enable(uint32_t usart_periph);</code>
<b>Function descriptions</b>	enable mute mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 receiver in mute mode */
```

```
usart_mute_mode_enable(USART0);
```

### usart\_mute\_mode\_disable

The description of usart\_mute\_mode\_disable is shown as below:

**Table 3-593. Function usart\_mute\_mode\_disable**

<b>Function name</b>	usart_mute_mode_disable
<b>Function prototype</b>	void usart_mute_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable mute mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 receiver in mute mode */
```

```
usart_mute_mode_disable(USART0);
```

### usart\_mute\_mode\_wakeup\_config

The description of usart\_mute\_mode\_wakeup\_config is shown as below:

**Table 3-594. Function usart\_mute\_mode\_wakeup\_config**

<b>Function name</b>	usart_mute_mode_wakeup_config
<b>Function prototype</b>	void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);
<b>Function descriptions</b>	configure wakeup method in mute mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1
<b>Input parameter{in}</b>	
<b>wmethod</b>	two methods be used to enter or exit the mute mode
USART_WM_IDLE	idle line
USART_WM_ADDR	address mask
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 wakeup method in mute mode */
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

### usart\_lin\_mode\_enable

The description of usart\_lin\_mode\_enable is shown as below:

**Table 3-595. 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>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 LIN mode enable */
usart_lin_mode_enable(USART0);
```

### usart\_lin\_mode\_disable

The description of usart\_lin\_mode\_disable is shown as below:

**Table 3-596. Function usart\_lin\_mode\_disable**

<b>Function name</b>	usart_lin_mode_disable
<b>Function prototype</b>	void usart_lin_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable LIN mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:



```
/* USART0 LIN mode disable */
```

```
usart_lin_mode_disable(USART0);
```

### usart\_lin\_break\_dection\_length\_config

The description of usart\_lin\_break\_dection\_length\_config is shown as below:

**Table 3-597. Function usart\_lin\_break\_dection\_length\_config**

<b>Function name</b>	usart_lin_break_dection_length_config
<b>Function prototype</b>	void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lblen);
<b>Function descriptions</b>	configure lin break frame length
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0
<b>Input parameter{in}</b>	
<b>lblen</b>	LIN break detection length
USART_LBLEN_10B	10 bits break detection
USART_LBLEN_11B	11 bits break detection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LIN break frame length */
```

```
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

### usart\_halfduplex\_enable

The description of usart\_halfduplex\_enable is shown as below:

**Table 3-598. 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>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1
<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 3-599. 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	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 half duplex mode*/
```

```
usart_halfduplex_disable(USART0);
```

### usart\_clock\_enable

The description of usart\_clock\_enable is shown as below:

**Table 3-600. Function usart\_clock\_enable**

Function name	usart_clock_enable
Function prototype	void usart_clock_enable(uint32_t usart_periph);
Function descriptions	enable USART clock
Precondition	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable USART0 CK pin */
```

```
usart_synchronous_clock_enable(USART0);
```

### usart\_clock\_disable

The description of usart\_clock\_disable is shown as below:

**Table 3-601. Function usart\_clock\_disable**

<b>Function name</b>	usart_clock_disable
<b>Function prototype</b>	void usart_clock_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable USART clock
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 CK pin */
```

```
usart_synchronous_clock_disable(USART0);
```

### usart\_synchronous\_clock\_config

The description of usart\_synchronous\_clock\_config is shown as below:

**Table 3-602. Function usart\_synchronous\_clock\_config**

<b>Function name</b>	usart_synchronous_clock_config
<b>Function prototype</b>	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
<b>Function descriptions</b>	configure USART synchronous mode parameters
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1
<b>Input parameter{in}</b>	
<b>clen</b>	last bit clock pulse
USART_CLEN_NONE	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 3-603. Function usart\_guard\_time\_config**

<b>Function name</b>	usart_guard_time_config
<b>Function prototype</b>	void usart_guard_time_config(uint32_t usart_periph,uint32_t guat);
<b>Function descriptions</b>	configure guard time value in smartcard mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<b>Input parameter{in}</b>	
<b>guat</b>	guard time value
<i>0-0x000000FF</i>	guard time value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 guard time value in smartcard mode */

usart_guard_time_config(USART0, 0x0000 0055);
```

## usart\_smartcard\_mode\_enable

The description of usart\_smartcard\_mode\_enable is shown as below:

**Table 3-604. 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>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<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 3-605. 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>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 smartcard mode disable */
usart_smartcard_mode_disable(USART0);
```

## usart\_smartcard\_mode\_nack\_enable

The description of usart\_smartcard\_mode\_nack\_enable is shown as below:

**Table 3-606. Function usart\_smartcard\_mode\_nack\_enable**

<b>Function name</b>	usart_smartcard_mode_nack_enable
<b>Function prototype</b>	void usart_smartcard_mode_nack_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable NACK in smartcard mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_enable(USART0);
```

## usart\_smartcard\_mode\_nack\_disable

The description of usart\_smartcard\_mode\_nack\_disable is shown as below:

**Table 3-607. Function usart\_smartcard\_mode\_nack\_disable**

<b>Function name</b>	usart_smartcard_mode_nack_disable
<b>Function prototype</b>	void usart_smartcard_mode_nack_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable NACK in smartcard mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_disable(USART0);
```

## usart\_smartcard\_autoretry\_config

The description of usart\_smartcard\_autoretry\_config is shown as below:

**Table 3-608. Function usart\_smartcard\_autoretry\_config**

<b>Function name</b>	usart_smartcard_autoretry_config
<b>Function prototype</b>	void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrtnum);
<b>Function descriptions</b>	configure smartcard auto-retry number
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0
<b>Input parameter{in}</b>	
<b>scrtnum</b>	smartcard auto-retry number
0x00000000- 0x00000007	smartcard auto-retry number
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure smartcard auto-retry number */
```

```
usart_smartcard_autoretry_config(USART0, 0x00000007);
```

## usart\_block\_length\_config

The description of usart\_block\_length\_config is shown as below:

**Table 3-609. Function usart\_block\_length\_config**

<b>Function name</b>	usart_block_length_config
<b>Function prototype</b>	void usart_block_length_config(uint32_t usart_periph, uint32_t bl);
<b>Function descriptions</b>	configure block length
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0
<b>Input parameter{in}</b>	
<b>bl</b>	block length
0-0x000000FF	block length
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* configure block length in Smartcard T=1 reception */
usart_block_length_config(USART0, 0x000000FF);
```

### usart\_irda\_mode\_enable

The description of usart\_irda\_mode\_enable is shown as below:

**Table 3-610. Function usart\_irda\_mode\_enable**

Function name	usart_irda_mode_enable
Function prototype	void usart_irda_mode_enable(uint32_t usart_periph);
Function descriptions	enable IrDA mode
Precondition	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 IrDA mode */
usart_irda_mode_enable(USART0);
```

### usart\_irda\_mode\_disable

The description of usart\_irda\_mode\_disable is shown as below:

**Table 3-611. Function usart\_irda\_mode\_disable**

Function name	usart_irda_mode_disable
Function prototype	void usart_irda_mode_disable(uint32_t usart_periph);
Function descriptions	disable IrDA mode
Precondition	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0
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 3-612. Function usart\_prescaler\_config**

<b>Function name</b>	usart_prescaler_config
<b>Function prototype</b>	void usart_prescaler_config(uint32_t usart_periph, uint8_t psc);
<b>Function descriptions</b>	configure the peripheral clock prescaler in USART IrDA low-power mode or SmartCard mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<b>Input parameter{in}</b>	
<b>psc</b>	clock prescaler
<i>0x00-0xFF</i>	clock prescaler
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-613. Function usart\_irda\_lowpower\_config**

<b>Function name</b>	usart_irda_lowpower_config
<b>Function prototype</b>	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
<b>Function descriptions</b>	configure IrDA low-power
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0

Input parameter{in}	
<b>irlp</b>	IrDA low-power or normal
<i>USART_IRLP_LOW</i>	low-power
<i>USART_IRLP_NORMAL</i>	normal
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 IrDA low-power */
```

```
usart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

### usart\_hardware\_flow\_rts\_config

The description of usart\_hardware\_flow\_rts\_config is shown as below:

**Table 3-614. 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>	-
Input parameter{in}	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
Input parameter{in}	
<b>rtsconfig</b>	enable or disable RTS
<i>USART_RTS_ENABLE</i>	enable RTS
<i>USART_RTS_DISABLE</i>	disable RTS
Output parameter{out}	
-	-
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 3-615. Function `usart_hardware_flow_cts_config`

<b>Function name</b>	<code>usart_hardware_flow_cts_config</code>
<b>Function prototype</b>	<code>void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);</code>
<b>Function descriptions</b>	configure hardware flow control CTS
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>ctsconfig</b>	enable or disable CTS
<i>USART_CTS_ENABLE</i>	enable CTS
<i>USART_CTS_DISABLE</i>	disable CTS
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 hardware flow control CTS */
usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

### **usart\_rs485\_driver\_enable**

The description of `usart_rs485_driver_enable` is shown as below:

Table 3-616. Function `usart_rs485_driver_enable`

<b>Function name</b>	<code>usart_rs485_driver_enable</code>
<b>Function prototype</b>	<code>void usart_rs485_driver_enable(uint32_t usart_periph);</code>
<b>Function descriptions</b>	enable USART RS485 driver
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 RS485 driver */
usart_rs485_driver_enable(USART0);
```

## usart\_rs485\_driver\_disable

The description of usart\_rs485\_driver\_disable is shown as below:

**Table 3-617. Function usart\_rs485\_driver\_disable**

<b>Function name</b>	usart_rs485_driver_disable
<b>Function prototype</b>	void usart_rs485_driver_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable USARTR485 driver
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 RS485 driver */
usart_rs485_driver_disable (USART0);
```

## usart\_driver\_assertime\_config

The description of usart\_driver\_assertime\_config is shown as below:

**Table 3-618. Function usart\_driver\_assertime\_config**

<b>Function name</b>	usart_driver_assertime_config
<b>Function prototype</b>	void usart_driver_assertime_config(uint32_t usart_periph, uint32_t deatime);
<b>Function descriptions</b>	configure driver enable assertion time
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>deatime</b>	driver enable assertion time
<i>0-0x0000001F</i>	driver enable assertion time
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set USART0 driver assertime */
```

```
usart_driver_asserttime_config(USART0,0x0000001F);
```

### usart\_driver\_deasserttime\_config

The description of usart\_driver\_deasserttime\_config is shown as below:

**Table 3-619. Function usart\_driver\_deasserttime\_config**

<b>Function name</b>	usart_driver_deasserttime_config
<b>Function prototype</b>	void usart_driver_deasserttime_config(uint32_t usart_periph, uint32_t dedtime);
<b>Function descriptions</b>	configure driver enable de-assertion time
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1
<b>Input parameter{in}</b>	
<b>deatime</b>	driver enable de-assertion time
0x00000000- 0x0000001F	driver enable de-assertion time
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set USART0 driver deasserttime */
```

```
usart_driver_deasserttime_config(USART0, 0x0000001F);
```

### usart\_depolarity\_config

The description of usart\_depolarity\_config is shown as below:

**Table 3-620. Function usart\_depolarity\_config**

<b>Function name</b>	usart_depolarity_config
<b>Function prototype</b>	void usart_depolarity_config(uint32_t usart_periph, uint32_t dep);
<b>Function descriptions</b>	configure driver enable polarity mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1
<b>Input parameter{in}</b>	
<b>dep</b>	DE signal
USART_DEP_HIGH	DE signal is active high
USART_DEP_LOW	DE signal is active low

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure driver enable polarity mode */
```

```
usart_driver_depolarity_config(USART0, USART_DEP_HIGH);
```

### usart\_dma\_receive\_config

The description of usart\_dma\_receive\_config is shown as below:

**Table 3-621. Function usart\_dma\_receive\_config**

Function name	usart_dma_receive_config
Function prototype	void usart_dma_receive_config(uint32_t usart_periph, uint8_t dmacmd);
Function descriptions	configure USART DMA reception
Precondition	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1
Input parameter{in}	
dmacmd	enable or disable DMA for reception
USART_RECEIVE_DMA_ENABLE	DMA enable for reception
USART_RECEIVE_DMA_DISABLE	DMA disable for reception
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 DMA enable for reception */
```

```
usart_dma_receive_config(USART0, USART_DENR_ENABLE);
```

### usart\_dma\_transmit\_config

The description of usart\_dma\_transmit\_config is shown as below:

**Table 3-622. Function usart\_dma\_transmit\_config**

Function name	usart_dma_transmit_config
Function prototype	void usart_dma_transmit_config(uint32_t usart_periph, uint8_t dmacmd);

<b>Function descriptions</b>	configure USART DMA transmission
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>dmacmd</b>	enable or disable DMA for transmission
<i>USART_TRANSMIT_DMA_ENABLE</i>	DMA enable for transmission
<i>USART_TRANSMIT_DMA_DISABLE</i>	DMA disable for transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 DMA enable for transmission */
```

```
usart_dma_transmit_config(USART0, USART_DENT_ENABLE);
```

### usart\_reception\_error\_dma\_disable

The description of usart\_reception\_error\_dma\_disable is shown as below:

**Table 3-623. Function usart\_reception\_error\_dma\_disable**

<b>Function name</b>	usart_reception_error_dma_disable
<b>Function prototype</b>	void usart_reception_error_dma_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable DMA on reception error
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA on reception error */
```

```
usart_reception_error_dma_disable(USART0);
```

## usart\_reception\_error\_dma\_enable

The description of usart\_reception\_error\_dma\_enable is shown as below:

**Table 3-624. Function usart\_reception\_error\_dma\_enable**

<b>Function name</b>	usart_reception_error_dma_enable
<b>Function prototype</b>	void usart_reception_error_dma_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable DMA on reception error
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMA on reception error */
usart_reception_error_dma_enable(USART0);
```

## usart\_wakeup\_enable

The description of usart\_wakeup\_enable is shown as below:

**Table 3-625. Function usart\_wakeup\_enable**

<b>Function name</b>	usart_wakeup_enable
<b>Function prototype</b>	void usart_wakeup_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable USART to wakeup the mcu from deep-sleep mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 wake up enable */
usart_wakeup_enable(USART0);
```



## usart\_wakeup\_disable

The description of usart\_wakeup\_disable is shown as below:

**Table 3-626. Function usart\_wakeup\_disable**

<b>Function name</b>	usart_wakeup_disable
<b>Function prototype</b>	void usart_wakeup_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable USART to wakeup the mcu from deep-sleep mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 wake up disable */
usart_wakeup_disable(USART0);
```

## usart\_wakeup\_mode\_config

The description of usart\_wakeup\_mode\_config is shown as below:

**Table 3-627. Function usart\_wakeup\_mode\_config**

<b>Function name</b>	usart_wakeup_mode_config
<b>Function prototype</b>	void usart_wakeup_mode_config(uint32_t usart_periph, uint32_t wum);
<b>Function descriptions</b>	wakeup mode from deep-sleep mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<b>Input parameter{in}</b>	
<b>wum</b>	wakeup mode
<i>USART_WUM_ADDR</i>	WUF active on address match
<i>USART_WUM_START</i> <i>B</i>	WUF active on start bit
<i>USART_WUM_RBNE</i>	WUF active on RBNE
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 wake up mode */

usart_wakeup_mode_config(USART0, USART_WUM_ADDR);
```

### usart\_command\_enable

The description of usart\_command\_enable is shown as below:

**Table 3-628. Function usart\_command\_enable**

<b>Function name</b>	usart_command_enable
<b>Function prototype</b>	void usart_command_enable(uint32_t usart_periph, uint32_t cmdtype);
<b>Function descriptions</b>	enable USART command
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1
<b>Input parameter{in}</b>	
<b>cmdtype</b>	USART interrupt flag
USART_CMD_SBKCM D	send break command
USART_CMD_MMCM	mute mode command
USART_CMD_RXFCM D	receive data flush command
USART_CMD_TXFCM D	transmit data flush request
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 command */

usart_command_enable(USART0, USART_CMD_SBKCMD);
```

### usart\_flag\_get

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

**Table 3-629. Function usart\_flag\_get**

<b>Function name</b>	usart_flag_get
<b>Function prototype</b>	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
<b>Function descriptions</b>	get flag in STAT/RFCR register
<b>Precondition</b>	-

Input parameter{in}	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
Input parameter{in}	
<b>flag</b>	USART flags
<i>USART_FLAG_PERR</i>	parity error flag
<i>USART_FLAG_FERR</i>	frame error flag
<i>USART_FLAG_NERR</i>	noise error flag
<i>USART_FLAG_ORERR</i>	overrun error
<i>USART_FLAG_IDLE</i>	idle line detected flag
<i>USART_FLAG_RBNE</i>	read data buffer not empty
<i>USART_FLAG_TC</i>	transmission completed
<i>USART_FLAG_TBE</i>	transmit data register empty
<i>USART_FLAG_LBD</i>	LIN break detected flag
<i>USART_FLAG_CTSF</i>	CTS change flag
<i>USART_FLAG_CTS</i>	CTS level
<i>USART_FLAG_RT</i>	receiver timeout flag
<i>USART_FLAG_EB</i>	end of block flag
<i>USART_FLAG_BSY</i>	busy flag
<i>USART_FLAG_AM</i>	address match flag
<i>USART_FLAG_SB</i>	send break flag
<i>USART_FLAG_RWU</i>	receiver wakeup from mute mode
<i>USART_FLAG_WU</i>	wakeup from deep-sleep mode flag
<i>USART_FLAG_TEA</i>	transmit enable acknowledge flag
<i>USART_FLAG_REA</i>	receive enable acknowledge flag
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get flag USART0 state */
```

```
FlagStatus status;
```

```
status = usart_flag_get(USART0,USART_FLAG_TBE);
```

### usart\_flag\_clear

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

**Table 3-630. 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>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags
<i>USART_FLAG_WU</i>	wakeup from deep-sleep mode flag
<i>USART_FLAG_AM</i>	address match flag
<i>USART_FLAG_EB</i>	end of block flag
<i>USART_FLAG_RT</i>	receiver timeout flag
<i>USART_FLAG_CTSF</i>	CTS change flag
<i>USART_FLAG_LBD</i>	LIN break detected flag
<i>USART_FLAG_TC</i>	transmission complete flag
<i>USART_FLAG_RBNE</i>	read data buffer not empty
<i>USART_FLAG_IDLE</i>	idle line detected flag
<i>USART_FLAG_ORERR</i>	overrun error flag
<i>USART_FLAG_NERR</i>	noise detected flag
<i>USART_FLAG_FERR</i>	frame error flag
<i>USART_FLAG_PERR</i>	parity error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear USART0 flag */
usart_flag_clear(USART0,USART_FLAG_TC);
```

### usart\_interrupt\_enable

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

**Table 3-631. 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>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1

Input parameter{in}	
<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_AM</i>	address match interrupt
<i>USART_INT_RT</i>	receiver timeout interrupt
<i>USART_INT_EB</i>	end of block 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
<i>USART_INT_WU</i>	wakeup from deep-sleep mode interrupt
Output parameter{out}	
-	-
Return value	
-	-

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 3-632. 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>	-
Input parameter{in}	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
Input parameter{in}	
<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_AM</i>	address match interrupt
<i>USART_INT_RT</i>	receiver timeout interrupt
<i>USART_INT_EB</i>	end of block 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
<i>USART_INT_WU</i>	wakeup from deep-sleep mode 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 3-633. 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>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag
<i>USART_INT_FLAG_EB</i>	end of block interrupt and flag
<i>USART_INT_FLAG_RT</i>	receiver timeout interrupt and flag
<i>USART_INT_FLAG_A M</i>	address match interrupt and flag
<i>USART_INT_FLAG_PE RR</i>	parity error interrupt and flag
<i>USART_INT_FLAG_TB E</i>	transmitter buffer empty interrupt and flag
<i>USART_INT_FLAG_TC</i>	transmission complete interrupt and flag

USART_INT_FLAG_RBNE	read data buffer not empty interrupt and flag
USART_INT_FLAG_RBNE_ORERR	read data buffer not empty interrupt and overrun error flag
USART_INT_FLAG_IDLE	IDLE line detected interrupt and flag
USART_INT_FLAG_LBD	LIN break detected interrupt and flag
USART_INT_FLAG_WU	wakeup from deep-sleep mode interrupt and flag
USART_INT_FLAG_CTS	CTS interrupt and flag
USART_INT_FLAG_ER_NERR	error interrupt and noise error flag
USART_INT_FLAG_ER_ORERR	error interrupt and overrun error
USART_INT_FLAG_ER_FERR	error interrupt and frame error flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the USART0 interrupt flag status */
```

```
FlagStatus status;
```

```
status = usart_interrupt_flag_get(USART0, USART_INT_FLAG_RBNE);
```

### usart\_interrupt\_flag\_clear

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

**Table 3-634. Function usart\_interrupt\_flag\_clear**

Function name	usart_interrupt_flag_clear
Function prototype	void usart_interrupt_flag_clear(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
Function descriptions	clear USART interrupt flag in STAT register
Precondition	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1
Input parameter{in}	

<b>int_flag</b>	USART interrupt flag
<i>USART_INT_FLAG_PERRR</i>	parity error flag
<i>USART_INT_FLAG_ERR_NERR</i>	error interrupt and noise error flag
<i>USART_INT_FLAG_RBNE_ORERR</i>	read data buffer not empty interrupt and overrun error flag
<i>USART_INT_FLAG_ERR_ORERR</i>	error interrupt and overrun error
<i>USART_INT_FLAG_ERR_FERR</i>	error interrupt and frame error flag
<i>USART_INT_FLAG_IDLE</i>	idle line detected flag
<i>USART_INT_FLAG_TC</i>	transmission complete flag
<i>USART_INT_FLAG_LBD</i>	LIN break detected flag
<i>USART_INT_FLAG_CTS</i>	CTS change flag
<i>USART_INT_FLAG_RTS</i>	receiver timeout flag
<i>USART_INT_FLAG_EOB</i>	end of block flag
<i>USART_INT_FLAG_AM</i>	address match flag
<i>USART_INT_FLAG_WU</i>	wakeup from deep-sleep mode flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the USART0 interrupt flag */
```

```
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_TC);
```

## 3.24. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.24.1](#), the WWDGT firmware functions are introduced in chapter [3.24.2](#).



### 3.24.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

**Table 3-635. WWDGT Registers**

Registers	Descriptions
WWDGT_CTL	Control register
WWDGT_CFG	Configuration register
WWDGT_STAT	Status register

### 3.24.2. Descriptions of Peripheral functions

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

**Table 3-636. 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 3-637. Function wwdgt\_deinit**

Function name	wwdgt_deinit
Function prototype	void wwdgt_deinit(void);
Function descriptions	reset the WWDGT counter configuration
Precondition	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the WWDGT configuration */
```

```
wwdgt_deinit ();
```

## wwdgt\_enable

The description of wwdgt\_enable is shown as below:

**Table 3-638. 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 3-639. Function wwdgt\_counter\_update**

<b>Function name</b>	wwdgt_counter_update
<b>Function prototype</b>	void wwdgt_counter_update(uint16_t counter_value);
<b>Function descriptions</b>	configure the WWDGT 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 3-640. 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_DIV1</code>	the time base of WWDGT counter = (PCLK1/4096)/1
<code>WWDGT_CFG_PSC_DIV2</code>	the time base of WWDGT counter = (PCLK1/4096)/2
<code>WWDGT_CFG_PSC_DIV4</code>	the time base of WWDGT counter = (PCLK1/4096)/4
<code>WWDGT_CFG_PSC_DIV8</code>	the time base of WWDGT counter = (PCLK1/4096)/8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
```

```
wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

### **wwdgt\_interrupt\_enable**

The description of `wwdgt_interrupt_enable` is shown as below:

Table 3-641. Function `wwdgt_interrupt_enable`

<b>Function name</b>	<code>wwdgt_interrupt_enable</code>
<b>Function prototype</b>	<code>void wwdgt_interrupt_enable(void);</code>
<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 3-642. 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 ();
```

### wwdgt\_flag\_clear

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

**Table 3-643. 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	Nov.20, 2019
1.1	1. Modified the description <b><u>3.26. USART</u></b> chapter. 2. Modified the description <b><u>3.3. CAN</u></b> chapter. 3. Modified the description <b><u>3.14. I2C</u></b> chapter.	Jun.30, 2022
1.2	1. DAC CMP consistency modification	Dec.31, 2023
1.3	1. <b><u>Table 3-66. Function cmp_output_init</u></b> modified CMP_OUTPUT_TIMER0_BKIN to CMP_OUTPUT_TIMER0_BRKIN. 2. Chapter <b><u>3.9 EXTI</u></b> consistency modification.	Jan.31, 2026

## Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company according to the laws of the People's Republic of China and other applicable laws. The Company reserves all rights under such laws and no Intellectual Property Rights are transferred (either wholly or partially) or licensed by the Company (either expressly or impliedly) herein. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

To the maximum extent permitted by applicable law, the Company makes no representations or warranties of any kind, express or implied, with regard to the merchantability and the fitness for a particular purpose of the Product, nor does the Company assume any liability arising out of the application or use of any Product. Any information provided in this document is provided only for reference purposes. It is the sole responsibility of the user of this document to determine whether the Product is suitable and fit for its applications and products planned, and properly design, program, and test the functionality and safety of its applications and products planned using the Product. The Product is designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only, and the Product is not designed or intended for use in (i) safety critical applications such as weapons systems, nuclear facilities, atomic energy controller, combustion controller, aeronautic or aerospace applications, traffic signal instruments, pollution control or hazardous substance management; (ii) life-support systems, other medical equipment or systems (including life support equipment and surgical implants); (iii) automotive applications or environments, including but not limited to applications for active and passive safety of automobiles (regardless of front market or aftermarket), for example, EPS, braking, ADAS (camera/fusion), EMS, TCU, BMS, BSG, TPMS, Airbag, Suspension, DMS, ICMS, Domain, ESC, DCDC, e-clutch, advanced-lighting, etc.. Automobile herein means a vehicle propelled by a self-contained motor, engine or the like, such as, without limitation, cars, trucks, motorcycles, electric cars, and other transportation devices; and/or (iv) other uses where the failure of the device or the Product can reasonably be expected to result in personal injury, death, or severe property or environmental damage (collectively "Unintended Uses"). Customers shall take any and all actions to ensure the Product meets the applicable laws and regulations. The Company is not liable for, in whole or in part, and customers shall hereby release the Company as well as its suppliers and/or distributors from, any claim, damage, or other liability arising from or related to all Unintended Uses of the Product. Customers shall indemnify and hold the Company, and its officers, employees, subsidiaries, affiliates as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Product.

Information in this document is provided solely in connection with the Product. The Company reserves the right to make changes, corrections, modifications or improvements to this document and the Product described herein at any time without notice. The Company shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. Information in this document supersedes and replaces information previously supplied in any prior versions of this document.