

**GigaDevice Semiconductor Inc.**

**GD32C2x1**

**Arm<sup>®</sup> Cortex<sup>®</sup>-M23 32-bit MCU**

**Firmware Library  
User Guide**

Revision 1.2

(Feb. 2026)

# Table of Contents

<b>Table of Contents .....</b>	<b>1</b>
<b>List of Figures .....</b>	<b>4</b>
<b>List of Tables .....</b>	<b>5</b>
<b>1. Introduction .....</b>	<b>21</b>
<b>1.1. Rules of User Manual and Firmware Library .....</b>	<b>21</b>
1.1.1. Peripherals.....	21
1.1.2. Naming rules.....	22
<b>2. Firmware Library Overview .....</b>	<b>23</b>
<b>2.1. File Structure of Firmware Library .....</b>	<b>23</b>
2.1.1. Examples Folder .....	23
2.1.2. Firmware Folder .....	24
2.1.3. Template Folder .....	24
2.1.4. Utilities Folder .....	27
<b>2.2. File descriptions of Firmware Library .....</b>	<b>27</b>
<b>3. Firmware Library of Standard Peripherals .....</b>	<b>29</b>
<b>3.1. Overview of Firmware Library of Standard Peripherals.....</b>	<b>29</b>
<b>3.2. ADC .....</b>	<b>29</b>
3.2.1. Descriptions of Peripheral registers.....	29
3.2.2. Descriptions of Peripheral functions .....	30
<b>3.3. CMP .....</b>	<b>56</b>
3.3.1. Descriptions of Peripheral registers.....	56
3.3.2. Descriptions of Peripheral functions .....	56
<b>3.4. CRC .....</b>	<b>65</b>
3.4.1. Descriptions of Peripheral registers.....	65
3.4.2. Descriptions of Peripheral functions .....	66
<b>3.5. DBG .....</b>	<b>73</b>
3.5.1. Descriptions of Peripheral registers.....	74
3.5.2. Descriptions of Peripheral functions .....	74
<b>3.6. DMA/DMAMUX .....</b>	<b>78</b>
3.6.1. Descriptions of Peripheral registers.....	78
3.6.2. Descriptions of Peripheral functions .....	79
<b>3.7. EXTI .....</b>	<b>118</b>
3.7.1. Descriptions of Peripheral registers.....	118

3.7.2.	Descriptions of Peripheral functions .....	119
<b>3.8.</b>	<b>FMC .....</b>	<b>126</b>
3.8.1.	Descriptions of Peripheral registers .....	126
3.8.2.	Descriptions of Peripheral functions .....	127
<b>3.9.</b>	<b>FWDGT .....</b>	<b>154</b>
3.9.1.	Descriptions of Peripheral registers .....	155
3.9.2.	Descriptions of Peripheral functions .....	155
<b>3.10.</b>	<b>GPIO .....</b>	<b>160</b>
3.10.1.	Descriptions of Peripheral registers .....	160
3.10.2.	Descriptions of Peripheral functions .....	161
<b>3.11.</b>	<b>I2C .....</b>	<b>171</b>
3.11.1.	Descriptions of Peripheral registers .....	171
3.11.2.	Descriptions of Peripheral functions .....	172
<b>3.12.</b>	<b>MISC .....</b>	<b>210</b>
3.12.1.	Descriptions of Peripheral registers .....	210
3.12.2.	Descriptions of Peripheral functions .....	211
<b>3.13.</b>	<b>PMU .....</b>	<b>216</b>
3.13.1.	Descriptions of Peripheral registers .....	216
3.13.2.	Descriptions of Peripheral functions .....	216
<b>3.14.</b>	<b>RCU .....</b>	<b>226</b>
3.14.1.	Descriptions of Peripheral registers .....	226
3.14.2.	Descriptions of Peripheral functions .....	227
<b>3.15.</b>	<b>RTC .....</b>	<b>267</b>
3.15.1.	Descriptions of Peripheral registers .....	267
3.15.2.	Descriptions of Peripheral functions .....	267
<b>3.16.</b>	<b>SPI .....</b>	<b>287</b>
3.16.1.	Descriptions of Peripheral registers .....	287
3.16.2.	Descriptions of Peripheral functions .....	288
<b>3.17.</b>	<b>SYSCFG .....</b>	<b>316</b>
3.17.1.	Descriptions of Peripheral registers .....	316
3.17.2.	Descriptions of Peripheral functions .....	317
<b>3.18.</b>	<b>TIMER .....</b>	<b>328</b>
3.18.1.	Descriptions of Peripheral registers .....	328
3.18.2.	Descriptions of Peripheral functions .....	329
<b>3.19.</b>	<b>USART .....</b>	<b>388</b>
3.19.1.	Descriptions of Peripheral registers .....	389
3.19.2.	Descriptions of Peripheral functions .....	389
<b>3.20.</b>	<b>WWDGT .....</b>	<b>439</b>
3.20.1.	Descriptions of Peripheral registers .....	439

---

3.20.2. Descriptions of Peripheral functions .....	439
<b>4. Revision history .....</b>	<b>446</b>

## List of Figures

Figure 2-1. File structure of firmware library of GD32C2x1 .....	23
Figure 2-2. Select peripheral example files .....	25
Figure 2-3. Copy the peripheral example files .....	25
Figure 2-4. Open the project file .....	26
Figure 2-5. Configure project files .....	26
Figure 2-6. Compile-debug-download .....	27

## List of Tables

Table 1-1. Peripherals .....	21
Table 2-1. Function descriptions of Firmware Library .....	27
Table 3-1. Peripheral function format of Firmware Library .....	29
Table 3-2. ADC Registers .....	29
Table 3-3. ADC firmware function .....	30
Table 3-4. Function adc_deinit .....	31
Table 3-5. Function adc_enable .....	32
Table 3-6. Function adc_disable .....	32
Table 3-7. Function adc_dma_mode_enable .....	32
Table 3-8. Function adc_dma_mode_disable .....	33
Table 3-9. Function adc_discontinuous_mode_config .....	33
Table 3-10. Function adc_special_function_config .....	34
Table 3-11. Function adc_internal_channel_config .....	35
Table 3-12. Function adc_data_alignment_config .....	35
Table 3-13. Function adc_channel_length_config .....	36
Table 3-14. Function adc_routine_channel_config .....	37
Table 3-15. Function adc_inserted_channel_config .....	38
Table 3-16. Function adc_inserted_channel_offset_config .....	39
Table 3-17. Function adc_external_trigger_config .....	39
Table 3-18. Function adc_external_trigger_source_config .....	40
Table 3-19. Function adc_software_trigger_enable .....	41
Table 3-20. Function adc_routine_data_read .....	42
Table 3-21. Function adc_inserted_data_read .....	43
Table 3-22. Function adc_watchdog0_single_channel_enable .....	43
Table 3-23. Function adc_watchdog0_sequence_channel_enable .....	44
Table 3-24. Function adc_watchdog0_disable .....	44
Table 3-25. Function adc_watchdog1_channel_config .....	45
Table 3-26. Function adc_watchdog2_channel_config .....	45
Table 3-27. Function adc_watchdog1_disable .....	46
Table 3-28. Function adc_watchdog2_disable .....	47
Table 3-29. Function adc_watchdog0_threshold_config .....	47
Table 3-30. Function adc_watchdog1_threshold_config .....	48
Table 3-31. Function adc_watchdog2_threshold_config .....	48
Table 3-32. Function adc_resolution_config .....	49
Table 3-33. Function adc_oversample_mode_config .....	49
Table 3-34. Function adc_oversample_mode_enable .....	51
Table 3-35. Function adc_oversample_mode_disable .....	51
Table 3-36. Function adc_flag_get .....	52
Table 3-37. Function adc_flag_clear .....	53
Table 3-38. Function adc_interrupt_enable .....	53

Table 3-39. Function <code>adc_interrupt_disable</code> .....	54
Table 3-40. Function <code>adc_interrupt_flag_get</code> .....	54
Table 3-41. Function <code>adc_interrupt_flag_clear</code> .....	55
Table 3-42. CMP Registers .....	56
Table 3-43. CMP firmware function .....	56
Table 3-44. Enum <code>cmp_enum</code> .....	57
Table 3-45. Function <code>cmp_deinit</code> .....	57
Table 3-46. Function <code>cmp_mode_init</code> .....	57
Table 3-47. Function <code>cmp_output_init</code> .....	59
Table 3-48. Function <code>cmp_blanking_init</code> .....	59
Table 3-49. Function <code>cmp_enable</code> .....	60
Table 3-50. Function <code>cmp_disable</code> .....	61
Table 3-51. Function <code>cmp_switch_enable</code> .....	61
Table 3-52. Function <code>cmp_switch_disable</code> .....	62
Table 3-53. Function <code>cmp_lock_enable</code> .....	62
Table 3-54. Function <code>cmp_voltage_scaler_enable</code> .....	63
Table 3-55. Function <code>cmp_voltage_scaler_disable</code> .....	63
Table 3-56. Function <code>cmp_scaler_bridge_enable</code> .....	64
Table 3-57. Function <code>cmp_scaler_bridge_disable</code> .....	64
Table 3-58. Function <code>cmp_output_level_get</code> .....	65
Table 3-59. CRC Registers .....	65
Table 3-60. CRC firmware function .....	66
Table 3-61. Function <code>crc_deinit</code> .....	66
Table 3-62. Function <code>crc_init_data_register_write</code> .....	67
Table 3-63. Function <code>crc_data_register_read</code> .....	67
Table 3-64. Function <code>crc_free_data_register_read</code> .....	68
Table 3-65. Function <code>crc_free_data_register_write</code> .....	68
Table 3-66. Function <code>crc_reverse_output_data_disable</code> .....	69
Table 3-67. Function <code>crc_reverse_output_data_enable</code> .....	69
Table 3-68. Function <code>crc_input_data_reverse_config</code> .....	70
Table 3-69. Function <code>crc_data_register_reset</code> .....	70
Table 3-70. Function <code>crc_polynomial_size_set</code> .....	71
Table 3-71. Function <code>crc_polynomial_set</code> .....	71
Table 3-72. Function <code>crc_single_data_calculate</code> .....	72
Table 3-73. Function <code>crc_block_data_calculate</code> .....	73
Table 3-74. DBG Registers .....	74
Table 3-75. DBG firmware function .....	74
Table 3-76. Enum <code>dbg_periph_enum</code> .....	74
Table 3-77. Function <code>dbg_deinit</code> .....	74
Table 3-78. Function <code>dbg_id_get</code> .....	75
Table 3-79. Function <code>dbg_low_power_enable</code> .....	75
Table 3-80. Function <code>dbg_low_power_disable</code> .....	76
Table 3-81. Function <code>dbg_periph_enable</code> .....	77
Table 3-82. Function <code>dbg_periph_disable</code> .....	77

Table 3-83. DMA Registers .....	78
Table 3-84. DMAMUX Registers .....	79
Table 3-85. DMA firmware function .....	79
Table 3-86. DMAMUX firmware function.....	80
Table 3-87. Structure dma_parameter_struct.....	81
Table 3-88. Structure dmamux_sync_parameter_struct .....	81
Table 3-89. Structure dmamux_gen_parameter_struct .....	81
Table 3-90. Enum dmamux_interrupt_enum .....	81
Table 3-91. Enum dmamux_flag_enum .....	82
Table 3-92. Enum dmamux_interrupt_flag_enum.....	82
Table 3-93. Enum dma_channel_enum.....	83
Table 3-94. Enum dmamux_multiplexer_channel_enum .....	83
Table 3-95. Enum dmamux_generator_channel_enum .....	83
Table 3-96. Function dma_deinit .....	83
Table 3-97. Function dma_para_init.....	84
Table 3-98. Function dma_init .....	84
Table 3-99. Function dma_circulation_enable .....	85
Table 3-100. Function dma_circulation_disable .....	86
Table 3-101. Function dma_memory_to_memory_enable .....	86
Table 3-102. Function dma_memory_to_memory_disable .....	87
Table 3-103. Function dma_channel_enable .....	87
Table 3-104. Function dma_channel_disable .....	88
Table 3-105. Function dma_periph_address_config .....	88
Table 3-106. Function dma_memory_address_config.....	89
Table 3-107. Function dma_transfer_number_config .....	89
Table 3-108. Function dma_transfer_number_get.....	90
Table 3-109. Function dma_priority_config .....	90
Table 3-110. Function dma_memory_width_config.....	91
Table 3-111. Function dma_periph_width_config .....	92
Table 3-112. Function dma_memory_increase_enable .....	93
Table 3-113. Function dma_memory_increase_disable .....	93
Table 3-114. Function dma_periph_increase_enable.....	94
Table 3-115. Function dma_periph_increase_disable.....	94
Table 3-116. Function dma_transfer_direction_config .....	95
Table 3-117. Function dma_flag_get .....	95
Table 3-118. Function dma_flag_clear .....	96
Table 3-119. Function dma_interrupt_enable .....	97
Table 3-120. Function dma_interrupt_disable .....	97
Table 3-121. Function dma_interrupt_flag_get .....	98
Table 3-122. Function dma_interrupt_flag_clear .....	98
Table 3-123. Function dmamux_sync_struct_para_init.....	99
Table 3-124. Function dmamux_synchronization_init.....	100
Table 3-125. Function dmamux_synchronization_enable.....	101
Table 3-126. Function dmamux_synchronization_disable.....	101



Table 3-127. Function <code>dmamux_event_generation_enable</code> .....	102
Table 3-128. Function <code>dmamux_event_generation_disable</code> .....	102
Table 3-129. Function <code>dmamux_gen_struct_para_init</code> .....	103
Table 3-130. Function <code>dmamux_request_generator_init</code> .....	103
Table 3-131. Function <code>dmamux_request_generator_channel_enable</code> .....	104
Table 3-132. Function <code>dmamux_request_generator_channel_disable</code> .....	105
Table 3-133. Function <code>dmamux_synchronization_polarity_config</code> .....	105
Table 3-134. Function <code>dmamux_request_forward_number_config</code> .....	106
Table 3-135. Function <code>dmamux_sync_id_config</code> .....	107
Table 3-136. Function <code>dmamux_request_id_config</code> .....	108
Table 3-137. Function <code>dmamux_trigger_polarity_config</code> .....	110
Table 3-138. Function <code>dmamux_request_generate_number_config</code> .....	111
Table 3-139. Function <code>dmamux_trigger_id_config</code> .....	112
Table 3-140. Function <code>dmamux_flag_get</code> .....	113
Table 3-141. Function <code>dmamux_flag_clear</code> .....	114
Table 3-142. Function <code>dmamux_interrupt_enable</code> .....	115
Table 3-143. Function <code>dmamux_interrupt_disable</code> .....	116
Table 3-144. Function <code>dmamux_interrupt_flag_get</code> .....	116
Table 3-145. Function <code>dmamux_interrupt_flag_clear</code> .....	117
Table 3-146. EXTI Registers .....	118
Table 3-147. EXTI firmware function .....	119
Table 3-148. <code>exti_line_enum</code> .....	119
Table 3-149. <code>exti_mode_enum</code> .....	120
Table 3-150. <code>exti_trig_type_enum</code> .....	120
Table 3-151. Function <code>exti_deinit</code> .....	120
Table 3-152. Function <code>exti_init</code> .....	121
Table 3-153. Function <code>exti_interrupt_enable</code> .....	121
Table 3-154. Function <code>exti_interrupt_disable</code> .....	122
Table 3-155. Function <code>exti_event_enable</code> .....	122
Table 3-156. Function <code>exti_event_disable</code> .....	123
Table 3-157. Function <code>exti_software_interrupt_enable</code> .....	123
Table 3-158. Function <code>exti_software_interrupt_disable</code> .....	124
Table 3-159. Function <code>exti_flag_get</code> .....	124
Table 3-160. Function <code>exti_flag_clear</code> .....	125
Table 3-161. Function <code>exti_interrupt_flag_get</code> .....	125
Table 3-162. Function <code>exti_interrupt_flag_clear</code> .....	126
Table 3-163. FMC Registers .....	126
Table 3-164. FMC firmware function .....	127
Table 3-165. <code>fmc_state_enum</code> .....	128
Table 3-166. <code>ob_user_data_extract_info_enum</code> .....	128
Table 3-167. Function <code>fmc_unlock</code> .....	129
Table 3-168. Function <code>fmc_lock</code> .....	129
Table 3-169. Function <code>fmc_wscnt_set</code> .....	131
Table 3-170. Function <code>fmc_prefetch_enable</code> .....	131

Table 3-171. Function <code>fmc_prefetch_disable</code> .....	132
Table 3-172. Function <code>fmc_icache_enable</code> .....	132
Table 3-173. Function <code>fmc_icache_disable</code> .....	133
Table 3-174. Function <code>fmc_icache_reset_enable</code> .....	133
Table 3-175. Function <code>fmc_page_erase</code> .....	134
Table 3-176. Function <code>fmc_mass_erase</code> .....	134
Table 3-177. Function <code>fmc_doubleword_program</code> .....	135
Table 3-178. Function <code>fmc_fast_program</code> .....	136
Table 3-179. Function <code>fmc_debugger_enable</code> .....	137
Table 3-180. Function <code>fmc_debugger_disable</code> .....	137
Table 3-181. Function <code>fmc_scr_area_enable</code> .....	138
Table 3-182. Function <code>ob_unlock</code> .....	138
Table 3-183. Function <code>ob_lock</code> .....	139
Table 3-184. Function <code>ob_reload</code> .....	139
Table 3-185. Function <code>ob_user_write</code> .....	140
Table 3-186. Function <code>ob_security_protection_level_config</code> .....	142
Table 3-187. Function <code>ob_dcrp_area_config</code> .....	143
Table 3-188. Function <code>ob_write_protection_config</code> .....	144
Table 3-189. Function <code>ob_scr_area_config</code> .....	145
Table 3-190. Function <code>ob_boot_lock_config</code> .....	145
Table 3-191. Function <code>ob_user_get</code> .....	146
Table 3-192. Function <code>ob_security_protection_level_get</code> .....	146
Table 3-193. Function <code>ob_dcrp_area_get</code> .....	147
Table 3-194. Function <code>ob_write_protection_area_get</code> .....	148
Table 3-195. Function <code>ob_scr_area_get</code> .....	148
Table 3-196. Function <code>ob_boot_lock_get</code> .....	149
Table 3-197. Function <code>fmc_flag_get</code> .....	149
Table 3-198. Function <code>fmc_flag_clear</code> .....	150
Table 3-199. Function <code>fmc_interrupt_enable</code> .....	151
Table 3-200. Function <code>fmc_interrupt_disable</code> .....	151
Table 3-201. Function <code>fmc_interrupt_flag_get</code> .....	152
Table 3-202. Function <code>fmc_interrupt_flag_clear</code> .....	153
Table 3-203. Function <code>fmc_state_get</code> .....	153
Table 3-204. Function <code>fmc_ready_wait</code> .....	154
Table 3-205. FWDGT Registers .....	155
Table 3-206. FWDGT firmware function.....	155
Table 3-207. Function <code> fwdgt_write_enable</code> .....	155
Table 3-208. Function <code> fwdgt_write_disable</code> .....	156
Table 3-209. Function <code> fwdgt_enable</code> .....	156
Table 3-210. Function <code> fwdgt_prescaler_value_config</code> .....	157
Table 3-211. Function <code> fwdgt_reload_value_config</code> .....	157
Table 3-212. Function <code> fwdgt_window_value_config</code> .....	158
Table 3-213. Function <code> fwdgt_counter_reload</code> .....	158
Table 3-214. Function <code> fwdgt_config</code> .....	159

Table 3-215. Function fwdgt_flag_get.....	160
Table 3-216. GPIO Registers.....	160
Table 3-217. GPIO firmware function .....	161
Table 3-218. Function gpio_deinit .....	161
Table 3-219. Function gpio_mode_set.....	162
Table 3-220. Function gpio_output_options_set .....	163
Table 3-221. Function gpio_bit_set .....	164
Table 3-222. Function gpio_bit_reset .....	164
Table 3-223. Function gpio_bit_write.....	165
Table 3-224. Function gpio_port_write .....	165
Table 3-225. Function gpio_input_bit_get.....	166
Table 3-226. Function gpio_input_port_get.....	167
Table 3-227. Function gpio_output_bit_get .....	167
Table 3-228. Function gpio_output_port_get .....	168
Table 3-229. Function gpio_af_set .....	168
Table 3-230. Function gpio_pin_lock .....	169
Table 3-231. Function gpio_bit_toggle .....	170
Table 3-232. Function gpio_port_toggle .....	171
Table 3-233. I2C Registers .....	171
Table 3-234. I2C firmware function.....	172
Table 3-235. i2c_interrupt_flag_enum .....	173
Table 3-236. Function i2c_deinit.....	174
Table 3-237. Function i2c_timing_config .....	174
Table 3-238. Function i2c_digital_noise_filter_config .....	175
Table 3-239. Function i2c_analog_noise_filter_enable .....	176
Table 3-240. Function i2c_analog_noise_filter_disable .....	177
Table 3-241. Function i2c_master_clock_config .....	177
Table 3-242. Function i2c_master_addressing .....	178
Table 3-243. Function i2c_address10_header_enable.....	178
Table 3-244. Function i2c_address10_header_disable.....	179
Table 3-245. Function i2c_address10_enable .....	179
Table 3-246. Function i2c_address10_disable .....	180
Table 3-247. Function i2c_automatic_end_enable .....	180
Table 3-248. Function i2c_automatic_end_disable .....	181
Table 3-249. Function i2c_slave_response_to_gcall_enable .....	181
Table 3-250. Function i2c_slave_response_to_gcall_disable .....	182
Table 3-251. Function i2c_stretch_scl_low_enable .....	182
Table 3-252. Function i2c_stretch_scl_low_disable .....	183
Table 3-253. Function i2c_address_config .....	183
Table 3-254. Function i2c_address_bit_compare_config .....	184
Table 3-255. Function i2c_address_disable .....	185
Table 3-256. Function i2c_second_address_config.....	186
Table 3-257. Function i2c_second_address_disable .....	187
Table 3-258. Function i2c_received_address_get .....	187

Table 3-259. Function i2c_slave_byte_control_enable.....	188
Table 3-260. Function i2c_slave_byte_control_disable.....	188
Table 3-261. Function i2c_nack_enable .....	189
Table 3-262. Function i2c_wakeup_from_deepsleep_enable.....	189
Table 3-263. Function i2c_wakeup_from_deepsleep_disable.....	190
Table 3-264. Function i2c_enable .....	190
Table 3-265. Function i2c_disable .....	191
Table 3-266. Function i2c_start_on_bus .....	191
Table 3-267. Function i2c_stop_on_bus.....	192
Table 3-268. Function i2c_data_transmit .....	192
Table 3-269. Function i2c_data_receive .....	193
Table 3-270. Function i2c_reload_enable.....	193
Table 3-271. Function i2c_reload_disable.....	194
Table 3-272. Function i2c_transfer_byte_number_config.....	194
Table 3-273. Function i2c_dma_enable .....	195
Table 3-274. Function i2c_dma_disable .....	195
Table 3-275. Function i2c_pec_transfer .....	196
Table 3-276. Function i2c_pec_enable.....	197
Table 3-277. Function i2c_pec_disable.....	197
Table 3-278. Function i2c_pec_value_get .....	198
Table 3-279. Function i2c_smbus_alert_enable.....	198
Table 3-280. Function i2c_smbus_alert_disable.....	199
Table 3-281. Function i2c_smbus_default_addr_enable .....	199
Table 3-282. Function i2c_smbus_default_addr_disable .....	200
Table 3-283. Function i2c_smbus_host_addr_enable .....	200
Table 3-284. Function i2c_smbus_host_addr_disable .....	201
Table 3-285. Function i2c_extended_clock_timeout_enable .....	201
Table 3-286. Function i2c_extended_clock_timeout_disable.....	202
Table 3-287. Function i2c_clock_timeout_enable.....	202
Table 3-288. Function i2c_clock_timeout_disable.....	203
Table 3-289. Function i2c_bus_timeout_b_config.....	203
Table 3-290. Function i2c_bus_timeout_a_config .....	204
Table 3-291. Function i2c_idle_clock_timeout_config.....	204
Table 3-292. Function i2c_flag_get.....	205
Table 3-293. Function i2c_flag_clear.....	206
Table 3-294. Function i2c_interrupt_enable.....	207
Table 3-295. Function i2c_interrupt_disable .....	207
Table 3-296. Function i2c_interrupt_flag_get.....	208
Table 3-297. Function i2c_interrupt_flag_clear .....	209
Table 3-298. NVIC Registers .....	210
Table 3-299. SysTick Registers .....	211
Table 3-300. MISC firmware function .....	211
Table 3-301. IRQn_Type.....	211
Table 3-302. Function nvic_irq_enable.....	212

Table 3-303. Function nvic_irq_disable .....	213
Table 3-304. Function nvic_vector_table_set .....	213
Table 3-305. Function nvic_system_reset .....	214
Table 3-306. Function system_lowpower_set .....	214
Table 3-307. Function system_lowpower_reset .....	215
Table 3-308. Function systick_clksource_set .....	216
Table 3-309. PMU Registers .....	216
Table 3-310. PMU firmware function .....	217
Table 3-311. Function pmu_deinit .....	217
Table 3-312. pmu_deepsleep_voltage_select .....	218
Table 3-313. Function pmu_low_power_ldo_enable .....	218
Table 3-314. Function pmu_low_power_ldo_disable .....	219
Table 3-315. Function pmu_to_sleepmode .....	219
Table 3-316. Function pmu_to_deepsleepmode .....	220
Table 3-317. Function pmu_to_standbymode .....	220
Table 3-318. Function pmu_wakeup_pin_enable .....	221
Table 3-319. Function pmu_wakeup_pin_disable .....	221
Table 3-320. Function pmu_backup_write_enable .....	222
Table 3-321. Function pmu_backup_write_disable .....	222
Table 3-322. Function pmu_eflash_run_power_config .....	223
Table 3-323. Function pmu_eflash_deepsleep_power_config .....	224
Table 3-324. Function pmu_eflash_wakeup_time_config .....	224
Table 3-325. Function pmu_deepsleep_wait_time_config .....	225
Table 3-326. Function pmu_flag_get .....	225
Table 3-327. Function pmu_flag_clear .....	226
Table 3-328. RCU Registers .....	226
Table 3-329. RCU firmware function .....	227
Table 3-330. Enum rcu_periph_enum .....	229
Table 3-331. Enum rcu_periph_sleep_enum .....	230
Table 3-332. Enum rcu_periph_reset_enum .....	230
Table 3-333. Enum rcu_flag_enum .....	231
Table 3-334. Enum rcu_int_flag_enum .....	232
Table 3-335. Enum rcu_int_flag_clear_enum .....	232
Table 3-336. Enum rcu_int_enum .....	232
Table 3-337. Enum rcu_osci_type_enum .....	233
Table 3-338. Enum rcu_clock_freq_enum .....	233
Table 3-339. Enum usart_idx_enum .....	233
Table 3-340. Enum i2c_idx_enum .....	233
Table 3-341. Function rcu_deinit .....	233
Table 3-342. Function rcu_periph_clock_enable .....	234
Table 3-343. Function rcu_periph_clock_disable .....	235
Table 3-344. Function rcu_periph_clock_sleep_enable .....	236
Table 3-345. Function rcu_periph_clock_sleep_disable .....	237
Table 3-346. Function rcu_periph_reset_enable .....	238

Table 3-347.	Function rcu_periph_reset_disable.....	239
Table 3-348.	Function rcu_bkp_reset_enable.....	239
Table 3-349.	Function rcu_bkp_reset_disable.....	240
Table 3-350.	Function rcu_rtc_reset_enable.....	240
Table 3-351.	Function rcu_rtc_reset_disable .....	241
Table 3-352.	Function rcu_system_clock_source_config .....	241
Table 3-353.	Function rcu_system_clock_source_get .....	242
Table 3-354.	Function rcu_ahb_clock_config.....	243
Table 3-355.	Function rcu_apb_clock_config.....	243
Table 3-356.	Function rcu_adc_clock_config.....	244
Table 3-357.	Function rcu_adc_clock_config.....	245
Table 3-358.	Function rcu_ckout0_config .....	246
Table 3-359.	Function rcu_ckout1_config .....	247
Table 3-360.	Function rcu_lsckout_enable .....	248
Table 3-361.	Function rcu_lsckout_disable .....	248
Table 3-362.	Function rcu_lsckout_config.....	249
Table 3-363.	Function rcu_usart_clock_config .....	249
Table 3-364.	Function rcu_i2c_clock_config .....	250
Table 3-365.	Function rcu_i2s_clock_config.....	251
Table 3-366.	Function rcu_irc48mdiv_sys_clock_config .....	251
Table 3-367.	Function rcu_irc48mdiv_per_clock_config.....	252
Table 3-368.	Function rcu_rtc_clock_config.....	253
Table 3-369.	Function rcu_lxtal_drive_capability_config .....	254
Table 3-370.	Function rcu_osci_stab_wait.....	254
Table 3-371.	Function rcu_osci_on .....	255
Table 3-372.	Function rcu_osci_off .....	255
Table 3-373.	Function rcu_osci_start_enable.....	256
Table 3-374.	Function rcu_osci_start_disable.....	256
Table 3-375.	Function rcu_osci_bypass_mode_enable.....	257
Table 3-376.	Function rcu_osci_bypass_mode_disable.....	258
Table 3-377.	Function rcu_irc48m_adjust_value_set.....	258
Table 3-378.	Function rcu_lxtal_stab_reset_enable .....	259
Table 3-379.	Function rcu_lxtal_stab_reset_disable .....	259
Table 3-380.	Function rcu_hxtal_clock_monitor_enable .....	260
Table 3-381.	Function rcu_hxtal_clock_monitor_disable.....	260
Table 3-382.	Function rcu_lxtal_clock_monitor_enable .....	261
Table 3-383.	Function rcu_lxtal_clock_monitor_disable .....	261
Table 3-384.	Function rcu_clock_freq_get.....	262
Table 3-385.	Function rcu_flag_get .....	262
Table 3-386.	Function rcu_all_reset_flag_clear .....	263
Table 3-387.	Function rcu_interrupt_enable .....	264
Table 3-388.	Function rcu_interrupt_disable .....	264
Table 3-389.	Function rcu_interrupt_flag_get .....	265
Table 3-390.	Function rcu_interrupt_flag_clear .....	266

Table 3-391. RTC Registers .....	267
Table 3-392. RTC firmware function.....	267
Table 3-393. rtc_parameter_struct.....	268
Table 3-394. rtc_alarm_struct.....	269
Table 3-395. rtc_timestamp_struct.....	269
Table 3-396. rtc_bypass_shadow_enable .....	269
Table 3-397. rtc_bypass_shadow_disable .....	270
Table 3-398. Function rtc_register_sync_wait .....	270
Table 3-399. Function rtc_alarm_enable .....	271
Table 3-400. Function rtc_alarm_disable .....	271
Table 3-401. Function rtc_alarm_config.....	272
Table 3-402. Function rtc_alarm_subsecond_config.....	272
Table 3-403. Function rtc_alarm_get.....	274
Table 3-404. Function rtc_alarm_subsecond_get .....	274
Table 3-405. Function rtc_init_mode_enter .....	275
Table 3-406. Function rtc_init.....	275
Table 3-407. Function rtc_init_mode_exit .....	276
Table 3-408. Function rtc_current_time_get.....	277
Table 3-409. Function rtc_subsecond_get.....	277
Table 3-410. Function rtc_deinit .....	278
Table 3-411. rtc_hour_adjust .....	278
Table 3-412. rtc_second_adjust .....	279
Table 3-413. rtc_refclock_detection_enable .....	279
Table 3-414. rtc_refclock_detection_disable .....	280
Table 3-415. rtc_smooth_calibration_config .....	280
Table 3-416. Function rtc_timestamp_enable .....	281
Table 3-417. Function rtc_timestamp_disable .....	282
Table 3-418. Function rtc_timestamp_get.....	282
Table 3-419. Function rtc_timestamp_subsecond_get.....	283
Table 3-420. rtc_calibration_output_config .....	283
Table 3-421. Function rtc_alarm_output_config .....	284
Table 3-422. Function rtc_output_pin_select .....	284
Table 3-423. Function rtc_interrupt_enable .....	285
Table 3-424. Function rtc_interrupt_disable.....	286
Table 3-425. Function rtc_flag_get.....	286
Table 3-426. Function rtc_flag_clear .....	287
Table 3-427. SPI/I2S Registers .....	288
Table 3-428. SPI/I2S firmware function.....	288
Table 3-429. spi_parameter_struct.....	289
Table 3-430. Function spi_i2s_deinit .....	290
Table 3-431. Function spi_struct_para_init .....	290
Table 3-432. Function spi_init .....	291
Table 3-433. Function spi_enable .....	292
Table 3-434. Function spi_disable .....	292



Table 3-435. Function i2s_init .....	293
Table 3-436. Function i2s_psc_config .....	294
Table 3-437. Function i2s_enable .....	295
Table 3-438. Function i2s_disable .....	296
Table 3-439. Function spi_nss_output_enable .....	296
Table 3-440. Function spi_nss_output_disable .....	297
Table 3-441. Function spi_nss_internal_high .....	297
Table 3-442. Function spi_nss_internal_low .....	298
Table 3-443. Function spi_dma_enable .....	298
Table 3-444. Function spi_dma_disable .....	299
Table 3-445. Function spi_transmit_odd_config .....	299
Table 3-446. Function spi_receive_odd_config .....	300
Table 3-447. Function spi_i2s_data_frame_format_config .....	301
Table 3-448. Function spi_fifo_access_size_config .....	301
Table 3-449. Function spi_bidirectional_transfer_config.....	302
Table 3-450. Function spi_i2s_data_transmit.....	302
Table 3-451. Function spi_i2s_data_receive .....	303
Table 3-452. Function spi_crc_polynomial_set.....	304
Table 3-453. Function spi_crc_polynomial_get .....	304
Table 3-454. Function spi_crc_length_set .....	305
Table 3-455. Function spi_crc_on.....	305
Table 3-456. Function spi_crc_off .....	306
Table 3-457. Function spi_crc_next .....	306
Table 3-458. Function spi_crc_get.....	307
Table 3-459. Function spi_crc_error_clear .....	307
Table 3-460. Function spi_ti_mode_enable .....	308
Table 3-461. Function spi_ti_mode_disable .....	309
Table 3-462. Function spi_nssp_mode_enable.....	309
Table 3-463. Function spi_nssp_mode_disable.....	310
Table 3-464. Function spi_quad_enable.....	310
Table 3-465. Function spi_quad_disable.....	311
Table 3-466. Function spi_quad_write_enable.....	311
Table 3-467. Function spi_quad_read_enable.....	312
Table 3-470. Function spi_i2s_format_error_clear.....	312
Table 3-471. Function spi_i2s_flag_get .....	313
Table 3-472. Function spi_i2s_interrupt_enable .....	314
Table 3-473. Function spi_i2s_interrupt_disable.....	315
Table 3-474. Function spi_i2s_interrupt_flag_get .....	315
Table 3-475. SYSCFG Registers.....	316
Table 3-476. SYSCFG firmware function .....	317
Table 3-477. Function syscfg_deinit .....	317
Table 3-478. Function syscfg_i2c_fast_mode_plus_enable .....	318
Table 3-479. Function syscfg_i2c_fast_mode_plus_disable .....	319
Table 3-480. Function syscfg_pin_remap_enable .....	319



Table 3-481. Function syscfg_pin_remap_disable .....	320
Table 3-482. Function syscfg_bootmode_get .....	320
Table 3-483. Function syscfg_exti_line_config .....	321
Table 3-484. Function syscfg_lockup_enable .....	322
Table 3-485. Function syscfg_lockup_disable .....	322
Table 3-486. Function syscfg_sram_ecc_single_correctable_bit_get .....	323
Table 3-487. Function syscfg_sram_ecc_error_address_get .....	323
Table 3-488. Function syscfg_irq_latency_set .....	324
Table 3-489. Function syscfg_pinmux_config .....	324
Table 3-490. Function syscfg_interrupt_enable .....	325
Table 3-491. Function syscfg_interrupt_disable .....	326
Table 3-492. Function syscfg_interrupt_flag_get .....	327
Table 3-493. Function syscfg_interrupt_flag_clear .....	327
Table 3-494. TIMERx Registers .....	328
Table 3-495. TIMERx firmware function .....	329
Table 3-496. Structure timer_parameter_struct .....	331
Table 3-497. Structure timer_break_parameter_struct .....	331
Table 3-498. Structure timer_oc_parameter_struct .....	332
Table 3-499. Structure timer_ic_parameter_struct .....	332
Table 3-500. Function timer_deinit .....	333
Table 3-501. Function timer_struct_para_init .....	333
Table 3-502. Function timer_init .....	334
Table 3-503. Function timer_enable .....	335
Table 3-504. Function timer_disable .....	335
Table 3-505. Function timer_auto_reload_shadow_enable .....	336
Table 3-506. Function timer_auto_reload_shadow_disable .....	336
Table 3-507. Function timer_update_event_enable .....	337
Table 3-508. Function timer_update_event_disable .....	337
Table 3-509. Function timer_counter_alignment .....	338
Table 3-510. Function timer_counter_up_direction .....	339
Table 3-511. timer_counter_down_direction .....	339
Table 3-512. Function timer_prescaler_config .....	340
Table 3-513. Function timer_repetition_value_config .....	341
Table 3-514. Function timer_autoreload_value_config .....	341
Table 3-515. Function timer_counter_value_config .....	342
Table 3-516. Function timer_counter_read .....	342
Table 3-517. Function timer_prescaler_read .....	343
Table 3-518. Function timer_single_pulse_mode_config .....	343
Table 3-519. Function timer_update_source_config .....	344
Table 3-520. Function timer_dma_enable .....	345
Table 3-521. Function timer_dma_disable .....	346
Table 3-522. Function timer_channel_dma_request_source_select .....	347
Table 3-523. Function timer_dma_transfer_config .....	347
Table 3-524. Function timer_event_software_generate .....	349

Table 3-525. Function timer_break_struct_para_init .....	350
Table 3-526. Function timer_break_config .....	350
Table 3-527. Function timer_break_enable .....	351
Table 3-528. Function timer_break_disable .....	352
Table 3-529. Function timer_automatic_output_enable .....	352
Table 3-530. Function timer_automatic_output_disable .....	353
Table 3-531. Function timer_primary_output_config .....	353
Table 3-532. Function timer_channel_control_shadow_config .....	354
Table 3-533. Function timer_channel_control_shadow_update_config .....	355
Table 3-534. Function timer_channel_output_struct_para_init .....	355
Table 3-535. Function timer_channel_output_config .....	356
Table 3-536. Function timer_channel_output_mode_config .....	357
Table 3-537. Function timer_channel_combined_3_phase_pwm_config .....	358
Table 3-538. Function timer_channel_output_pulse_value_config .....	359
Table 3-539. Function timer_channel_output_shadow_config .....	360
Table 3-540. Function timer_channel_output_fast_config .....	361
Table 3-541. Function timer_channel_output_clear_config .....	362
Table 3-542. Function timer_channel_output_polarity_config .....	363
Table 3-543. Function timer_channel_complementary_output_polarity_config .....	364
Table 3-544. Function timer_channel_output_state_config .....	365
Table 3-545. Function timer_channel_complementary_output_state_config .....	366
Table 3-546. Function timer_channel_input_struct_para_init .....	366
Table 3-547. Function timer_input_capture_config .....	367
Table 3-548. Function timer_channel_input_capture_prescaler_config .....	368
Table 3-549. Function timer_channel_capture_value_register_read .....	369
Table 3-550. Function timer_input_pwm_capture_config .....	370
Table 3-551. Function timer_hall_mode_config .....	371
Table 3-552. Function timer_input_trigger_source_select .....	371
Table 3-553. Function timer_master_output_trigger_source_select .....	372
Table 3-554. Function timer_slave_mode_select .....	373
Table 3-555. Function timer_master_slave_mode_config .....	374
Table 3-556. Function timer_external_trigger_config .....	375
Table 3-557. Function timer_quadrature_decoder_mode_config .....	376
Table 3-558. Function timer_internal_clock_config .....	377
Table 3-559. Function timer_external_clock_mode0_config .....	378
Table 3-560. Function timer_external_clock_mode1_config .....	378
Table 3-561. Function timer_external_clock_mode1_disable .....	379
Table 3-562. Function timer_input_selection_config .....	380
Table 3-563. Function timer_write_chxval_register_config .....	381
Table 3-564. Function timer_output_value_selection_config .....	381
Table 3-565. Function timer_break_external_source_config .....	382
Table 3-566. Function timer_break_external_polarity_config .....	383
Table 3-567. Function timer_flag_get .....	383
Table 3-568. Function timer_flag_clear .....	384

Table 3-569. Function timer_interrupt_enable .....	385
Table 3-570. Function timer_interrupt_disable .....	386
Table 3-571. Function timer_interrupt_flag_get .....	387
Table 3-572. Function timer_interrupt_flag_clear .....	388
Table 3-573. USART Registers .....	389
Table 3-574. USART firmware function .....	389
Table 3-575. Enum usart_flag_enum .....	391
Table 3-576. Enum usart_interrupt_flag_enum .....	392
Table 3-577. Enum usart_interrupt_enum .....	393
Table 3-578. Enum usart_invert_enum .....	393
Table 3-579. Function usart_deinit .....	393
Table 3-580. Function usart_baudrate_set .....	394
Table 3-581. Function usart_parity_config .....	394
Table 3-582. Function usart_word_length_set .....	395
Table 3-583. Function usart_stop_bit_set .....	396
Table 3-584. Function usart_enable .....	396
Table 3-585. Function usart_disable .....	397
Table 3-586. Function usart_transmit_config .....	397
Table 3-587. Function usart_receive_config .....	398
Table 3-588. Function usart_data_first_config .....	399
Table 3-589. Function usart_invert_config .....	399
Table 3-590. Function usart_overrun_enable .....	400
Table 3-591. Function usart_overrun_disable .....	401
Table 3-592. Function usart_oversample_config .....	401
Table 3-593. Function usart_sample_bit_config .....	402
Table 3-594. Function usart_receiver_timeout_enable .....	402
Table 3-595. Function usart_receiver_timeout_disable .....	403
Table 3-596. Function usart_receiver_timeout_threshold_config .....	403
Table 3-597. Function usart_data_transmit .....	404
Table 3-598. Function usart_data_receive .....	405
Table 3-599. Function usart_command_enable .....	405
Table 3-600. Function usart_autobaud_detection_enable .....	406
Table 3-601. Function usart_autobaud_detection_disable .....	406
Table 3-602. Function usart_autobaud_detection_mode_config .....	407
Table 3-603. Function usart_address_config .....	408
Table 3-604. Function usart_address_detection_mode_config .....	408
Table 3-605. Function usart_mute_mode_enable .....	409
Table 3-606. Function usart_mute_mode_disable .....	409
Table 3-607. Function usart_mute_mode_wakeup_config .....	410
Table 3-608. Function usart_lin_mode_enable .....	410
Table 3-609. Function usart_lin_mode_disable .....	411
Table 3-610. Function usart_lin_break_detection_length_config .....	411
Table 3-611. Function usart_halfduplex_enable .....	412
Table 3-612. Function usart_halfduplex_disable .....	413

Table 3-613. Function <code>usart_clock_enable</code> .....	413
Table 3-614. Function <code>usart_clock_disable</code> .....	414
Table 3-615. Function <code>usart_synchronous_clock_config</code> .....	414
Table 3-616. Function <code>usart_guard_time_config</code> .....	415
Table 3-617. Function <code>usart_smartcard_mode_enable</code> .....	415
Table 3-618. Function <code>usart_smartcard_mode_disable</code> .....	416
Table 3-619. Function <code>usart_smartcard_mode_nack_enable</code> .....	416
Table 3-620. Function <code>usart_smartcard_mode_nack_disable</code> .....	417
Table 3-621. Function <code>usart_smartcard_mode_early_nack_enable</code> .....	417
Table 3-622. Function <code>usart_smartcard_mode_early_nack_disable</code> .....	418
Table 3-623. Function <code>usart_smartcard_autoretry_config</code> .....	419
Table 3-624. Function <code>usart_block_length_config</code> .....	419
Table 3-625. Function <code>usart_irda_mode_enable</code> .....	420
Table 3-626. Function <code>usart_irda_mode_disable</code> .....	420
Table 3-627. Function <code>usart_prescaler_config</code> .....	421
Table 3-628. Function <code>usart_irda_lowpower_config</code> .....	421
Table 3-629. Function <code>usart_hardware_flow_rts_config</code> .....	422
Table 3-630. Function <code>usart_hardware_flow_cts_config</code> .....	422
Table 3-631. Function <code>usart_hardware_flow_coherence_config</code> .....	423
Table 3-632. Function <code>usart_rs485_driver_enable</code> .....	424
Table 3-633. Function <code>usart_rs485_driver_disable</code> .....	424
Table 3-634. Function <code>usart_driver_assertime_config</code> .....	425
Table 3-635. Function <code>usart_driver_deassertime_config</code> .....	425
Table 3-636. Function <code>usart_depolarity_config</code> .....	426
Table 3-637. Function <code>usart_dma_receive_config</code> .....	427
Table 3-638. Function <code>usart_dma_transmit_config</code> .....	427
Table 3-639. Function <code>usart_reception_error_dma_disable</code> .....	428
Table 3-640. Function <code>usart_reception_error_dma_enable</code> .....	428
Table 3-641. Function <code>usart_wakeup_enable</code> .....	429
Table 3-642. Function <code>usart_wakeup_disable</code> .....	429
Table 3-643. Function <code>usart_wakeup_mode_config</code> .....	430
Table 3-644. Function <code>usart_receive_fifo_enable</code> .....	431
Table 3-645. Function <code>usart_receive_fifo_disable</code> .....	431
Table 3-646. Function <code>usart_receive_fifo_counter_number</code> .....	432
Table 3-647. Function <code>usart_flag_get</code> .....	432
Table 3-648. Function <code>usart_flag_clear</code> .....	433
Table 3-649. Function <code>usart_interrupt_enable</code> .....	434
Table 3-650. Function <code>usart_interrupt_disable</code> .....	435
Table 3-651. Function <code>usart_interrupt_flag_get</code> .....	436
Table 3-652. Function <code>usart_interrupt_flag_clear</code> .....	437
Table 3-650. WWDGT Registers .....	439
Table 3-651. WWDGT firmware function .....	439
Table 3-652. Function <code>wwdgt_deinit</code> .....	439
Table 3-653. Function <code>wwdgt_enable</code> .....	440

Table 3-654. Function <code>wwdgt_counter_update</code> .....	442
Table 3-655. Function <code>wwdgt_config</code> .....	443
Table 3-656. Function <code>wwdgt_interrupt_enable</code> .....	444
Table 3-657. Function <code>wwdgt_flag_get</code> .....	444
Table 3-658. Function <code>wwdgt_flag_clear</code> .....	445
Table 4-1. Revision history .....	446

## 1. Introduction

This manual introduces firmware library of GD32C2x1 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 GD32C2x1 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
CMP	Comparator
CRC	CRC calculation unit
DBG	Debug
DMA	Direct memory access controller

Peripherals	Descriptions
DMAMUX	DMA request multiplexer
EXTI	Interrupt/event controller
FMC	Flash memory controller
FWDGT	Free watchdog timer
GPIO/AFIO	General-purpose and alternate-function I/Os
I2C	Inter-integrated circuit interface
MISC	Nested Vectored Interrupt Controller
PMU	Power management unit
RCU	Reset and clock unit
RTC	Real-time Clock
SPI/I2S	Serial peripheral interface/Inter-IC sound
SYSCFG	System configuration
TIMER	TIMER
USART	Universal synchronous/asynchronous receiver /transmitter
WWDGT	Window watchdog timer

### 1.1.2. Naming rules

The firmware library naming rules are shown as below:

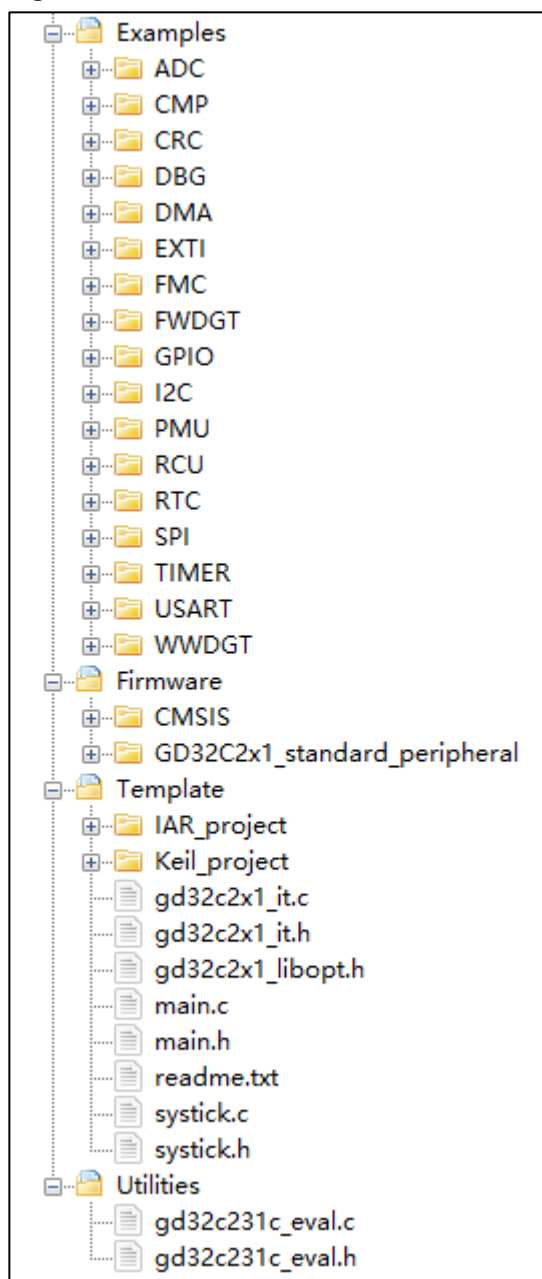
- The peripherals are shortened in XXX format, such as: ADC. More shorten information of peripherals refer to [Peripherals](#);
- The name of sourcefile and header file are started with “gd32c2x`\_”, such as: gd32c2x1\_adc.h;
- The constants used only in one file should be defined in the used file; the constants used in many files should be defined in corresponding header file. All the constants are written in uppercase of English letters;
- Registers are handled as constants. The naming of them are written in uppercase of English letters. In most cases, register names are shortened accord with the user manual;
- Variables are written in lowercase, when concluded by several words, underlines should be adapted among words;
- The naming of peripheral functions are started with the peripheral abbreviation added with an underline, when the function name is concluded by several words, underlines should be adapted among words, and all the peripheral functions are written in lowercase.

## 2. Firmware Library Overview

### 2.1. File Structure of Firmware Library

GD32C2x1\_Firmware\_Library, the file structure is shown as below:

**Figure 2-1. File structure of firmware library of GD32C2x1**



#### 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;
- GD32C2x1\_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);
- GD32C2x1\_it.c: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- GD32C2x1\_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 M23 kernel support files, the startup file based on the Cortex M23 kernel processor, the global header file of GD32C2x1 and system configuration file;
- GD32C2x1\_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.

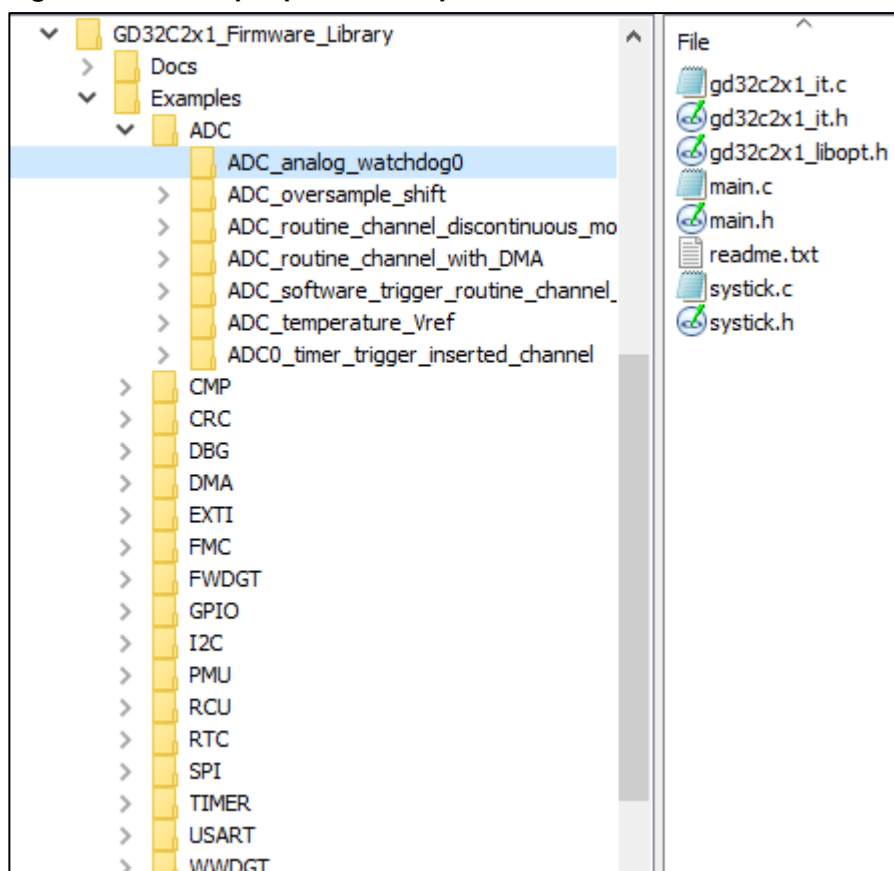
### 2.1.3. Template Folder

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, and Keil\_project is run in Keil5). User can use the project template to compile the formware examples, the steps are shown as below:

#### Select files

Open "Examples" folder, select the module to be tested, such as ADC, open "ADC" folder, select an example of ADC, such as "Analog\_watchdog", shown as below:

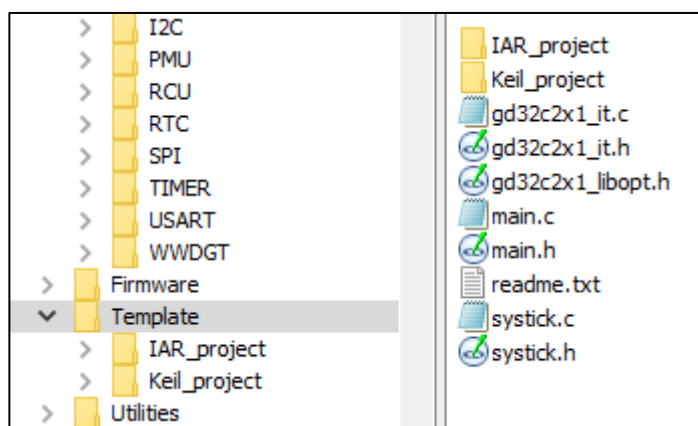
**Figure 2-2. Select peripheral example files**



## Copy files

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

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

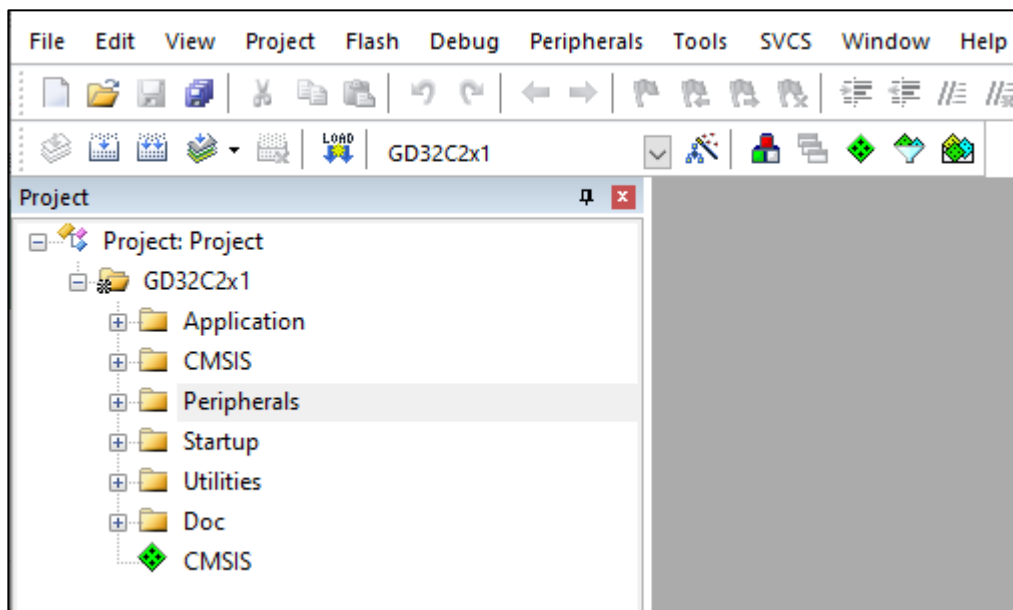


## Open a project

GD provides project in Keil and IAR, users can open project in different IDEs according to

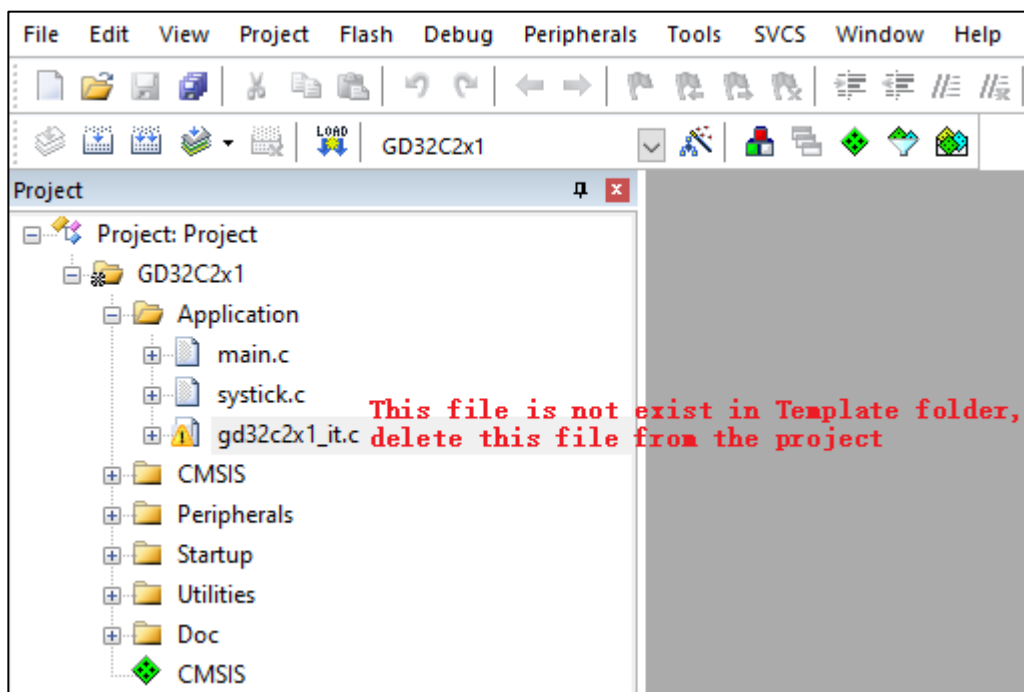
their need, such as "Keil\_project", open \Template\Keil\_project\Project.uvprojx, shown as below:

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



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

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

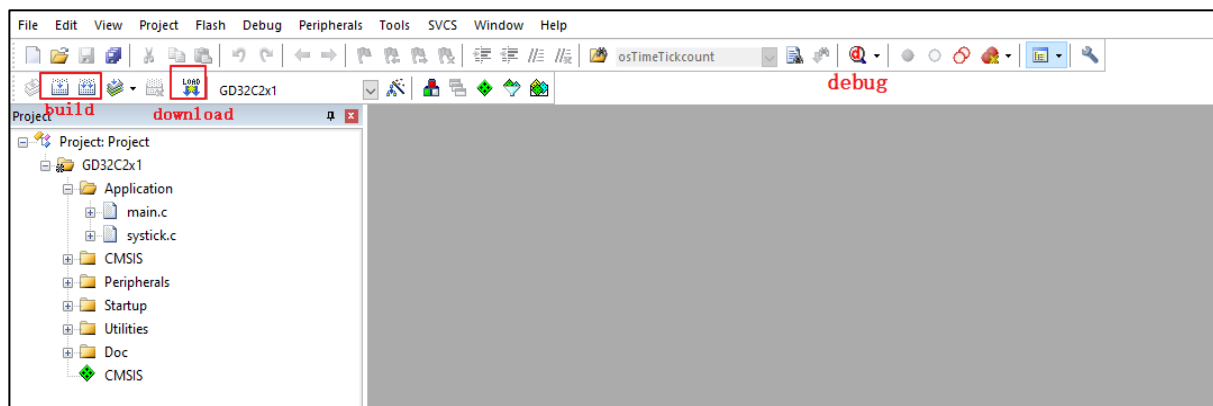


## Compile-Debug-Download

First compile the project, if there is no error, then select the right jumper cap according to the description of readme, download the project to the target board, and there will be the

phenomenon showed accord with the description of readme. The usage of IDE can refer to corresponding software user guide. If users are using Keil, the figure is shown as below:

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



## 2.1.4. Utilities Folder

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

- gd32C231C\_eval.c is related source file 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
GD32C2x1_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.
GD32C2x1_it.h	Header file, including all the prototypes of interrupt service routines.
GD32C2x1_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.
GD32C2x1_xxx.h	The header file of peripheral xxx, including functions about peripheral xxx, and the variables used for functions.
GD32C2x1_xxx.c	The C source file for driving peripheral xxx.
systick.h	The header file of systick.c, including prototypes of systick configuration

Files	Descriptions
	function and delay function.
systick.c	The source file about systick configuration function and delay function.
readme.txt	Description document about how to configure and how to use the firmware example.

## 3. Firmware Library of Standard Peripherals

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

The description format of firmware functions are shown as below:

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

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

### 3.2. ADC

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

#### 3.2.1. Descriptions of Peripheral registers

ADC registers are listed in the table shown as below:

**Table 3-2. ADC Registers**

<b>Registers</b>	<b>Descriptions</b>
ADC_STAT	Status register
ADC_CTL0	Control register 0
ADC_CTL1	Control register 1
ADC_SAMPT0	Sample time register 0
ADC_SAMPT1	Sample time register 1
ADC_IOFFx	Inserted channel data offset register x(x=0..3)
ADC_WD0HT	Watchdog 0 high threshold register

Registers	Descriptions
ADC_WD0LT	Watchdog 0 low threshold register
ADC_RSQ0	Regular sequence register 0
ADC_RSQ1	Regular sequence register 1
ADC_RSQ2	Regular sequence register 2
ADC_ISQ	Inserted sequence register
ADC_IDATAx	Inserted data register x(x=0..3)
ADC_RDATA	Regular data register
ADC_WD1SR	Watchdog 1 channel selection register
ADC_WD2SR	Watchdog 2 channel selection register
ADC_WD1HT	Watchdog 1 high threshold register
ADC_WD1LT	Watchdog 1 low threshold register
ADC_WD2HT	Watchdog 2 high threshold register
ADC_WD2LT	Watchdog 2 low threshold register
ADC_OVSAMPCTL	Oversampling control register

### 3.2.2. Descriptions of Peripheral functions

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

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

Function name	Function description
adc_deinit	reset ADC
adc_enable	enable ADC interface
adc_disable	disable ADC interface
adc_dma_mode_enable	enable DMA request
adc_dma_mode_disable	disable DMA request
adc_discontinuous_mode_config	configure ADC discontinuous mode
adc_special_function_config	configure ADC special function
adc_internal_channel_config	enable or disable ADC internal channels
adc_data_alignment_config	configure ADC data alignment
adc_channel_length_config	configure the channel length of routine sequence or inserted sequence
adc_routine_channel_config	configure ADC routine channel
adc_inserted_channel_config	configure ADC inserted channel
adc_inserted_channel_offset_config	configure ADC inserted channel offset
adc_external_trigger_config	configure ADC external trigger
adc_external_trigger_source_config	configure ADC external trigger source
adc_software_trigger_enable	enable ADC software trigger
adc_routine_data_read	read ADC routine sequence data register
adc_inserted_data_read	read ADC inserted sequence data register
adc_watchdog0_single_channel_enable	enable ADC analog watchdog 0 single channel

Function name	Function description
adc_watchdog0_sequence_channel_enable	enable ADC analog watchdog 0 sequence channel
adc_watchdog0_disable	disable ADC analog watchdog 0
adc_watchdog1_channel_config	configure ADC analog watchdog 1 channel
adc_watchdog2_channel_config	configure ADC analog watchdog 2 channel
adc_watchdog1_disable	disable ADC analog watchdog 1
adc_watchdog2_disable	disable ADC analog watchdog 2
adc_watchdog0_threshold_config	configure ADC analog watchdog 0 threshold
adc_watchdog1_threshold_config	configure ADC analog watchdog 1 threshold
adc_watchdog2_threshold_config	configure ADC analog watchdog 2 threshold
adc_resolution_config	configure ADC resolution
adc_oversample_mode_config	configure ADC oversample mode
adc_oversample_mode_enable	enable ADC oversample mode
adc_oversample_mode_disable	disable ADC oversample mode
adc_flag_get	get ADC flag
adc_flag_clear	clear ADC flag
adc_interrupt_enable	enable ADC interrupt
adc_interrupt_disable	disable ADC interrupt
adc_interrupt_flag_get	get the ADC interrupt flag
adc_interrupt_flag_clear	clear the ADC interrupt flag

## adc\_deinit

The description of adc\_deinit is shown as below:

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

Function name	adc_deinit
Function prototype	void adc_deinit(void);
Function descriptions	reset ADC
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
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**

<b>Function name</b>	adc_enable
<b>Function prototype</b>	void adc_enable(void);
<b>Function descriptions</b>	enable ADC interface
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## adc\_disable

The description of adc\_disable is shown as below:

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

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

Example:

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

## adc\_dma\_mode\_enable

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

**Table 3-7. Function adc\_dma\_mode\_enable**

<b>Function name</b>	adc_dma_mode_enable
----------------------	---------------------

Function prototype	void adc_dma_mode_enable(void);
Function descriptions	enable DMA request
Precondition	-
The called functions	-
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 3-8. Function adc\_dma\_mode\_disable**

Function name	adc_dma_mode_disable
Function prototype	void adc_dma_mode_disable(void);
Function descriptions	disable DMA request
Precondition	-
The called functions	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### adc\_discontinuous\_mode\_config

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

**Table 3-9. Function adc\_discontinuous\_mode\_config**

Function name	adc_discontinuous_mode_config
Function prototype	void adc_discontinuous_mode_config(uint8_t adc_sequence, uint8_t length);
Function descriptions	configure ADC discontinuous mode
Precondition	-
The called functions	-

Input parameter{in}	
<b>adc_sequence</b>	select the sequence
<i>ADC_ROUTINE_CHANNEL</i>	routine sequence
<i>ADC_INSERTED_CHANNEL</i>	inserted sequence
<i>ADC_CHANNEL_DISCON_DISABLE</i>	disable discontinuous mode of routine and inserted sequence
Input parameter{in}	
<b>length</b>	number of conversions in discontinuous mode,the number can be 1..8 for routine sequence, the number has no effect for inserted sequence
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC routine sequence discontinuous mode */
```

```
adc_discontinuous_mode_config(ADC_ROUTINE_CHANNEL, 6);
```

### adc\_special\_function\_config

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

**Table 3-10. 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>	configure ADC special function
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>function</b>	the function to configure
<i>ADC_SCAN_MODE</i>	scan mode select
<i>ADC_INSERTED_CHANNEL_AUTO</i>	inserted sequence convert automatically
<i>ADC_CONTINUOUS_MODE</i>	continuous mode select
Input parameter{in}	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
Output parameter{out}	

-	-
Return value	
-	-

Example:

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

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

### adc\_internal\_channel\_config

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

**Table 3-11. Function adc\_internal\_channel\_config**

<b>Function name</b>	adc_internal_channel_config
<b>Function prototype</b>	void adc_internal_channel_config(uint32_t internal_channel, ControlStatus newvalue)
<b>Function descriptions</b>	enable or disable ADC internal channels
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>internal_channel</b>	the internal channels
ADC_CHANNEL_INTE RNAL_TEMPSENSOR	temperature sensor channel
ADC_CHANNEL_INTE RNAL_VREFINT	vrefint channel
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
ENABLE	enable function
DISABLE	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC temperature sensor*/
```

```
adc_internal_channel_config(ADC_CHANNEL_INTERNAL_TEMPSENSOR, ENABLE);
```

### adc\_data\_alignment\_config

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

**Table 3-12. 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 ADC data alignment
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>data_alignment</b>	data alignment select
ADC_DATAALIGN_RIGHT	right alignment
ADC_DATAALIGN_LEFT	left alignment
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure 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 3-13. Function adc\_channel\_length\_config**

<b>Function name</b>	adc_channel_length_config
<b>Function prototype</b>	void adc_channel_length_config(uint8_t adc_sequence, uint32_t length);
<b>Function descriptions</b>	configure the channel length of routine sequence or inserted sequence
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	select the sequence
ADC_ROUTINE_CHANNEL	routine sequence
ADC_INSERTED_CHANNEL	inserted sequence
<b>Input parameter{in}</b>	
<b>length</b>	the channel length of the sequence, routine sequence 1-16, inserted sequence 1-4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the length of ADC routine sequence */
```

```
adc_channel_length_config(ADC_ROUTINE_CHANNEL, 4);
```

### adc\_routine\_channel\_config

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

**Table 3-14. Function adc\_routine\_channel\_config**

<b>Function name</b>	adc_routine_channel_config
<b>Function prototype</b>	void adc_routine_channel_config(uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
<b>Function descriptions</b>	configure ADC routine channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rank</b>	the routine sequence rank, this parameter must be between 0 to 15
<b>Input parameter{in}</b>	
<b>adc_channel</b>	the selected ADC channel
ADC_CHANNEL_x	ADC Channelx (x=0..15)
<b>Input parameter{in}</b>	
<b>sample_time</b>	the sample time value
ADC_SAMPLETIME_2POINT5	2.5 cycles
ADC_SAMPLETIME_3POINT5	3.5 cycles
ADC_SAMPLETIME_7POINT5	7.5 cycles
ADC_SAMPLETIME_12POINT5	12.5 cycles
ADC_SAMPLETIME_19POINT5	19.5 cycles
ADC_SAMPLETIME_39POINT5	39.5 cycles
ADC_SAMPLETIME_79POINT5	79.5 cycles
ADC_SAMPLETIME_160POINT5	160.5 cycles
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC routine channel */
```

```
adc_routine_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 3-15. Function adc\_inserted\_channel\_config**

<b>Function name</b>	adc_inserted_channel_config
<b>Function prototype</b>	void adc_inserted_channel_config(uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
<b>Function descriptions</b>	configure ADC inserted channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rank</b>	the inserted sequencer rank,this parameter must be between 0 to 3
<b>Input parameter{in}</b>	
<b>adc_channel</b>	the selected ADC channel
ADC_CHANNEL_x	ADC Channelx (x=0..15)
<b>Input parameter{in}</b>	
<b>sample_time</b>	the sample time value
ADC_SAMPLETIME_2POINT5	2.5 cycles
ADC_SAMPLETIME_3POINT5	3.5 cycles
ADC_SAMPLETIME_7POINT5	7.5 cycles
ADC_SAMPLETIME_12POINT5	12.5 cycles
ADC_SAMPLETIME_19POINT5	19.5 cycles
ADC_SAMPLETIME_39POINT5	39.5 cycles
ADC_SAMPLETIME_79POINT5	79.5 cycles
ADC_SAMPLETIME_160POINT5	160.5 cycles
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
adc_inserted_channel_config(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 3-16. Function adc\_inserted\_channel\_offset\_config**

<b>Function name</b>	adc_inserted_channel_offset_config
<b>Function prototype</b>	void adc_inserted_channel_offset_config(uint8_t inserted_channel, uint16_t offset);
<b>Function descriptions</b>	configure ADC inserted channel offset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>inserted_channel</b>	inserted channel select
<i>ADC_INSERTED_CHANNEL_x</i>	inserted channel, x=0,1,2,3
<b>Input parameter{in}</b>	
<b>offset</b>	the offset data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-17. Function adc\_external\_trigger\_config**

<b>Function name</b>	adc_external_trigger_config
<b>Function prototype</b>	void adc_external_trigger_config(uint8_t adc_sequence, ControlStatus newvalue);
<b>Function descriptions</b>	configure ADC external trigger
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	select the sequence



<i>ADC_ROUTINE_CHAN</i> <i>NEL</i>	routine sequence
<i>ADC_INSERTED_CHA</i> <i>NNEL</i>	inserted sequence
<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 inserted sequence external trigger */
```

```
adc_external_trigger_config(ADC_INSERTED_CHANNEL, ENABLE);
```

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

The description of `adc_external_trigger_source_config` is shown as below:

**Table 3-18. Function `adc_external_trigger_source_config`**

<b>Function name</b>	<code>adc_external_trigger_source_config</code>
<b>Function prototype</b>	<code>void adc_external_trigger_source_config(uint8_t adc_sequence, uint32_t external_trigger_source);</code>
<b>Function descriptions</b>	configure ADC external trigger source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	select the sequence
<i>ADC_ROUTINE_CHAN</i> <i>NEL</i>	routine sequence
<i>ADC_INSERTED_CHA</i> <i>NNEL</i>	inserted sequence
<b>Input parameter{in}</b>	
<b>external_trigger_source</b>	routine or inserted sequence trigger source
<i>ADC_EXTTRIG_ROUTI</i> <i>NE_T2_CH1</i>	external trigger TIMER2 CH1 event select for routine channel
<i>ADC_EXTTRIG_ROUTI</i> <i>NE_T0_CH2</i>	external trigger TIMER0 CH2 event select for routine channel
<i>ADC_EXTTRIG_ROUTI</i> <i>NE_T0_CH1</i>	external trigger TIMER0 CH1 event select for routine channel

ADC_EXTTRIG_ROUTINE_T2_TRGO	external trigger TIMER2 TRGO event select for routine channel
ADC_EXTTRIG_ROUTINE_T0_CH0	external trigger TIMER0 CH0 event select for routine channel
ADC_EXTTRIG_ROUTINE_T2_CH0	external trigger TIMER2 CH0 event select for routine channel
ADC_EXTTRIG_ROUTINE_EXTI_11	external trigger interrupt line 11 select for routine channel
ADC_EXTTRIG_ROUTINE_NONE	external trigger software event select for routine channel
ADC_EXTTRIG_INSERTED_T0_TRGO	external trigger TIMER0 TRGO event select for inserted channel
ADC_EXTTRIG_INSERTED_T0_CH3	external trigger TIMER0 CH3 event select for inserted channel
ADC_EXTTRIG_INSERTED_T13_CH0	external trigger TIMER13 CH0 event select for inserted channel
ADC_EXTTRIG_INSERTED_T2_CH3	external trigger TIMER2 CH3 event select for inserted channel
ADC_EXTTRIG_INSERTED_T2_CH2	external trigger TIMER2 CH2 event select for inserted channel
ADC_EXTTRIG_INSERTED_T15_CH0	external trigger TIMER15 CH0 event select for inserted channel
ADC_EXTTRIG_INSERTED_EXTI_15	external trigger interrupt line 15 event select for inserted channel
ADC_EXTTRIG_INSERTED_NONE	external trigger software event select for inserted channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

### adc\_software\_trigger\_enable

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

**Table 3-19. Function adc\_software\_trigger\_enable**

Function name	adc_software_trigger_enable
Function prototype	void adc_software_trigger_enable(uint8_t adc_sequence);

<b>Function descriptions</b>	enable ADC software trigger
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	select the sequence
ADC_ROUTINE_CHAN NEL	routine sequence
ADC_INSERTED_CHA NNEL	inserted sequence
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC routine sequence software trigger */
adc_software_trigger_enable(ADC_ROUTINE_CHANNEL);
```

### adc\_routine\_data\_read

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

**Table 3-20. Function adc\_routine\_data\_read**

<b>Function name</b>	adc_routine_data_read
<b>Function prototype</b>	uint16_t adc_routine_data_read(void);
<b>Function descriptions</b>	read ADC routine sequence data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	the conversion value: 0~0xFFFF

Example:

```
/* read ADC routine sequence data register */
uint16_t adc_value = 0;
adc_value = adc_routine_data_read ();
```

### adc\_inserted\_data\_read

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

Table 3-21. Function `adc_inserted_data_read`

Function name	<code>adc_inserted_data_read</code>
Function prototype	<code>uint16_t adc_inserted_data_read(uint8_t inserted_channel);</code>
Function descriptions	read ADC inserted sequence data register
Precondition	-
The called functions	-
Input parameter{in}	
<code>inserted_channel</code>	insert channel select
<code>ADC_INSERTED_CHANNEL_x</code>	inserted Channelx, x=0,1,2,3
Output parameter{out}	
-	-
Return value	
<code>uint16_t</code>	the 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_watchdog0_single_channel_enable`

The description of `adc_watchdog0_single_channel_enable` is shown as below:

Table 3-22. Function `adc_watchdog0_single_channel_enable`

Function name	<code>adc_watchdog0_single_channel_enable</code>
Function prototype	<code>void adc_watchdog0_single_channel_enable(uint8_t adc_channel);</code>
Function descriptions	enable ADC analog watchdog 0 single channel
Precondition	-
The called functions	-
Input parameter{in}	
<code>adc_channel</code>	the selected ADC channel
<code>ADC_CHANNEL_x</code>	ADC Channelx (x=0..15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC analog watchdog0 single channel */
```

```
adc_watchdog0_single_channel_enable(ADC_CHANNEL_1);
```

## adc\_watchdog0\_sequence\_channel\_enable

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

**Table 3-23. Function adc\_watchdog0\_sequence\_channel\_enable**

<b>Function name</b>	adc_watchdog0_sequence_channel_enable
<b>Function prototype</b>	void adc_watchdog0_sequence_channel_enable(uint8_t adc_sequence);
<b>Function descriptions</b>	enable ADC analog watchdog 0 sequence channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	the sequence use analog watchdog
ADC_ROUTINE_CHANNEL	routine sequence
ADC_INSERTED_CHANNEL	inserted sequence
ADC_ROUTINE_INSERTED_CHANNEL	both routine and inserted sequence
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC analog watchdog 0 sequence channel */
adc_watchdog0_sequence_channel_enable(ADC_ROUTINE_CHANNEL);
```

## adc\_watchdog0\_disable

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

**Table 3-24. Function adc\_watchdog0\_disable**

<b>Function name</b>	adc_watchdog0_disable
<b>Function prototype</b>	void adc_watchdog0_disable(void);
<b>Function descriptions</b>	disable ADC analog watchdog 0
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable ADC analog watchdog 0*/
```

```
adc_watchdog0_disable();
```

### adc\_watchdog1\_channel\_config

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

**Table 3-25. Function adc\_watchdog1\_channel\_config**

Function name	adc_watchdog1_channel_config
Function prototype	void adc_watchdog1_channel_config(uint32_t selection_channel, ControlStatus newvalue);
Function descriptions	configure ADC analog watchdog 1 channel
Precondition	-
The called functions	-
Input parameter{in}	
<b>selection_channel</b>	the channel use analog watchdog 1
ADC_AWD1_2_SELECTION_CHANNEL_x	ADC analog watchdog 1/2 select ADC channelx
ADC_ADC_AWD1_2_SELECTION_CHANNEL_ALL	ADC analog watchdog 1/2 select all ADC channel
Input parameter{in}	
<b>newvalue</b>	control value
ENABLE	enable function
DISABLE	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC analog watchdog1 channel */
```

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

### adc\_watchdog2\_channel\_config

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

**Table 3-26. Function adc\_watchdog2\_channel\_config**

Function name	adc_watchdog2_channel_config
Function prototype	void adc_watchdog2_channel_config(uint32_t selection_channel, ControlStatus newvalue);

<b>Function descriptions</b>	configure ADC analog watchdog 2 channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>selection_channel</b>	the channel use analog watchdog 2
<i>ADC_AWD1_2_SELECTION_CHANNEL_x</i>	ADC analog watchdog 1/2 select ADC channelx
<i>ADC_ADC_AWD1_2_SELECTION_CHANNEL_ALL</i>	ADC analog watchdog 1/2 select all ADC channel
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### adc\_watchdog1\_disable

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

**Table 3-27. Function adc\_watchdog1\_disable**

<b>Function name</b>	adc_watchdog1_disable
<b>Function prototype</b>	void adc_watchdog1_disable(void);
<b>Function descriptions</b>	disable ADC analog watchdog 1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable ADC analog watchdog 1*/
```

```
adc_watchdog1_disable();
```

## adc\_watchdog2\_disable

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

**Table 3-28. Function adc\_watchdog2\_disable**

<b>Function name</b>	adc_watchdog2_disable
<b>Function prototype</b>	void adc_watchdog2_disable(void);
<b>Function descriptions</b>	disable ADC analog watchdog 2
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable ADC analog watchdog 2*/
adc_watchdog2_disable();
```

## adc\_watchdog0\_threshold\_config

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

**Table 3-29. Function adc\_watchdog0\_threshold\_config**

<b>Function name</b>	adc_watchdog0_threshold_config
<b>Function prototype</b>	void adc_watchdog0_threshold_config(uint16_t low_threshold, uint16_t high_threshold);
<b>Function descriptions</b>	configure ADC analog watchdog 0 threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>low_threshold</b>	analog watchdog 0 low threshold, 0..0xFFFF
<b>Input parameter{in}</b>	
<b>high_threshold</b>	analog watchdog 0 high threshold, 0..0xFFFF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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



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

### adc\_watchdog1\_threshold\_config

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

**Table 3-30. Function adc\_watchdog1\_threshold\_config**

<b>Function name</b>	adc_watchdog1_threshold_config
<b>Function prototype</b>	void adc_watchdog1_threshold_config(uint16_t low_threshold, uint16_t high_threshold);
<b>Function descriptions</b>	configure ADC analog watchdog 1 threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>low_threshold</b>	analog watchdog 1 low threshold, 0..0xFFF
<b>Input parameter{in}</b>	
<b>high_threshold</b>	analog watchdog 1 high threshold, 0..0xFFF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
adc_watchdog1_threshold_config(0x0400, 0x0A00);
```

### adc\_watchdog2\_threshold\_config

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

**Table 3-31. Function adc\_watchdog2\_threshold\_config**

<b>Function name</b>	adc_watchdog2_threshold_config
<b>Function prototype</b>	void adc_watchdog2_threshold_config(uint16_t low_threshold, uint16_t high_threshold);
<b>Function descriptions</b>	configure ADC analog watchdog 2 threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>low_threshold</b>	analog watchdog 2 low threshold, 0..0xFFF
<b>Input parameter{in}</b>	
<b>high_threshold</b>	analog watchdog 2 high threshold, 0..0xFFF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

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

```
adc_watchdog2_threshold_config(0x0400, 0x0A00);
```

### adc\_resolution\_config

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

**Table 3-32. Function adc\_resolution\_config**

<b>Function name</b>	adc_resolution_config
<b>Function prototype</b>	void adc_resolution_config(uint32_t resolution);
<b>Function descriptions</b>	configure ADC resolution
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>resolution</b>	ADC resolution
ADC_RESOLUTION_12B	12-bit ADC resolution
ADC_RESOLUTION_10B	10-bit ADC resolution
ADC_RESOLUTION_8B	8-bit ADC resolution
ADC_RESOLUTION_6B	6-bit ADC resolution
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC resolution */
```

```
adc_resolution_config(ADC_RESOLUTION_12B);
```

### adc\_oversample\_mode\_config

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

**Table 3-33. Function adc\_oversample\_mode\_config**

<b>Function name</b>	adc_oversample_mode_config
<b>Function prototype</b>	void adc_oversample_mode_config(uint32_t mode, uint16_t shift, uint8_t ratio);

<b>Function descriptions</b>	configure ADC oversample mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mode</b>	ADC oversampling mode
ADC_OVERSAMPLING _ALL_CONVERT	all oversampled conversions for a channel are done consecutively after a trigger
ADC_OVERSAMPLING _ONE_CONVERT	each oversampled conversion for a channel needs a trigger
<b>Input parameter{in}</b>	
<b>shift</b>	ADC oversampling shift
ADC_OVERSAMPLING _SHIFT_NONE	no oversampling shift
ADC_OVERSAMPLING _SHIFT_1B	1-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_2B	2-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_3B	3-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_4B	4-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_5B	5-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_6B	6-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_7B	7-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_8B	8-bit oversampling shift
<b>Input parameter{in}</b>	
<b>ratio</b>	ADC oversampling ratio
ADC_OVERSAMPLING _RATIO_MUL2	oversampling ratio multiple 2
ADC_OVERSAMPLING _RATIO_MUL4	oversampling ratio multiple 4
ADC_OVERSAMPLING _RATIO_MUL8	oversampling ratio multiple 8
ADC_OVERSAMPLING _RATIO_MUL16	oversampling ratio multiple 16
ADC_OVERSAMPLING _RATIO_MUL32	oversampling ratio multiple 32
ADC_OVERSAMPLING	oversampling ratio multiple 64

<code>_RATIO_MUL64</code>	
<code>ADC_OVERSAMPLING_RATIO_MUL128</code>	oversampling ratio multiple 128
<code>ADC_OVERSAMPLING_RATIO_MUL256</code>	oversampling ratio multiple 256
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### **adc\_oversample\_mode\_enable**

The description of `adc_oversample_mode_enable` is shown as below:

**Table 3-34. Function `adc_oversample_mode_enable`**

<b>Function name</b>	<code>adc_oversample_mode_enable</code>
<b>Function prototype</b>	<code>void adc_oversample_mode_enable(void);</code>
<b>Function descriptions</b>	enable ADC oversample 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 ADC oversample mode */
```

```
adc_oversample_mode_enable();
```

### **adc\_oversample\_mode\_disable**

The description of `adc_oversample_mode_disable` is shown as below:

**Table 3-35. Function `adc_oversample_mode_disable`**

<b>Function name</b>	<code>adc_oversample_mode_disable</code>
<b>Function prototype</b>	<code>void adc_oversample_mode_disable(void);</code>
<b>Function descriptions</b>	disable ADC oversample 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 ADC oversample mode */
adc_oversample_mode_disable();
```

### adc\_flag\_get

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

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

<b>Function name</b>	adc_flag_get
<b>Function prototype</b>	FlagStatus adc_flag_get(uint32_t flag);
<b>Function descriptions</b>	get ADC flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	ADC flag
ADC_FLAG_WD0E	analog watchdog 0 event flag
ADC_FLAG_WD1E	analog watchdog 1 event flag
ADC_FLAG_WD2E	analog watchdog 2 event flag
ADC_FLAG_EOC	end of sequence conversion flag
ADC_FLAG_EOIC	end of inserted sequence conversion flag
ADC_FLAG_STIC	start flag of inserted sequence
ADC_FLAG_STRC	start flag of routine sequence
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the ADC analog watchdog 0 flag */
FlagStatus flag_value;

flag_value = adc_flag_get(ADC_FLAG_WD0E);
```

## adc\_flag\_clear

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

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

<b>Function name</b>	adc_flag_clear
<b>Function prototype</b>	void adc_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear ADC flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	ADC flag
ADC_FLAG_WD0E	analog watchdog 0 event flag
ADC_FLAG_WD1E	analog watchdog 1 event flag
ADC_FLAG_WD2E	analog watchdog 2 event flag
ADC_FLAG_EOC	end of sequence conversion flag
ADC_FLAG_EOIC	end of inserted sequence conversion flag
ADC_FLAG_STIC	start flag of inserted sequence
ADC_FLAG_STRC	start flag of routine sequence
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the ADC analog watchdog 0 flag */
```

```
adc_flag_clear(ADC_FLAG_WD0E);
```

## adc\_interrupt\_enable

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

**Table 3-38. 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>	ADC interrupt
ADC_INT_WD0E	analog watchdog 0 interrupt
ADC_INT_WD1E	analog watchdog 1 interrupt
ADC_INT_WD2E	analog watchdog 2 interrupt
ADC_INT_EOC	end of sequence conversion interrupt

<i>ADC_INT_EOIC</i>	end of inserted sequence conversion interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC analog watchdog 0 interrupt */
adc_interrupt_enable(ADC_INT_WD0E);
```

### adc\_interrupt\_disable

The description of `adc_interrupt_disable` is shown as below:

**Table 3-39. Function `adc_interrupt_disable`**

<b>Function name</b>	<code>adc_interrupt_disable</code>
<b>Function prototype</b>	<code>void adc_interrupt_disable(uint32_t interrupt);</code>
<b>Function descriptions</b>	disable ADC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	ADC interrupt
<i>ADC_INT_WD0E</i>	analog watchdog 0 interrupt
<i>ADC_INT_WD1E</i>	analog watchdog 1 interrupt
<i>ADC_INT_WD2E</i>	analog watchdog 2 interrupt
<i>ADC_INT_EOC</i>	end of sequence conversion interrupt
<i>ADC_INT_EOIC</i>	end of inserted sequence conversion interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable ADC analog watchdog 0 interrupt */
adc_interrupt_disable(ADC_INT_WD0E);
```

### adc\_interrupt\_flag\_get

The description of `adc_interrupt_flag_get` is shown as below:

**Table 3-40. Function `adc_interrupt_flag_get`**

<b>Function name</b>	<code>adc_interrupt_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus adc_interrupt_flag_get(uint32_t int_flag);</code>

<b>Function descriptions</b>	get ADC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	ADC interrupt flag
<i>ADC_INT_FLAG_WD0</i> <i>E</i>	analog watchdog 0 interrupt flag
<i>ADC_INT_FLAG_WD1</i> <i>E</i>	analog watchdog 1 interrupt flag
<i>ADC_INT_FLAG_WD2</i> <i>E</i>	analog watchdog 2 interrupt flag
<i>ADC_INT_FLAG_EOC</i>	end of sequence conversion interrupt flag
<i>ADC_INT_FLAG_EOIC</i>	end of inserted sequence conversion interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the ADC analog watchdog 0 interrupt flag */
```

```
FlagStatus flag_value;
```

```
flag_value = adc_interrupt_flag_get(ADC_INT_FLAG_WD0E);
```

### adc\_interrupt\_flag\_clear

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

**Table 3-41. Function adc\_interrupt\_flag\_clear**

<b>Function name</b>	adc_interrupt_flag_clear
<b>Function prototype</b>	void adc_interrupt_flag_clear(uint32_t int_flag);
<b>Function descriptions</b>	clear ADC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	ADC interrupt flag
<i>ADC_INT_FLAG_WD0</i> <i>E</i>	analog watchdog 0 interrupt flag
<i>ADC_INT_FLAG_WD1</i> <i>E</i>	analog watchdog 1 interrupt flag
<i>ADC_INT_FLAG_WD2</i> <i>E</i>	analog watchdog 2 interrupt flag
<i>ADC_INT_FLAG_EOC</i>	end of sequence conversion interrupt flag
<i>ADC_INT_FLAG_EOIC</i>	end of inserted sequence conversion interrupt flag



Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the ADC analog watchdog 0 interrupt flag */
adc_interrupt_flag_clear(ADC_INT_FLAG_WD0E);
```

### 3.3. 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. The CMP registers are listed in chapter [3.3.1](#), the CMP firmware functions are introduced in chapter [3.3.2](#).

#### 3.3.1. Descriptions of Peripheral registers

CMP registers are listed in the table shown as below:

**Table 3-42. CMP Registers**

Registers	Descriptions
CMP0_CS	CMP0 control and status register
CMP1_CS	CMP1 control and status register

#### 3.3.2. Descriptions of Peripheral functions

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

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

Function name	Function description
cmp_deinit	CMP deinit
cmp_mode_init	CMP mode init
cmp_output_init	CMP output init
cmp_blanking_init	CMP output blanking function init
cmp_enable	enable CMP
cmp_disable	disable CMP
cmp_switch_enable	enable CMP switch
cmp_switch_disable	disable CMP switch
cmp_lock_enable	lock the CMP
cmp_voltage_scaler_enable	enable the voltage scaler
cmp_voltage_scaler_disable	disable the voltage scaler

Function name	Function description
cmp_scaler_bridge_enable	enable the scaler bridge
cmp_scaler_bridge_disable	disable the scaler bridge
cmp_output_level_get	get output level

## Enum cmp\_enum

**Table 3-44. Enum cmp\_enum**

Member name	Function description
CMP0	cmoparator 0
CMP1	cmoparator 1

## cmp\_deinit

The description of cmp\_deinit is shown as below:

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

Function name	cmp_deinit
Function prototype	void cmp_deinit(cmp_enum cmp_periph);
Function descriptions	CMP deinit
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-44. Enum cmp_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

## cmp\_mode\_init

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

**Table 3-46. Function cmp\_mode\_init**

Function name	cmp_mode_init
Function prototype	void cmp_mode_init(cmp_enum cmp_periph, uint32_t operating_mode, uint32_t inverting_input, uint32_t output_hysteresis);
Function descriptions	CMP mode init
Precondition	-
The called functions	-

Input parameter{in}	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-44. Enum cmp_enum</a>
Input parameter{in}	
<b>operating_mode</b>	operating mode
<i>CMP_MODE_HIGHSPEED</i>	high speed mode
<i>CMP_MODE_MIDDLE_SPEED</i>	medium speed mode
<i>CMP_MODE_LOWSPEED</i>	low speed mode
<i>CMP_MODE_VERYLOW_SPEED</i>	very low speed mode
Input parameter{in}	
<b>inverting_input</b>	inverting input select
<i>CMP_INVERTING_INPUT_1_4VREFINT</i>	VREFINT *1/4 input
<i>CMP_INVERTING_INPUT_1_2VREFINT</i>	VREFINT *1/2 input
<i>CMP_INVERTING_INPUT_3_4VREFINT</i>	VREFINT *3/4 input
<i>CMP_INVERTING_INPUT_VREFINT</i>	VREFINT input
<i>CMP_INVERTING_INPUT_PB2_PB6</i>	PB2 input for CMP0 or PB6 input for CMP1
<i>CMP_INVERTING_INPUT_PA0_PA2</i>	PA0 input for CMP0 or PA2 input for CMP1
<i>CMP_INVERTING_INPUT_PB1_PB3</i>	PB1 input for CMP0 or PB3 input for CMP1
<i>CMP_INVERTING_INPUT_VSSA_PB4</i>	VSSA input for CMP0 or PB4 input for CMP1
Input parameter{in}	
<b>output_hysteresis</b>	hysteresis level
<i>CMP_HYSTERESIS_NONE</i>	output no hysteresis
<i>CMP_HYSTERESIS_LOW</i>	output low hysteresis
<i>CMP_HYSTERESIS_MIDDLE</i>	output middle hysteresis
<i>CMP_HYSTERESIS_HIGH</i>	output high hysteresis
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

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

```
cmp_mode_init(CMP0, CMP_MODE_HIGHSPEED, CMP_INVERTING_INPUT_1_4VREFI
NT, CMP_HYSTERESIS_NO);
```

### cmp\_output\_init

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

**Table 3-47. Function cmp\_output\_init**

<b>Function name</b>	cmp_output_init
<b>Function prototype</b>	void cmp_output_init(cmp_enum cmp_periph, uint32_t output_polarity);
<b>Function descriptions</b>	CMP output init
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-44. Enum cmp_enum</a>
<b>Input parameter{in}</b>	
<b>output_polarity</b>	CMP output polarity
<i>CMP_OUTPUT_POLA RITY_INVERTED</i>	output is inverted
<i>CMP_OUTPUT_POLA RITY_NONINVERTED</i>	output is not inverted
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### cmp\_blanking\_init

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

**Table 3-48. Function cmp\_blanking\_init**

<b>Function name</b>	cmp_blanking_init
<b>Function prototype</b>	void cmp_blanking_init(cmp_enum cmp_periph, uint32_t blanking_source_selection);
<b>Function descriptions</b>	CMP output blanking function init
<b>Precondition</b>	-

The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-44. Enum cmp_enum</a>
Input parameter{in}	
blanking_source_selection	blanking source selection
CMP_BLANKING_NONE	CMP no blanking source
CMP_BLANKING_TIMER0_OC1	CMP TIMER0_CH1 output compare signal selected as blanking source
CMP_BLANKING_TIMER2_OC1	CMP TIMER2_CH1 output compare signal selected as blanking source
CMP_BLANKING_TIMER13_OC0	CMP TIMER13_CH0 output compare signal selected as blanking source
CMP_BLANKING_TIMER15_OC0	CMP TIMER15_CH0 output compare signal selected as blanking source
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize CMP0 blanking function */
```

```
cmp_blanking_init(CMP0, CMP_BLANKING_NONE);
```

### cmp\_enable

The description of cmp\_enable is shown as below:

**Table 3-49. Function cmp\_enable**

Function name	cmp_enable
Function prototype	void cmp_enable(cmp_enum cmp_periph);
Function descriptions	enable CMP
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-44. Enum cmp_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable CMP0*/
```

```
cmp_enable(CMP0);
```

### cmp\_disable

The description of cmp\_disable is shown as below:

**Table 3-50. 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 3-44. 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 3-51. Function cmp\_switch\_enable**

<b>Function name</b>	cmp_switch_enable
<b>Function prototype</b>	cmp_switch_enable (void);
<b>Function descriptions</b>	enable CMP switch
<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 CMP0 switch */
```

```
cmp_switch_enable(CMP0);
```

### cmp\_switch\_disable

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

**Table 3-52. Function cmp\_switch\_disable**

<b>Function name</b>	cmp_switch_disable
<b>Function prototype</b>	void cmp_switch_disable(cmp_enum cmp_periph);
<b>Function descriptions</b>	disable CMP switch
<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 CMP0 switch */
cmp_switch_disable(CMP0);
```

### cmp\_lock\_enable

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

**Table 3-53. 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>	
cmp_periph	refer to enum <a href="#">Table 3-44. Enum cmp_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## cmp\_voltage\_scaler\_enable

The description of cmp\_voltage\_scaler\_enable is shown as below:

**Table 3-54. Function cmp\_voltage\_scaler\_enable**

<b>Function name</b>	cmp_voltage_scaler_enable
<b>Function prototype</b>	void cmp_voltage_scaler_enable(cmp_enum cmp_periph);
<b>Function descriptions</b>	enable the voltage scaler
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-44. Enum cmp_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable CMP0 the voltage scaler */
cmp_voltage_scaler_enable(CMP0);
```

## cmp\_voltage\_scaler\_disable

The description of cmp\_voltage\_scaler\_disable is shown as below:

**Table 3-55. Function cmp\_voltage\_scaler\_disable**

<b>Function name</b>	cmp_voltage_scaler_disable
<b>Function prototype</b>	void cmp_voltage_scaler_disable(cmp_enum cmp_periph);
<b>Function descriptions</b>	disable the voltage scaler
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-44. Enum cmp_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable CMP0 the voltage scaler */
cmp_voltage_scaler_disable(CMP0);
```



## cmp\_scaler\_bridge\_enable

The description of cmp\_scaler\_bridge\_enable is shown as below:

**Table 3-56. Function cmp\_scaler\_bridge\_enable**

<b>Function name</b>	cmp_scaler_bridge_enable
<b>Function prototype</b>	void cmp_scaler_bridge_enable(cmp_enum cmp_periph);
<b>Function descriptions</b>	enable the scaler bridge
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-44. Enum cmp_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable CMP0 the scaler bridge */
cmp_scaler_bridge_enable(CMP0);
```

## cmp\_scaler\_bridge\_disable

The description of cmp\_scaler\_bridge\_disable is shown as below:

**Table 3-57. Function cmp\_scaler\_bridge\_disable**

<b>Function name</b>	cmp_scaler_bridge_disable
<b>Function prototype</b>	void cmp_scaler_bridge_disable(cmp_enum cmp_periph);
<b>Function descriptions</b>	disable the scaler bridge
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-44. Enum cmp_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable CMP0 the scaler bridge */
cmp_scaler_bridge_disable(CMP0);
```

## cmp\_output\_level\_get

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

**Table 3-58. 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 3-44. Enum cmp_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	the output level
CMP_OUTPUTLEVEL_HIGH	comparator output high
CMP_OUTPUTLEVEL_LOW	comparator output low

Example:

```
uint32_t level;

/* get CMP0 output level */

level = cmp_output_level_get(CMP0);
```

## 3.4. CRC

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

### 3.4.1. Descriptions of Peripheral registers

CRC registers are listed in the table shown as below:

**Table 3-59. 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

Registers	Descriptions
CRC_POLY	CRC polynomial register

### 3.4.2. Descriptions of Peripheral functions

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

**Table 3-60. CRC firmware function**

Function name	Function description
crc_deinit	deinit CRC calculation unit
crc_init_data_register_write	write the initial value 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_reverse_output_data_disable	disable the reverse operation of output data
crc_reverse_output_data_enable	enable the reverse operation of output data
crc_input_data_reverse_config	configure the CRC input data function
crc_data_register_reset	reset data register to the value of initializaiton data register
crc_polynomial_size_set	configure the CRC size of polynomial function
crc_polynomial_set	configure the CRC polynomial value function
crc_single_data_calculate	CRC calculate single data
crc_block_data_calculate	CRC calculate a data array

#### **crc\_deinit**

The description of `crc_deinit` is shown as below:

**Table 3-61. Function `crc_deinit`**

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

Example:

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

## crc\_init\_data\_register\_write

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

**Table 3-62. 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\_data\_register\_read

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

**Table 3-63. Function crc\_data\_register\_read**

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

Example:

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

## crc\_free\_data\_register\_read

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

**Table 3-64. Function crc\_free\_data\_register\_read**

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

Example:

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

## crc\_free\_data\_register\_write

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

**Table 3-65. 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>	
free_data	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\_reverse\_output\_data\_disable

The description of crc\_reverse\_output\_data\_disable is shown as below:

**Table 3-66. 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\_reverse\_output\_data\_enable

The description of crc\_reverse\_output\_data\_enable is shown as below:

**Table 3-67. Function crc\_reverse\_output\_data\_enable**

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

Example:

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

```
crc_reverse_output_data_enable();
```

## crc\_input\_data\_reverse\_config

The description of crc\_input\_data\_reverse\_config is shown as below:

**Table 3-68. Function crc\_input\_data\_reverse\_config**

<b>Function name</b>	crc_input_data_reverse_config
<b>Function prototype</b>	void crc_input_data_reverse_config(uint32_t data_reverse)
<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
CRC_INPUT_DATA_NOT	input data is not reversed
CRC_INPUT_DATA_BYTE	input data is reversed on 8 bits
CRC_INPUT_DATA_HALFWORD	input data is reversed on 16 bits
CRC_INPUT_DATA_WORD	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\_data\_register\_reset

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

**Table 3-69. 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\_polynomial\_size\_set

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

**Table 3-70. Function crc\_polynomial\_size\_set**

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

Example:

```
/* configure the CRC polynomial size*/
```

```
crc_polynomial_size_set(CRC_CTL_PS_7);
```

### crc\_polynomial\_set

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

**Table 3-71. Function crc\_polynomial\_set**

<b>Function name</b>	crc_polynomial_set
<b>Function prototype</b>	void crc_polynomial_set(uint32_t poly)
<b>Function descriptions</b>	configure the CRC polynomial value function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>poly</b>	configurable polynomial value



Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

### crc\_single\_data\_calculate

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

**Table 3-72. Function crc\_single\_data\_calculate**

Function name	crc_single_data_calculate
Function prototype	uint32_t crc_single_data_calculate(uint32_t sdata, uint8_t data_format);
Function descriptions	CRC calculate single data
Precondition	-
The called functions	-
Input parameter{in}	
<b>sdata</b>	specify input data
Input parameter{in}	
<b>data_format</b>	input data format
INPUT_FORMAT_WORD	input data in word format
INPUT_FORMAT_HALFWORD	input data in half-word format
INPUT_FORMAT_BYTE	input data in byte format
Output parameter{out}	
-	-
Return value	
uint32_t	CRC calculate value (0-0xFFFFFFFF)

Example:

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

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

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

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

## crc\_block\_data\_calculate

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

**Table 3-73. Function crc\_block\_data\_calculate**

<b>Function name</b>	crc_block_data_calculate
<b>Function prototype</b>	uint32_t crc_block_data_calculate(void *array, uint32_t size, uint8_t data_format);
<b>Function descriptions</b>	CRC calculate a data array
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>array</b>	pointer to the input data array
<b>Input parameter{in}</b>	
<b>size</b>	size of the array
<b>Input parameter{in}</b>	
<b>data_format</b>	input data format
INPUT_FORMAT_WORD	input data in word format
INPUT_FORMAT_HALFWORD	input data in half-word format
INPUT_FORMAT_BYTE	input data in byte format
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	CRC calculate value (0-0xFFFFFFFF)

Example:

```

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

#define BUFFER_SIZE    6

uint32_t valcrc = 0;

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

valcrc = crc_block_data_calculate(data_buffer, BUFFER_SIZE, INPUT_FORMAT_WORD);

```

## 3.5. DBG

The DBG hold unit helps debugger to debug power saving mode. The DBG registers are listed in chapter [3.5.1](#). the DBG firmware functions are introduced in chapter [3.5.2](#).

### 3.5.1. Descriptions of Peripheral registers

DBG registers are listed in the table shown as below:

**Table 3-74. DBG Registers**

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

### 3.5.2. Descriptions of Peripheral functions

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

**Table 3-75. 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 3-76. Enum dbg\_periph\_enum**

Member name	Function description
DBG_FWDGT_HOLD	debug FWDGT kept when core is halted
DBG_WWDGT_HOLD	debug WWDGT kept when core is halted
DBG_TIMER0_HOLD	hold TIMER0 counter when core is halted
DBG_TIMER2_HOLD	hold TIMER2 counter when core is halted
DBG_TIMER13_HOLD	hold TIMER13 counter when core is halted
DBG_I2C0_HOLD	hold I2C0 smbus 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_RTC_HOLD	hold RTC counter when core is halted

#### dbg\_deinit

The description of dbg\_deinit is shown as below:

**Table 3-77. Function dbg\_deinit**

Function name	dbg_deinit
Function prototype	void dbg_deinit(void);
Function descriptions	deinitialize the DBG

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

Example:

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

```
dbg_deinit();
```

### dbg\_id\_get

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

**Table 3-78. Function dbg\_id\_get**

Function name	dbg_id_get
Function prototype	uint32_t dbg_id_get(void);
Function descriptions	Read DBG_ID code register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	DBG_ID co de (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 3-79. Function dbg\_low\_power\_enable**

Function name	dbg_low_power_enable
Function prototype	void dbg_low_power_enable(uint32_t dbg_low_power);
Function descriptions	Enable low power behavior when the mcu is in debug mode

<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 3-80. Function dbg\_low\_power\_disable**

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

Example:

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

```
dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

## dbg\_periph\_enable

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

**Table 3-81. 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 3-76. Enum dbg_periph_enum</a>
<i>DBG_FWDGT_HOLD</i>	debug FWDGT kept when core is halted
<i>DBG_WWDGT_HOLD</i>	debug WWDGT kept when core is halted
<i>DBG_TIMERx_HOLD</i>	x=0,2,13,15,16, hold TIMERx counter when core is halted
<i>DBG_I2C0_HOLD</i>	hold I2C0 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_TIMER2_HOLD);
```

## dbg\_periph\_disable

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

**Table 3-82. 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 3-76. Enum dbg_periph_enum</a>
<i>DBG_FWDGT_HOLD</i>	debug FWDGT kept when core is halted
<i>DBG_WWDGT_HOLD</i>	debug WWDGT kept when core is halted

<i>DBG_TIMERx_HOLD</i>	x=0,2,13,15,16, hold TIMERx counter when core is halted
<i>DBG_I2C0_HOLD</i>	hold I2C0 smbus when core is halted
<i>DBG_RTC_HOLD</i>	hold RTC counter when core is halted
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
dbg_periph_disable(DBG_TIMER2_HOLD);
```

## 3.6. DMA/DMAMUX

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

DMAMUX is a transmission scheduler for DMA requests. The DMAMUX request multiplexer is used for routing a DMA request line between the peripherals / generated DMA request (from the DMAMUX request generator) and the DMA controller. The DMAMUX registers are listed in chapter [3.6.1](#), the DMAMUX firmware functions are introduced in chapter [3.6.2](#).

### 3.6.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

**Table 3-83. DMA Registers**

Registers	Descriptions
DMA_INTF	Interrupt flag register
DMA_INTC	Interrupt flag clear register
DMA_CHxCTL (x=0..2)	Channel x control register
DMA_CHxCNT (x=0..2)	Channel x counter register
DMA_CHxPADDR (x=0..2)	Channel x peripheral base address register
DMA_CHxMADDR (x=0..2)	Channel x memory base address register

DMAMUX registers are listed in the table shown as below:

**Table 3-84. DMAMUX Registers**

Registers	Descriptions
DMAMUX_RM_CHx CFG (x=0..2)	Request multiplexer channel x configuration register
DMAMUX_RM_INT F	Request multiplexer channel interrupt flag register
DMAMUX_RM_INT C	Request multiplexer channel interrupt flag clear register
DMAMUX_RG_CHx CFG (x=0..3)	Request generator channel x configuration register
DMAMUX_RG_INT F	Request generator channel interrupt flag register
DMAMUX_RG_INT C	Request generator channel interrupt flag clear register

### 3.6.2. Descriptions of Peripheral functions

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

**Table 3-85. DMA firmware function**

Function name	Function description
dma_deinit	deinitialize DMA a channel registers
dma_struct_para_init	initialize the parameters of DMA struct with the default values
dma_init	initialize DMA channel
dma_circulation_enable	enable DMA circulation mode
dma_circulation_disable	disable DMA circulation mode
dma_memory_to_memory_enable	enable memory to memory mode
dma_memory_to_memory_disable	disable memory to memory mode
dma_channel_enable	enable DMA channel
dma_channel_disable	disable DMA channel
dma_periph_address_config	set DMA peripheral base address
dma_memory_address_config	set DMA memory base address
dma_transfer_number_config	set the number of remaining data to be transferred by the DMA
dma_transfer_number_get	get the number of remaining data to be transferred by the DMA
dma_priority_config	configure priority level of DMA channel
dma_memory_width_config	configure transfer data size of memory
dma_periph_width_config	configure transfer data size of peripheral
dma_memory_increase_enable	enable next address increasement algorithm of memory
dma_memory_increase_disable	disable next address increasement algorithm of memory
dma_periph_increase_enable	enable next address increasement algorithm of peripheral
dma_periph_increase_disable	disable next address increasement algorithm of peripheral



Function name	Function description
<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

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

**Table 3-86. DMAMUX firmware function**

Function name	Function description
<code>dmamux_sync_struct_para_init</code>	initialize the parameters of DMAMUX synchronization mode structure with the default values
<code>dmamux_synchronization_init</code>	initialize DMAMUX request multiplexer channel synchronization mode
<code>dmamux_synchronization_enable</code>	enable synchronization mode
<code>dmamux_synchronization_disable</code>	disable synchronization mode
<code>dmamux_event_generation_enable</code>	enable event generation
<code>dmamux_event_generation_disable</code>	disable event generation
<code>dmamux_gen_struct_para_init</code>	initialize the parameters of DMAMUX request generator structure with the default values
<code>dmamux_request_generator_init</code>	initialize DMAMUX request generator channel
<code>dmamux_request_generator_channel_enable</code>	enable DMAMUX request generator channel
<code>dmamux_request_generator_channel_disable</code>	disable DMAMUX request generator channel
<code>dmamux_synchronization_polarity_config</code>	configure synchronization input polarity
<code>dmamux_request_forward_number_config</code>	configure number of DMA requests to forward
<code>dmamux_sync_id_config</code>	configure synchronization input identification
<code>dmamux_request_id_config</code>	configure multiplexer input identification
<code>dmamux_trigger_polarity_config</code>	configure trigger input polarity
<code>dmamux_request_generate_number_config</code>	configure number of DMA requests to be generated
<code>dmamux_trigger_id_config</code>	configure trigger input identification
<code>dmamux_flag_get</code>	get DMAMUX flag
<code>dmamux_flag_clear</code>	clear DMAMUX flag
<code>dmamux_interrupt_enable</code>	enable DMAMUX interrupt
<code>dmamux_interrupt_disable</code>	disable DMAMUX interrupt
<code>dmamux_interrupt_flag_get</code>	get DMAMUX interrupt flag

Function name	Function description
dmamux_interrupt_flag_clear	clear DMAMUX interrupt flag

### Structure dma\_parameter\_struct

**Table 3-87. Structure dma\_parameter\_struct**

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

### Structure dmamux\_sync\_parameter\_struct

**Table 3-88. Structure dmamux\_sync\_parameter\_struct**

Member name	Function description
sync_id	synchronization input identification
sync_polarity	synchronization input polarity
request_number	number of DMA requests to forward

### Structure dmamux\_gen\_parameter\_struct

**Table 3-89. Structure dmamux\_gen\_parameter\_struct**

Member name	Function description
trigger_id	trigger input identification
trigger_polarity	DMAMUX request generator trigger polarity
request_number	number of DMA requests to be generated

### Enum dmamux\_interrupt\_enum

**Table 3-90. Enum dmamux\_interrupt\_enum**

Member name	Function description
DMAMUX_INT_MU_XCH0_SO	DMAMUX request multiplexer channel 0 synchronization overrun interrupt
DMAMUX_INT_MU_XCH1_SO	DMAMUX request multiplexer channel 1 synchronization overrun interrupt
DMAMUX_INT_MU_XCH2_SO	DMAMUX request multiplexer channel 2 synchronization overrun interrupt

DMAMUX_INT_GE NCH0_TO	DMAMUX request generator channel 0 trigger overrun interrupt
DMAMUX_INT_GE NCH1_TO	DMAMUX request generator channel 1 trigger overrun interrupt
DMAMUX_INT_GE NCH2_TO	DMAMUX request generator channel 2 trigger overrun interrupt
DMAMUX_INT_GE NCH3_TO	DMAMUX request generator channel 3 trigger overrun interrupt

### Enum dmamux\_flag\_enum

**Table 3-91. Enum dmamux\_flag\_enum**

Member name	Function description
DMAMUX_FLAG_M UXCH0_SO	DMAMUX request multiplexer channel 0 synchronization overrun flag
DMAMUX_FLAG_M UXCH1_SO	DMAMUX request multiplexer channel 1 synchronization overrun flag
DMAMUX_FLAG_M UXCH2_SO	DMAMUX request multiplexer channel 2 synchronization overrun flag
DMAMUX_FLAG_G ENCH0_TO	DMAMUX request generator channel 0 trigger overrun flag
DMAMUX_FLAG_G ENCH1_TO	DMAMUX request generator channel 1 trigger overrun flag
DMAMUX_FLAG_G ENCH2_TO	DMAMUX request generator channel 2 trigger overrun flag
DMAMUX_FLAG_G ENCH3_TO	DMAMUX request generator channel 3 trigger overrun flag

### Enum dmamux\_interrupt\_flag\_enum

**Table 3-92. Enum dmamux\_interrupt\_flag\_enum**

Member name	Function description
DMAMUX_INT_FL G_MUXCH0_SO	DMAMUX request multiplexer channel 0 synchronization overrun interrupt flag
DMAMUX_INT_FL G_MUXCH1_SO	DMAMUX request multiplexer channel 1 synchronization overrun interrupt flag
DMAMUX_INT_FL G_MUXCH2_SO	DMAMUX request multiplexer channel 2 synchronization overrun interrupt flag
DMAMUX_INT_FL G_GENCH0_TO	DMAMUX request generator channel 0 trigger overrun interrupt flag
DMAMUX_INT_FL G_GENCH1_TO	DMAMUX request generator channel 1 trigger overrun interrupt flag
DMAMUX_INT_FL G_GENCH2_TO	DMAMUX request generator channel 2 trigger overrun interrupt flag

DMAMUX_INT_FLG G_GENCH3_TO	DMAMUX request generator channel 3 trigger overrun interrupt flag
-------------------------------	---

## Enum dma\_channel\_enum

**Table 3-93. Enum dma\_channel\_enum**

Member name	Function description
DMA_CH0	DMA Channel 0
DMA_CH1	DMA Channel 1
DMA_CH2	DMA Channel 2

## Enum dmamux\_multiplexer\_channel\_enum

**Table 3-94. Enum dmamux\_multiplexer\_channel\_enum**

Member name	Function description
DMAMUX_MUXCH 0	DMAMUX request multiplexer Channel0
DMAMUX_MUXCH 1	DMAMUX request multiplexer Channel1
DMAMUX_MUXCH 2	DMAMUX request multiplexer Channel2

## Enum dmamux\_generator\_channel\_enum

**Table 3-95. Enum dmamux\_generator\_channel\_enum**

Member name	Function description
DMAMUX_GENCH0	DMAMUX request generator Channel0
DMAMUX_GENCH1	DMAMUX request generator Channel1
DMAMUX_GENCH2	DMAMUX request generator Channel2
DMAMUX_GENCH3	DMAMUX request generator Channel3

## dma\_deinit

The description of dma\_deinit is shown as below:

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

Function name	dma_deinit
Function prototype	void dma_deinit(dma_channel_enum channelx);
Function descriptions	deinitialize DMA a channel registers
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..2)	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* deinitialize DMA channel0 registers*/
dma_deinit(DMA_CH0);
```

### **dma\_struct\_para\_init**

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

**Table 3-97. 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>	the initialization data needed to initialize DMA channel, refer to <a href="#">Table 3-87. 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 3-98. 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>	DMA channel

<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-87. Structure dma_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* DMA channel0 initialize */
dma_parameter_struct dma_init_struct;

dma_struct_para_init(&dma_init_struct);
dma_init_struct.direction = DMA_PERIPHERAL_TO_MEMORY;
dma_init_struct.memory_addr = (uint32_t)g_destbuf;
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_struct.memory_width = DMA_MEMORY_WIDTH_8BIT;
dma_init_struct.number = TRANSFER_NUM;
dma_init_struct.periph_addr = (uint32_t)BANK0_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_8BIT;
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_init(DMA_CH0, dma_init_struct);

```

## dma\_circulation\_enable

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

**Table 3-99. Function dma\_circulation\_enable**

<b>Function name</b>	dma_circulation_enable
<b>Function prototype</b>	void dma_circulation_enable(dma_channel_enum channelx);
<b>Function descriptions</b>	enable DMA circulation mode
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-100. 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>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<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 3-101. 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>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<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 3-102. Function dma\_memory\_to\_memory\_disable**

<b>Function name</b>	dma_memory_to_memory_disable
<b>Function prototype</b>	void dma_memory_to_memory_disable(dma_channel_enum channelx);
<b>Function descriptions</b>	disable memory to memory mode
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-103. Function dma\_channel\_enable**

<b>Function name</b>	dma_channel_enable
<b>Function prototype</b>	void dma_channel_enable(dma_channel_enum channelx);
<b>Function descriptions</b>	enable DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMA channel0 */
dma_channel_enable(DMA_CH0);
```



## dma\_channel\_disable

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

**Table 3-104. Function dma\_channel\_disable**

<b>Function name</b>	dma_channel_disable
<b>Function prototype</b>	void dma_channel_disable(dma_channel_enum channelx);
<b>Function descriptions</b>	disable DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-105. Function dma\_periph\_address\_config**

<b>Function name</b>	dma_periph_address_config
<b>Function prototype</b>	void dma_periph_address_config(dma_channel_enum channelx, uint32_t address);
<b>Function descriptions</b>	set DMA peripheral base address
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>address</b>	peripheral base address, 0 – 0xFFFFFFFF
<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 3-106. 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>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>address</b>	memory base address, 0 – 0xFFFFFFFF
<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 3-107. 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>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>

Input parameter{in}	
number	data transfer number(0x0-0xFFFF)
Output parameter{out}	
-	-
Return value	
-	-

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 3-108. Function dma\_transfer\_number\_get**

Function name	dma_transfer_number_get
Function prototype	uint32_t dma_transfer_number_get(dma_channel_enum channelx);
Function descriptions	get the number of remaining data to be transferred by the DMA
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..2)	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
Output parameter{out}	
-	-
Return value	
uint32_t	DMA data transmission remaining quantity (0x0-0xFFFF)

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 3-109. 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	corresponding channel enable bit CHEN should be 0

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>priority</b>	priority Level of this channel
<i>DMA_PRIORITY_LOW</i>	low priority
<i>DMA_PRIORITY_MEDIUM</i>	medium priority
<i>DMA_PRIORITY_HIGH</i>	high priority
<i>DMA_PRIORITY_ULTRA_HIGH</i>	ultra high priority
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* 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 3-110. Function dma\_memory\_width\_config**

<b>Function name</b>	dma_memory_width_config
<b>Function prototype</b>	void dma_memory_width_config( dma_channel_enum channelx, uint32_t mwidth);
<b>Function descriptions</b>	configure transfer data size of memory
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>mwidth</b>	transfer data width of memory
<i>DMA_MEMORY_WIDTH_8BIT</i>	transfer data width of memory is 8-bit
<i>DMA_MEMORY_WIDTH_16BIT</i>	transfer data width of memory is 16-bit
<i>DMA_MEMORY_WIDTH_32BIT</i>	transfer data width of memory is 32-bit

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 3-111. 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	corresponding channel enable bit CHEN should be 0
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..2)	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
Input parameter{in}	
pwidth	transfer data width of peripheral
DMA_PERIPHERAL_WIDTH_8BIT	transfer data width of peripheral is 8-bit
DMA_PERIPHERAL_WIDTH_16BIT	transfer data width of peripheral is 16-bit
DMA_PERIPHERAL_WIDTH_32BIT	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 3-112. 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>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-113. 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>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<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 3-114. 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>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable next address increasement algorithm of DMA channel0 */
dma_periph_increase_enable(DMA_CH0);
```

### dma\_periph\_increase\_disable

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

Table 3-115. 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>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable next address increasement algorithm of DMA channel0 */
dma_periph_increase_disable(DMA_CH0);
```

### dma\_transfer\_direction\_config

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

Table 3-116. Function dma\_transfer\_direction\_config

Function name	dma_transfer_direction_config
Function prototype	void dma_transfer_direction_config(dma_channel_enum channelx, uint32_t direction);
Function descriptions	configure the direction of data transfer on the channel
Precondition	corresponding channel enable bit CHEN should be 0
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..6)	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
Input parameter{in}	
direction	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
Output parameter{out}	
-	-
Return value	
-	-

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 3-117. Function dma\_flag\_get

Function name	dma_flag_get
Function prototype	FlagStatus dma_flag_get(dma_channel_enum channelx, uint32_t flag);
Function descriptions	check DMA flag is set or not
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..2)	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
Input parameter{in}	
flag	specify get which flag
DMA_FLAG_G	global interrupt flag of channel
DMA_FLAG_FTF	full transfer finish flag of channel
DMA_FLAG_HTF	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 3-118. 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>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>DMA_FLAG_G</i>	global interrupt flag of channel
<i>DMA_FLAG_FTF</i>	full transfer finish flag of channel
<i>DMA_FLAG_HTF</i>	half transfer finish flag of channel
<i>DMA_FLAG_ERR</i>	error flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* 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 3-119. 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>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>source</b>	DMA interrupt source
<i>DMA_INT_FTF</i>	full transfer finish interrupt of channel
<i>DMA_INT_HTF</i>	half transfer finish interrupt of channel
<i>DMA_INT_ERR</i>	error interrupt of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* 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 3-120. 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>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>source</b>	DMA interrupt source
<i>DMA_INT_FTF</i>	full transfer finish interrupt of channel
<i>DMA_INT_HTF</i>	half transfer finish interrupt of channel
<i>DMA_INT_ERR</i>	error interrupt of channel
<b>Output parameter{out}</b>	
-	-

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 3-121. Function dma\_interrupt\_flag\_get**

Function name	dma_interrupt_flag_get
Function prototype	FlagStatus dma_interrupt_flag_get(dma_channel_enum channelx, uint32_t flag);
Function descriptions	check DMA flag and interrupt enable bit is set or not
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..2)	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
Input parameter{in}	
flag	specify get which flag
DMA_INT_FLAG_FTF	full transfer finish interrupt flag of channel
DMA_INT_FLAG_HTF	half transfer finish interrupt flag of channel
DMA_INT_FLAG_ERR	error interrupt flag of channel
Output parameter{out}	
-	-
Return value	
FlagStatus	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

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

**Table 3-122. Function dma\_interrupt\_flag\_clear**

Function name	dma_interrupt_flag_clear
Function prototype	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>	DMA channel
<i>DMA_CHx(x=0..2)</i>	DMA channel selection, refer to <a href="#">Table 3-93. Enum dma_channel_enum</a>
<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);
}

```

### dmamux\_sync\_struct\_para\_init

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

**Table 3-123. Function dmamux\_sync\_struct\_para\_init**

<b>Function name</b>	dmamux_sync_struct_para_init
<b>Function prototype</b>	void dmamux_sync_struct_para_init(dmamux_sync_parameter_struct *init_struct);
<b>Function descriptions</b>	initialize the parameters of DMAMUX synchronization mode structure with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>init_struct</b>	the initialization data needed to initialize DMAMUX request multiplexer channel synchronization mode, refer to <a href="#">Table 3-88. Structure dmamux_sync_parameter_struct</a>
<b>Return value</b>	
-	-

Example:

```
/* initialize DMAMUX synchronization mode structure */
dmamux_sync_parameter_struct dmamux_sync_init_struct;
dmamux_sync_struct_para_init(&dmamux_sync_init_struct);
```

### dmamux\_synchronization\_init

The description of dmamux\_synchronization\_init is shown as below:

**Table 3-124. Function dmamux\_synchronization\_init**

<b>Function name</b>	dmamux_synchronization_init
<b>Function prototype</b>	void dmamux_synchronization_init(dmamux_multiplexer_channel_enum channelx, dmamux_sync_parameter_struct *init_struct);
<b>Function descriptions</b>	initialize DMAMUX request multiplexer channel synchronization mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMAMUX request multiplexer channel is initialized
DMAMUX_MUXCHx(x=0..2)	DMAMUX channel selection, refer to <a href="#">Table 3-94. Enum dmamux_multiplexer_channel_enum</a>
<b>Input parameter{in}</b>	
<b>init_struct</b>	the initialization data needed to initialize DMAMUX request multiplexer channel synchronization mode, refer to <a href="#">Table 3-88. Structure dmamux_sync_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize DMAMUX synchronization mode structure */
dmamux_sync_parameter_struct dmamux_sync_init_struct;
dmamux_sync_struct_para_init(&dmamux_sync_init_struct);
/* initialize DMA request multiplexer channel 0 with synchronization mode */
dmamux_sync_init_struct.sync_id      = DMAMUX_SYNC_EXTI0;
dmamux_sync_init_struct.sync_polarity = DMAMUX_SYNC_RISING;
dmamux_sync_init_struct.request_number = 4;
dmamux_synchronization_init(DMAMUX_MUXCH0, &dmamux_sync_init_struct);
```

### dmamux\_synchronization\_enable

The description of dmamux\_synchronization\_enable is shown as below:

Table 3-125. Function dmamux\_synchronization\_enable

Function name	dmamux_synchronization_enable
Function prototype	void dmamux_synchronization_enable(dmamux_multiplexer_channel_enum channelx);
Function descriptions	enable synchronization mode
Precondition	-
The called functions	-
Input parameter{in}	
channelx	specify which DMAMUX request multiplexer channel is initialized
DMAMUX_MUXCHx(x=0..2)	DMAMUX channel selection, refer to <a href="#">Table 3-94. Enum dmamux_multiplexer_channel_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable synchronization mode */
dmamux_synchronization_enable(DMAMUX_MUXCH0);
```

### dmamux\_synchronization\_disable

The description of dmamux\_synchronization\_disable is shown as below:

Table 3-126. Function dmamux\_synchronization\_disable

Function name	dmamux_synchronization_disable
Function prototype	void dmamux_synchronization_disable(dmamux_multiplexer_channel_enum channelx);
Function descriptions	disable synchronization mode
Precondition	-
The called functions	-
Input parameter{in}	
channelx	specify which DMAMUX request multiplexer channel is initialized
DMAMUX_MUXCHx(x=0..2)	DMAMUX channel selection, refer to <a href="#">Table 3-94. Enum dmamux_multiplexer_channel_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable synchronization mode */
dmamux_synchronization_disable(DMAMUX_MUXCH0);
```

## dmamux\_event\_generation\_enable

The description of dmamux\_event\_generation\_enable is shown as below:

**Table 3-127. Function dmamux\_event\_generation\_enable**

<b>Function name</b>	dmamux_event_generation_enable
<b>Function prototype</b>	void dmamux_event_generation_enable(dmamux_mux_channel_enum channelx);
<b>Function descriptions</b>	enable event generation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMAMUX request multiplexer channel is initialized
DMAMUX_MUXCHx(x=0..2)	DMAMUX channel selection, refer to <a href="#">Table 3-94. Enum dmamux_mux_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable event generation */
dmamux_event_generation_enable(DMAMUX_MUXCH0);
```

## dmamux\_event\_generation\_disable

The description of dmamux\_event\_generation\_disable is shown as below:

**Table 3-128. Function dmamux\_event\_generation\_disable**

<b>Function name</b>	dmamux_event_generation_disable
<b>Function prototype</b>	void dmamux_event_generation_disable(dmamux_mux_channel_enum channelx);
<b>Function descriptions</b>	disable event generation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMAMUX request multiplexer channel is initialized
DMAMUX_MUXCHx(x=0..2)	DMAMUX channel selection, refer to <a href="#">Table 3-94. Enum dmamux_mux_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* disable event generation */
dmamux_event_generation_disable(DMAMUX_MUXCH0);
```

### dmamux\_gen\_struct\_para\_init

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

**Table 3-129. Function dmamux\_gen\_struct\_para\_init**

<b>Function name</b>	dmamux_gen_struct_para_init
<b>Function prototype</b>	void dmamux_gen_struct_para_init(dmamux_gen_parameter_struct *init_struct);
<b>Function descriptions</b>	initialize the parameters of DMAMUX request generator structure with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>init_struct</b>	the initialization data needed to initialize DMAMUX request generator channel, refer to <a href="#">Table 3-89. Structure dmamux_gen_parameter_struct</a>
<b>Return value</b>	
-	-

Example:

```
/* initialize DMA request generator structure */
dmamux_gen_parameter_struct dmamux_gen_init_struct;
dmamux_gen_struct_para_init(&dmamux_gen_init_struct);
```

### dmamux\_request\_generator\_init

The description of dmamux\_request\_generator\_init is shown as below:

**Table 3-130. Function dmamux\_request\_generator\_init**

<b>Function name</b>	dmamux_request_generator_init
<b>Function prototype</b>	void dmamux_request_generator_init(dmamux_generator_channel_enum channelx, dmamux_gen_parameter_struct *init_struct);
<b>Function descriptions</b>	initialize DMAMUX request generator channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMAMUX request generator channel is initialized
DMAMUX_GENCHx(x=	DMAMUX generation channel selection, refer to <a href="#">Table 3-95. Enum</a>



0..3)	<a href="#">dmamux_generator_channel_enum</a>
Input parameter{in}	
init_struct	the initialization data needed to initialize DMAMUX request generator channel, refer to <a href="#">Table 3-89. Structure dmamux_gen_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize DMA request generator channel 0 */
dmamux_gen_parameter_struct  dmamux_gen_init_struct;
dmamux_gen_struct_para_init(&dmamux_gen_init_struct);
dmamux_gen_init_struct.trigger_id      = DMAMUX_TRIGGER_EXTI13;
dmamux_gen_init_struct.trigger_polarity = DMAMUX_GEN_RISING;
dmamux_gen_init_struct.request_number = 1;
dmamux_request_generator_init(DMAMUX_GENCH0, &dmamux_gen_init_struct);

```

### dmamux\_request\_generator\_channel\_enable

The description of dmamux\_request\_generator\_channel\_enable is shown as below:

**Table 3-131. Function dmamux\_request\_generator\_channel\_enable**

Function name	dmamux_request_generator_channel_enable
Function prototype	void dmamux_request_generator_channel_enable(dmamux_generator_channel_enum channelx);
Function descriptions	enable DMAMUX request generator channel
Precondition	-
The called functions	-
Input parameter{in}	
channelx	specify which DMAMUX request generator channel is initialized
DMAMUX_GENCHx(x=0..3)	DMAMUX generation channel selection, refer to <a href="#">Table 3-95. Enum dmamux_generator_channel_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable DMAMUX request generator channel */
dmamux_request_generator_channel_enable(DMAMUX_GENCH0);

```

## dmamux\_request\_generator\_channel\_disable

The description of dmamux\_request\_generator\_channel\_disable is shown as below:

**Table 3-132. Function dmamux\_request\_generator\_channel\_disable**

<b>Function name</b>	dmamux_request_generator_channel_disable
<b>Function prototype</b>	void dmamux_request_generator_channel_disable(dmamux_generator_channel_enum channelx);
<b>Function descriptions</b>	disable DMAMUX request generator channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMAMUX request generator channel is initialized
DMAMUX_GENCHx(x=0..3)	DMAMUX generation channel selection, refer to <a href="#">Table 3-95. Enum dmamux_generator_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMAMUX request generator channel */
dmamux_request_generator_channel_disable(DMAMUX_GENCH0);
```

## dmamux\_synchronization\_polarity\_config

The description of dmamux\_synchronization\_polarity\_config is shown as below:

**Table 3-133. Function dmamux\_synchronization\_polarity\_config**

<b>Function name</b>	dmamux_synchronization_polarity_config
<b>Function prototype</b>	void dmamux_synchronization_polarity_config(dmamux_multiplexer_channel_enum channelx, uint32_t polarity);
<b>Function descriptions</b>	configure synchronization input polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMAMUX request multiplexer channel is initialized
DMAMUX_MUXCHx(x=0..2)	DMAMUX channel selection, refer to <a href="#">Table 3-94. Enum dmamux_multiplexer_channel_enum</a>
<b>Input parameter{in}</b>	
<b>polarity</b>	synchronization input polarity
DMAMUX_SYNC_NO_	no event detection

<i>EVENT</i>	
<i>DMAMUX_SYNC_RISING</i>	rising edge
<i>DMAMUX_SYNC_FALLING</i>	falling edge
<i>DMAMUX_SYNC_RISING_FALLING</i>	rising and falling edges
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure synchronization input polarity */
dmamux_synchronization_polarity_config(DMAMUX_MUXCH0, DMAMUX_SYNC_RISING);
```

### dmamux\_request\_forward\_number\_config

The description of dmamux\_request\_forward\_number\_config is shown as below:

**Table 3-134. Function dmamux\_request\_forward\_number\_config**

<b>Function name</b>	dmamux_request_forward_number_config
<b>Function prototype</b>	void dmamux_request_forward_number_config(dmamux_multiplexer_channel_enum channelx, uint32_t number);
<b>Function descriptions</b>	configure number of DMA requests to forward
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMAMUX request multiplexer channel is initialized
<i>DMAMUX_MUXCHx</i> (x=0..2)	DMAMUX channel selection, refer to <a href="#">Table 3-94. Enum dmamux_multiplexer_channel_enum</a>
<b>Input parameter{in}</b>	
<b>number</b>	DMA requests number to forward (1 - 32)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure number of DMA requests to forward */
dmamux_request_forward_number_config(DMAMUX_MUXCH0, 4);
```

## dmamux\_sync\_id\_config

The description of dmamux\_sync\_id\_config is shown as below:

**Table 3-135. Function dmamux\_sync\_id\_config**

<b>Function name</b>	dmamux_sync_id_config
<b>Function prototype</b>	void dmamux_sync_id_config(dmamux_multiplexer_channel_enum channelx, uint32_t id);
<b>Function descriptions</b>	configure synchronization input identification
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMAMUX request multiplexer channel is initialized
DMAMUX_MUXCHx(x=0..2)	DMAMUX channel selection, refer to <a href="#">Table 3-94. Enum dmamux_multiplexer_channel_enum</a>
<b>Input parameter{in}</b>	
<b>id</b>	synchronization input identification
DMAMUX_SYNC_EXTI0	synchronization input is EXTI0
DMAMUX_SYNC_EXTI1	synchronization input is EXTI1
DMAMUX_SYNC_EXTI2	synchronization input is EXTI2
DMAMUX_SYNC_EXTI3	synchronization input is EXTI3
DMAMUX_SYNC_EXTI4	synchronization input is EXTI4
DMAMUX_SYNC_EXTI5	synchronization input is EXTI5
DMAMUX_SYNC_EXTI6	synchronization input is EXTI6
DMAMUX_SYNC_EXTI7	synchronization input is EXTI7
DMAMUX_SYNC_EXTI8	synchronization input is EXTI8
DMAMUX_SYNC_EXTI9	synchronization input is EXTI9
DMAMUX_SYNC_EXTI10	synchronization input is EXTI10
DMAMUX_SYNC_EXTI11	synchronization input is EXTI11
DMAMUX_SYNC_EXTI12	synchronization input is EXTI12

<i>DMAMUX_SYNC_EXTI13</i>	synchronization input is EXTI13
<i>DMAMUX_SYNC_EXTI14</i>	synchronization input is EXTI14
<i>DMAMUX_SYNC_EXTI15</i>	synchronization input is EXTI15
<i>DMAMUX_SYNC_EVT0_OUT</i>	synchronization input is Evt0_out
<i>DMAMUX_SYNC_EVT1_OUT</i>	synchronization input is Evt1_out
<i>DMAMUX_SYNC_EVT2_OUT</i>	synchronization input is Evt2_out
<i>DMAMUX_SYNC_TIMER13_OER13_O</i>	synchronization input is DMAMUX_SYNC_TIMER13_O
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure synchronization input identification */
dmamux_sync_id_config(DMAMUX_MUXCH0, DMAMUX_SYNC_EXTI0);
```

### dmamux\_request\_id\_config

The description of dmamux\_request\_id\_config is shown as below:

**Table 3-136. Function dmamux\_request\_id\_config**

<b>Function name</b>	dmamux_request_id_config
<b>Function prototype</b>	void dmamux_request_id_config(dmamux_mux_channel_enum channelx, uint32_t id);
<b>Function descriptions</b>	configure multiplexer input identification
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMAMUX request multiplexer channel is initialized
<i>DMAMUX_MUXCHx(x=0..2)</i>	DMAMUX channel selection, refer to <a href="#">Table 3-94. Enum dmamux_mux_channel_enum</a>
<b>Input parameter{in}</b>	
<b>id</b>	input DMA request identification
<i>DMA_REQUEST_M2M</i>	memory to memory transfer
<i>DMA_REQUEST_GENERATOR0</i>	DMAMUX request generator 0
<i>DMA_REQUEST_GENERATOR1</i>	DMAMUX request generator 1

<i>ERATOR1</i>	
<i>DMA_REQUEST_GEN ERATOR2</i>	DMAMUX request generator 2
<i>DMA_REQUEST_GEN ERATOR3</i>	DMAMUX request generator 3
<i>DMA_REQUEST_ADC</i>	DMAMUX ADC request
<i>DMA_REQUEST_DAC</i>	DMAMUX DAC request
<i>DMA_REQUEST_I2C0 _RX</i>	DMAMUX I2C0 RX request
<i>DMA_REQUEST_I2C0 _TX</i>	DMAMUX I2C0 TX request
<i>DMA_REQUEST_I2C1 _RX</i>	DMAMUX I2C1 RX request
<i>DMA_REQUEST_I2C1 _TX</i>	DMAMUX I2C1 TX request
<i>DMA_REQUEST_SPI0 _RX</i>	DMAMUX SPI0 RX request
<i>DMA_REQUEST_SPI0 _TX</i>	DMAMUX SPI0 TX request
<i>DMA_REQUEST_SPI1 _RX</i>	DMAMUX SPI1 RX request
<i>DMA_REQUEST_SPI1 _TX</i>	DMAMUX SPI1 TX request
<i>DMA_REQUEST_TIME R0_CH</i>	DMAMUX TIMER0 CH0 request
<i>DMA_REQUEST_TIME R0_CH1</i>	DMAMUX TIMER0 CH1 request
<i>DMA_REQUEST_TIME R0_CH2</i>	DMAMUX TIMER0 CH2 request
<i>DMA_REQUEST_TIME R0_CH3</i>	DMAMUX TIMER0 CH3 request
<i>DMA_REQUEST_TIME R0_TRIG</i>	DMAMUX TIMER0 TRIG request
<i>DMA_REQUEST_TIME R0_UP</i>	DMAMUX TIMER0 UP request
<i>DMA_REQUEST_TIME R0_COM</i>	DMAMUX TIMER0 COM request
<i>DMA_REQUEST_TIME R0_CH0</i>	DMAMUX TIMER2 CH0 request
<i>DMA_REQUEST_TIME R2_CH1</i>	DMAMUX TIMER2 CH1 request
<i>DMA_REQUEST_TIME R2_CH2</i>	DMAMUX TIMER2 CH2 request

<i>DMA_REQUEST_TIME</i> <i>R2_CH3</i>	DMAMUX TIMER2 CH3 request
<i>DMA_REQUEST_TIME</i> <i>R2_TRIG</i>	DMAMUX TIMER2 TRIG request
<i>DMA_REQUEST_TIME</i> <i>R2_UP</i>	DMAMUX TIMER2 UP request
<i>DMA_REQUEST_TIME</i> <i>R15_CH0</i>	DMAMUX TIMER15 CH0 request
<i>DMA_REQUEST_TIME</i> <i>R15_UP</i>	DMAMUX TIMER15 UP request
<i>DMA_REQUEST_TIME</i> <i>R16_CH0</i>	DMAMUX TIMER16 CH0 request
<i>DMA_REQUEST_TIME</i> <i>R16_UP</i>	DMAMUX TIMER16 UP request
<i>DMA_REQUEST_USA</i> <i>RT0_RX</i>	DMAMUX USART0 RX request
<i>DMA_REQUEST_USA</i> <i>RT0_TX</i>	DMAMUX USART0 TX request
<i>DMA_REQUEST_USA</i> <i>RT1_RX</i>	DMAMUX USART1 RX request
<i>DMA_REQUEST_USA</i> <i>RT1_TX</i>	DMAMUX USART1 TX request
<i>DMA_REQUEST_UAR</i> <i>T2_RX</i>	DMAMUX UART2 RX request
<i>DMA_REQUEST_UAR</i> <i>T2_TX</i>	DMAMUX UART2 TX request
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure multiplexer input identification */
dmamux_request_id_config(DMAMUX_MUXCH0, DMA_REQUEST_GENERATOR0);
```

### dmamux\_trigger\_polarity\_config

The description of dmamux\_trigger\_polarity\_config is shown as below:

**Table 3-137. Function dmamux\_trigger\_polarity\_config**

<b>Function name</b>	dmamux_trigger_polarity_config
<b>Function prototype</b>	void dmamux_trigger_polarity_config(dmamux_generator_channel_enum channelx, uint32_t polarity);
<b>Function descriptions</b>	configure trigger input polarity

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMAMUX request generator channel is initialized
DMAMUX_GENCHx(x=0..3)	DMAMUX generation channel selection, refer to <a href="#">Table 3-95. Enum dmamux_generator_channel_enum</a>
<b>Input parameter{in}</b>	
<b>polarity</b>	trigger input polarity
DMAMUX_GEN_NO_EVENT	no event detection
DMAMUX_GEN_RISING	rising edge
DMAMUX_GEN_FALLING	falling edge
DMAMUX_GEN_RISING_FALLING	rising and falling edges
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure trigger input polarity */
dmamux_trigger_polarity_config(DMAMUX_GENCH0, DMAMUX_GEN_RISING);
```

### dmamux\_request\_generate\_number\_config

The description of dmamux\_request\_generate\_number\_config is shown as below:

**Table 3-138. Function dmamux\_request\_generate\_number\_config**

<b>Function name</b>	dmamux_request_generate_number_config
<b>Function prototype</b>	void dmamux_request_generate_number_config(dmamux_generator_channel_enum channelx, uint32_t number);
<b>Function descriptions</b>	configure number of DMA requests to be generated
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMAMUX request generator channel is initialized
DMAMUX_GENCHx(x=0..3)	DMAMUX generation channel selection, refer to <a href="#">Table 3-95. Enum dmamux_generator_channel_enum</a>
<b>Input parameter{in}</b>	
<b>number</b>	DMA requests number to be generated (1 - 32)
<b>Output parameter{out}</b>	



-	-
Return value	
-	-

Example:

```
/* configure number of DMA requests to be generated */
dmamux_request_generate_number_config(DMAMUX_GENCH0, 1);
```

### dmamux\_trigger\_id\_config

The description of dmamux\_trigger\_id\_config is shown as below:

**Table 3-139. Function dmamux\_trigger\_id\_config**

<b>Function name</b>	dmamux_trigger_id_config
<b>Function prototype</b>	void dmamux_trigger_id_config(dmamux_generator_channel_enum channelx, uint32_t id);
<b>Function descriptions</b>	configure trigger input identification
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMAMUX request generator channel is initialized
DMAMUX_GENCHx(x=0..3)	DMAMUX generation channel selection, refer to <a href="#">Table 3-95. Enum dmamux_generator_channel_enum</a>
<b>Input parameter{in}</b>	
<b>id</b>	trigger input identification
DMAMUX_SYNC_EXTI0	synchronization input is EXTI0
DMAMUX_SYNC_EXTI1	synchronization input is EXTI1
DMAMUX_SYNC_EXTI2	synchronization input is EXTI2
DMAMUX_SYNC_EXTI3	synchronization input is EXTI3
DMAMUX_SYNC_EXTI4	synchronization input is EXTI4
DMAMUX_SYNC_EXTI5	synchronization input is EXTI5
DMAMUX_SYNC_EXTI6	synchronization input is EXTI6
DMAMUX_SYNC_EXTI7	synchronization input is EXTI7
DMAMUX_SYNC_EXTI8	synchronization input is EXTI8
DMAMUX_SYNC_EXTI9	synchronization input is EXTI9

9	
DMAMUX_SYNC_EXTI10	synchronization input is EXTI10
DMAMUX_SYNC_EXTI11	synchronization input is EXTI11
DMAMUX_SYNC_EXTI12	synchronization input is EXTI12
DMAMUX_SYNC_EXTI13	synchronization input is EXTI13
DMAMUX_SYNC_EXTI14	synchronization input is EXTI14
DMAMUX_SYNC_EXTI15	synchronization input is EXTI15
DMAMUX_SYNC_EVT0_OUT	synchronization input is Evt0_out
DMAMUX_SYNC_EVT1_OUT	synchronization input is Evt1_out
DMAMUX_SYNC_EVT2_OUT	synchronization input is Evt2_out
DMAMUX_SYNC_TIMER13_O	synchronization input is DMAMUX_SYNC_TIMER13_O
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure trigger input identification */
dmamux_trigger_id_config(DMAMUX_GENCH0, DMAMUX_TRIGGER_EXTI13);
```

### dmamux\_flag\_get

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

**Table 3-140. Function dmamux\_flag\_get**

<b>Function name</b>	dmamux_flag_get
<b>Function prototype</b>	FlagStatus dmamux_flag_get(dmamux_flag_enum flag);
<b>Function descriptions</b>	get DMAMUX flag
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>flag</b>	flag type, refer to <a href="#">Table 3-91. Enum dmamux_flag_enum</a>
DMAMUX_FLAG_MUXCH0_SO	DMAMUX request multiplexer channel 0 synchronization overrun flag

<i>DMAMUX_FLAG_MUX</i> <i>CH1_SO</i>	DMAMUX request multiplexer channel 1 synchronization overrun flag
<i>DMAMUX_FLAG_MUX</i> <i>CH2_SO</i>	DMAMUX request multiplexer channel 2 synchronization overrun flag
<i>DMAMUX_FLAG_GEN</i> <i>CH0_TO</i>	DMAMUX request generator channel 0 trigger overrun flag
<i>DMAMUX_FLAG_GEN</i> <i>CH1_TO</i>	DMAMUX request generator channel 1 trigger overrun flag
<i>DMAMUX_FLAG_GEN</i> <i>CH2_TO</i>	DMAMUX request generator channel 2 trigger overrun flag
<i>DMAMUX_FLAG_GEN</i> <i>CH3_TO</i>	DMAMUX request generator channel 3 trigger overrun flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
FlagStatus flag = RESET;
/* get DMAMUX flag */
flag = dmamux_flag_get(DMAMUX_FLAG_GENCH0_TO);
```

### dmamux\_flag\_clear

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

**Table 3-141. Function dmamux\_flag\_clear**

<b>Function name</b>	dmamux_flag_clear
<b>Function prototype</b>	void dmamux_flag_clear(dmamux_flag_enum flag);
<b>Function descriptions</b>	clear DMAMUX flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flag type, refer to <a href="#">Table 3-91. Enum dmamux_flag_enum</a>
<i>DMAMUX_FLAG_MUX</i> <i>CH0_SO</i>	DMAMUX request multiplexer channel 0 synchronization overrun flag
<i>DMAMUX_FLAG_MUX</i> <i>CH1_SO</i>	DMAMUX request multiplexer channel 1 synchronization overrun flag
<i>DMAMUX_FLAG_MUX</i> <i>CH2_SO</i>	DMAMUX request multiplexer channel 2 synchronization overrun flag
<i>DMAMUX_FLAG_GEN</i> <i>CH0_TO</i>	DMAMUX request generator channel 0 trigger overrun flag
<i>DMAMUX_FLAG_GEN</i> <i>CH1_TO</i>	DMAMUX request generator channel 1 trigger overrun flag

<i>DMAMUX_FLAG_GEN</i> <i>CH2_TO</i>	DMAMUX request generator channel 2 trigger overrun flag
<i>DMAMUX_FLAG_GEN</i> <i>CH3_TO</i>	DMAMUX request generator channel 3 trigger overrun flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear DMAMUX flag */
dmamux_flag_clear(DMAMUX_FLAG_GENCH0_TO);
```

### dmamux\_interrupt\_enable

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

**Table 3-142. Function dmamux\_interrupt\_enable**

<b>Function name</b>	dmamux_interrupt_enable
<b>Function prototype</b>	void dmamux_interrupt_enable(dmamux_interrupt_enum interrupt);
<b>Function descriptions</b>	enable DMAMUX interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which interrupt to enable
<i>DMAMUX_INT_MUXC</i> <i>H0_SO</i>	DMAMUX request multiplexer channel 0 synchronization overrun interrupt
<i>DMAMUX_INT_MUXC</i> <i>H1_SO</i>	DMAMUX request multiplexer channel 1 synchronization overrun interrupt
<i>DMAMUX_INT_MUXC</i> <i>H2_SO</i>	DMAMUX request multiplexer channel 2 synchronization overrun interrupt
<i>DMAMUX_INT_GENC</i> <i>H0_TO</i>	DMAMUX request generator channel 0 trigger overrun interrupt
<i>DMAMUX_INT_GENC</i> <i>H1_TO</i>	DMAMUX request generator channel 1 trigger overrun interrupt
<i>DMAMUX_INT_GENC</i> <i>H2_TO</i>	DMAMUX request generator channel 2 trigger overrun interrupt
<i>DMAMUX_INT_GENC</i> <i>H3_TO</i>	DMAMUX request generator channel 3 trigger overrun interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMAMUX interrupt */
```

```
dmamux_interrupt_enable(DMAMUX_INT_MUXCH0_SO);
```

### dmamux\_interrupt\_disable

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

**Table 3-143. Function dmamux\_interrupt\_disable**

Function name	dmamux_interrupt_disable
Function prototype	void dmamux_interrupt_disable(dmamux_interrupt_enum interrupt);
Function descriptions	disable DMAMUX interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	specify which interrupt to disable
DMAMUX_INT_MUXC H0_SO	DMAMUX request multiplexer channel 0 synchronization overrun interrupt
DMAMUX_INT_MUXC H1_SO	DMAMUX request multiplexer channel 1 synchronization overrun interrupt
DMAMUX_INT_MUXC H2_SO	DMAMUX request multiplexer channel 2 synchronization overrun interrupt
DMAMUX_INT_GENC H0_TO	DMAMUX request generator channel 0 trigger overrun interrupt
DMAMUX_INT_GENC H1_TO	DMAMUX request generator channel 1 trigger overrun interrupt
DMAMUX_INT_GENC H2_TO	DMAMUX request generator channel 2 trigger overrun interrupt
DMAMUX_INT_GENC H3_TO	DMAMUX request generator channel 3 trigger overrun interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMAMUX interrupt */
```

```
dmamux_interrupt_disable(DMAMUX_INT_MUXCH0_SO);
```

### dmamux\_interrupt\_flag\_get

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

**Table 3-144. Function dmamux\_interrupt\_flag\_get**

Function name	dmamux_interrupt_flag_get
Function prototype	FlagStatus dmamux_interrupt_flag_get(dmamux_interrupt_flag_enum

	int_flag);
<b>Function descriptions</b>	get DMAMUX interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	flag type, refer to <a href="#">Table 3-92. Enum dmamux_interrupt_flag_enum</a>
DMAMUX_INT_FLAG_MUXCH0_SO	DMAMUX request multiplexer channel 0 synchronization overrun interrupt flag
DMAMUX_INT_FLAG_MUXCH1_SO	DMAMUX request multiplexer channel 1 synchronization overrun interrupt flag
DMAMUX_INT_FLAG_MUXCH2_SO	DMAMUX request multiplexer channel 2 synchronization overrun interrupt flag
DMAMUX_INT_FLAG_GENCH0_TO	DMAMUX request generator channel 0 trigger overrun interrupt flag
DMAMUX_INT_FLAG_GENCH1_TO	DMAMUX request generator channel 1 trigger overrun interrupt flag
DMAMUX_INT_FLAG_GENCH2_TO	DMAMUX request generator channel 2 trigger overrun interrupt flag
DMAMUX_INT_FLAG_GENCH3_TO	DMAMUX request generator channel 3 trigger overrun interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```

/* check DMAMUX interrupt flag */
if(dmamux_interrupt_flag_get(DMAMUX_INT_FLAG_GENCH0_TO)) {
    dmamux_interrupt_flag_clear(DMAMUX_INT_FLAG_GENCH0_TO);
}

```

### dmamux\_interrupt\_flag\_clear

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

**Table 3-145. Function dmamux\_interrupt\_flag\_clear**

<b>Function name</b>	dmamux_interrupt_flag_clear
<b>Function prototype</b>	FlagStatus dmamux_interrupt_flag_get(dmamux_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	clear DMAMUX interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	flag type, refer to <a href="#">Table 3-92. Enum dmamux_interrupt_flag_enum</a>

<i>DMAMUX_INT_FLAG_MUXCH0_SO</i>	DMAMUX request multiplexer channel 0 synchronization overrun interrupt flag
<i>DMAMUX_INT_FLAG_MUXCH1_SO</i>	DMAMUX request multiplexer channel 1 synchronization overrun interrupt flag
<i>DMAMUX_INT_FLAG_MUXCH2_SO</i>	DMAMUX request multiplexer channel 2 synchronization overrun interrupt flag
<i>DMAMUX_INT_FLAG_GENCH0_TO</i>	DMAMUX request generator channel 0 trigger overrun interrupt flag
<i>DMAMUX_INT_FLAG_GENCH1_TO</i>	DMAMUX request generator channel 1 trigger overrun interrupt flag
<i>DMAMUX_INT_FLAG_GENCH2_TO</i>	DMAMUX request generator channel 2 trigger overrun interrupt flag
<i>DMAMUX_INT_FLAG_GENCH3_TO</i>	DMAMUX request generator channel 3 trigger overrun interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* check DMAMUX interrupt flag */
if(dmamux_interrupt_flag_get(DMAMUX_INT_FLAG_GENCH0_TO)) {
    dmamux_interrupt_flag_clear(DMAMUX_INT_FLAG_GENCH0_TO);
}

```

## 3.7. EXTI

EXTI is the interrupt/event controller in the MCU. It contains up to 24 independent edge detectors and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter [3.7.1](#), the EXTI firmware functions are introduced in chapter [3.7.2](#).

### 3.7.1. Descriptions of Peripheral registers

EXTI registers are listed in the table shown as below:

**Table 3-146. 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

### 3.7.2. Descriptions of Peripheral functions

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

**Table 3-147. EXTI firmware function**

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

#### Enum exti\_line\_enum

**Table 3-148. exti\_line\_enum**

enum 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



enum name	Function description
EXTI_20	EXTI line 20
EXTI_21	EXTI line 21
EXTI_22	EXTI line 22
EXTI_23	EXTI line 23

### Enum exti\_mode\_enum

Table 3-149. exti\_mode\_enum

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

### Enum exti\_trig\_type\_enum

Table 3-150. exti\_trig\_type\_enum

enum 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 edge or falling edge trigger

### exti\_deinit

The description of exti\_deinit is shown as below:

Table 3-151. 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 3-152. 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 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 3-148. exti_line_enum</a>
<b>Input parameter{in}</b>	
<b>mode</b>	EXTI mode, refer to <a href="#">Table 3-149. exti_mode_enum</a>
<b>Input parameter{in}</b>	
<b>trig_type</b>	trigger type, refer to <a href="#">Table 3-150. 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 3-153. 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 3-148. exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
exti_interrupt_enable(EXTI_0);
```

### exti\_interrupt\_disable

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

**Table 3-154. Function exti\_interrupt\_disable**

<b>Function name</b>	exti_interrupt_disable
<b>Function prototype</b>	void exti_interrupt_disable(exti_line_enum linex);
<b>Function descriptions</b>	disable the interrupts from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-148. exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
exti_interrupt_disable(EXTI_0);
```

### exti\_event\_enable

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

**Table 3-155. Function exti\_event\_enable**

<b>Function name</b>	exti_event_enable
<b>Function prototype</b>	void exti_event_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable the events from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-148. exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
exti_event_enable(EXTI_0);
```

### exti\_event\_disable

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

**Table 3-156. 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 3-148. 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 3-157. 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 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 3-148. 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 3-158. 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 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 3-148. 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 3-159. 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 3-148. 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 3-160. 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 3-148. 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 3-161. 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 3-148. 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 3-162. Function exti\_interrupt\_flag\_clear**

<b>Function name</b>	exti_interrupt_flag_clear
<b>Function prototype</b>	void exti_interrupt_flag_clear(exti_line_enum linex);
<b>Function descriptions</b>	clear EXTI line x interrupt pending flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-148. exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## 3.8. FMC

There is flash controller and option byte. The FMC registers are listed in chapter [3.8.1](#) the FMC firmware functions are introduced in chapter [3.8.2](#).

### 3.8.1. Descriptions of Peripheral registers

FMC registers are listed in the table shown as below:

**Table 3-163. 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_OBCTL	FMC FMC option byte control register
FMC_DCRP_SADDR0	FMC DCRP area start address register 0
FMC_DCRP_EADDR0	FMC DCRP area end address register 0
FMC_WP0	FMC erase/program protection area 0 register
FMC_WP1	FMC erase/program protection area 1 register
FMC_DCRP_SADDR1	FMC DCRP area start address register 1

Registers	Descriptions
FMC_DCRP_EADDR1	FMC DCRP area end address register 1
FMC_SCR	FMC securable area register

### 3.8.2. Descriptions of Peripheral functions

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

**Table 3-164. FMC firmware function**

Function name	Function description
fmc_unlock	unlock the main FMC operation
fmc_lock	lock the main FMC operation
fmc_main_flash_empty_check_stat_get	get the main flash empty check status
fmc_main_flash_empty_check_stat_modify	modify the main flash empty check status
fmc_wscnt_set	set the wait state
fmc_prefetch_enable	enable pre-fetch
fmc_prefetch_disable	disable pre-fetch
fmc_icache_enable	enable IBUS cache
fmc_icache_disable	disable IBUS cache
fmc_icache_reset	reset IBUS cache
fmc_page_erase	erase page
fmc_mass_erase	erase whole chip
fmc_doubleword_program	program a doubleword at the given address
fmc_fast_program	fast program
fmc_debugger_enable	enable debugger
fmc_debugger_disable	disable debugger
fmc_scr_area_enable	enable secure area protection
ob_unlock	unlock the option bytes operation
ob_lock	lock the option bytes operation
ob_reload	reload the option bytes operation
ob_user_write	program option bytes USER
ob_security_protection_level_config	configure the option byte security protection level
ob_dcrp_area_config	configure the option bytes DCRP area
ob_write_protection_area_config	configure the option bytes write protection area
ob_scr_area_config	configure the option bytes secure area
ob_boot_lock_config	configure the option byte boot lock
ob_user_get	get the value of option bytes USER
ob_security_protection_level_get	get the value of option bytes security protection level in FMC_OBCTL register
ob_dcrp_area_get	get DCRP area configuration
ob_write_protection_area_get	get the value of option bytes write protection
ob_scr_area_get	get size of secure area
ob_boot_lock_get	get boot configuration



Function name	Function description
fmc_flag_get	get FMC flag status
fmc_flag_clear	clear the FMC flag
fmc_interrupt_enable	enable FMC interrupt
fmc_interrupt_disable	disable FMC interrupt
fmc_interrupt_flag_get	get FMC interrupt flag
fmc_interrupt_flag_clear	clear FMC interrupt flag
fmc_state_get	return FMC state
fmc_ready_wait	check FMC ready or not

### fmc\_state\_enum

Table 3-165. fmc\_state\_enum

enum name	enum description
FMC_READY	the operation has been completed
FMC_BUSY	the operation is in progress
FMC_OBERR	option byte read error
FMC_RPERR	read protection error
FMC_FSTPRR	fast program error
FMC_PGSERR	program sequence error
FMC_PGMERR	program size error
FMC_PGAERR	program alignment error
FMC_WPERR	erase/program protection error
FMC_PGERR	program error
FMC_OPRERR	operation error
FMC_TOERR	timeout error
FMC_OB_HSPC	high security protection
FMC_UNDEFINEDERR	undefined error for function input parameter checking

### ob\_user\_data\_extract\_info\_enum

Table 3-166. ob\_user\_data\_extract\_info\_enum

enum name	enum description
OBCTL_USER_DATA_BORST_EN	BORST_EN mask and bit position
OBCTL_USER_DATA_BORR_TH	BORR_TH mask and bit position
OBCTL_USER_DATA_BORF_TH	BORF_TH mask and bit position
OBCTL_USER_DATA_NRST_DPSLP	nRST_DPSLP mask and bit position
OBCTL_USER_DATA_NRST_STDBY	nRST_STDBY mask and bit position
OBCTL_USER_DATA_NFWDG_HW	nFWDG_HW mask and bit position
OBCTL_USER_DATA_NWWDG_HW	nWWDG mask and bit position
OBCTL_USER_DATA_HXTAL_REMAP	HXTAL_REMAP mask and bit position
OBCTL_USER_DATA_SRAM_ECC_EN	SRAM_ECC_EN mask and bit position
OBCTL_USER_DATA_SWBT0	SWBT0 mask and bit position

enum name	enum description
OBCTL_USER_DATA_NBOOT1	nBOOT1 mask and bit position
OBCTL_USER_DATA_NBOOT0	nBOOT0 mask and bit position
OBCTL_USER_DATA_NRST_MDSEL	NRST_MDSEL mask and bit position

## fmc\_unlock

The description of fmc\_unlock is shown as below:

**Table 3-167. Function fmc\_unlock**

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

Example:

```
/* unlock FMC_CTL register */
fmc_unlock();
```

## fmc\_lock

The description of fmc\_lock is shown as below:

**Table 3-168. Function fmc\_lock**

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

Example:

```
/* lock FMC_CTL register */
```

```
fmc_lock();
```

### fmc\_main\_flash\_empty\_check\_stat\_get

The description of fmc\_main\_flash\_empty\_check\_stat\_get is shown as below:

表 3-1. Function fmc\_wsctl\_set

<b>Function name</b>	fmc_main_flash_empty_check_stat_get
<b>Function prototype</b>	FlagStatus fmc_main_flash_empty_check_stat_get(void);
<b>Function descriptions</b>	get the main flash empty check status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	main flash empty check status, SET: empty; RESET: not empty

Example:

```
/* get the main flash empty check status */
```

```
FlagStatus status = fmc_main_flash_empty_check_stat_get();
```

### fmc\_main\_flash\_empty\_check\_stat\_modify

The description of fmc\_main\_flash\_empty\_check\_stat\_modify is shown as below:

表 3-2. Function fmc\_main\_flash\_empty\_check\_stat\_modify

<b>Function name</b>	fmc_main_flash_empty_check_stat_modify
<b>Function prototype</b>	void fmc_main_flash_empty_check_stat_modify(uint32_t empty_check_status)
<b>Function descriptions</b>	modify the main flash empty check status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>empty_check_status</b>	main flash empty check status to be set
FMC_WS_MFPE_PROGRAMMED	set the main flash empty check status to be not empty
FMC_WS_MFPE_EMPTY	set the main flash empty check status to be empty
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* set the main flash empty check status to be FMC_WS_MFPE_EMPTY */
```

```
fmc_main_flash_empty_check_stat_modify(FMC_WS_MFPE_EMPTY);
```

### fmc\_wscnt\_set

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

**Table 3-169. 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
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wscnt</b>	wait state
<i>FMC_WAIT_STATE_0</i>	0 wait state added
<i>FMC_WAIT_STATE_1</i>	1 wait state added
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the wait state 0 */
```

```
fmc_wscnt_set(FMC_WAIT_STATE_0);
```

### fmc\_prefetch\_enable

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

**Table 3-170. Function fmc\_prefetch\_enable**

<b>Function name</b>	fmc_prefetch_enable
<b>Function prototype</b>	void fmc_prefetch_enable(void);
<b>Function descriptions</b>	enable pre-fetch
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* enable pre-fetch */
```

```
fmc_prefetch_enable( );
```

### fmc\_prefetch\_disable

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

**Table 3-171. Function fmc\_prefetch\_disable**

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

Example:

```
/* disable pre-fetch */
```

```
fmc_prefetch_disable( );
```

### fmc\_icache\_enable

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

**Table 3-172. Function fmc\_icache\_enable**

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

-	-
---	---

Example:

```
/* enable icache */
fmc_icache_enable( );
```

### fmc\_icache\_disable

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

**Table 3-173. Function fmc\_icache\_disable**

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

Example:

```
/* disable icache */
fmc_icache_disable( );
```

### fmc\_icache\_reset

The description of fmc\_icache\_reset\_enable is shown as below::

**Table 3-174. Function fmc\_icache\_reset\_enable**

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

Example:

```
/* icache reset */

fmc_icache_reset( );
```

### fmc\_page\_erase

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

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

<b>Function name</b>	fmc_page_erase
<b>Function prototype</b>	fmc_page_erase(uint32_t bank, uint32_t page_number_in_bank)
<b>Function descriptions</b>	FMC erase page
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>bank</b>	specify which bank the target page is in
<i>FMC_BANK0</i>	BANK0 of main flash
<i>FMC_BANK1</i>	BANK1 of main flash
<b>Input parameter{in}</b>	
<b>page_number_in_bank</b>	Page number in the bank
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	FMC status, refers to <a href="#">Table 3-165. fmc_state_enum</a>

Example:

```
fmc_unlock();

/* erase page */

fmc_state_enum state = fmc_page_erase(0);
```

### fmc\_mass\_erase

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

**Table 3-176. Function fmc\_mass\_erase**

<b>Function name</b>	fmc_mass_erase
<b>Function prototype</b>	fmc_state_enum fmc_mass_erase(void);
<b>Function descriptions</b>	FMC erase whole chip
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-

Output parameter{out}	
-	-
Return value	
fmc_state_enum	FMC status, refers to <a href="#">Table 3-165. fmc_state_enum</a>

Example:

```
fmc_unlock();

/* erase whole chip */

fmc_state_enum state = fmc_mass_erase( );
```

### fmc\_doubleword\_program

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

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

Function name	fmc_doubleword_program
Function prototype	fmc_state_enum fmc_doubleword_program(uint32_t address, uint64_t data);
Function descriptions	program a double word at the given address in main flash
Precondition	fmc_unlock
The called functions	-
Input parameter{in}	
address	address to program
Input parameter{in}	
data	double word to program
Output parameter{out}	
-	-
Return value	
fmc_state_enum	FMC status, refers to <a href="#">Table 3-165. fmc_state_enum</a>

Example:

```
fmc_unlock();

fmc_page_erase(0x08004000);

/* program a double word at the corresponding address in main flash */

fmc_state_enum fmc_state = fmc_doubleword_program(0x08004000, 0x11223344aabbccdd);
```

### fmc\_fast\_program

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



Table 3-178. Function `fmc_fast_program`

Function name	<code>fmc_fast_program</code>
Function prototype	<code>fmc_state_enum fmc_fast_program(uint32_t address, uint32_t data_buf)</code>
Function descriptions	ast program a row at the corresponding address
Precondition	-
The called functions	-
Input parameter{in}	
address	address to program
Input parameter{in}	
Output parameter{out}	
data_buf	data buffer to program
Return value	
fmc_state_enum	FMC status, refers to <a href="#">Table 3-165. fmc_state_enum</a>

Example:

Example:

```
/* data buffer for fast programming */
```

```
static uint64_t data_buffer[32] = {
    0x0000000000000000U,    0x1111111111111111U,    0x2222222222222222U,
    0x3333333333333333U,
    0x4444444444444444U,    0x5555555555555555U,    0x6666666666666666U,
    0x7777777777777777U,
    0x8888888888888888U,    0x9999999999999999U,    0xAAAAAAAAAAAAAAAAAU,
    0BBBBBBBBBBBBBBBBBU,
    0xCCCCCCCCCCCCCCCCCU,    0xDDDDDDDDDDDDDDDDDU,
    0xEEEEEEEEEEEEEEEEU, 0xFFFFFFFFFFFFFFFFFU,
    0x0011001100110011U,    0x2233223322332233U,    0x4455445544554455U,
    0x6677667766776677U,
    0x8899889988998899U, 0xAABBAABBAABBAABBU, 0xCCDDCCDDCCDDCCDDU,
    0xEEFFEEFFEEFFEEFFU,
    0x2200220022002200U,    0x3311331133113311U,    0x6644664466446644U,
    0x7755775577557755U,
    0xAA88AA88AA88AA88U, 0xBB99BB99BB99BB99U, 0xEECCCEECCEECCECCU,
    0xFFDDFFDDFFDDFFDDU
};

fmc_unlock();
```

```
fmc_page_erase(0x08004000);
```

```
/* program flash */
```

```
fmc_state_enum fmc_state = fmc_fast_program(0x08004000, data_buffer);
```

### fmc\_debugger\_enable

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

**Table 3-179. Function fmc\_debugger\_enable**

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

Example:

```
/* enable debugger */
```

```
fmc_debugger_enable( );
```

### fmc\_debugger\_disable

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

**Table 3-180. Function fmc\_debugger\_disable**

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

Example:

```
/* disable debugger */
```

```
fmc_debugger_disable( );
```

### fmc\_scr\_area\_enable

The description of fmc\_scr\_area\_enable is shown as below:

**Table 3-181. Function fmc\_scr\_area\_enable**

<b>Function name</b>	fmc_scr_area_enable
<b>Function prototype</b>	void fmc_scr_area_enable();
<b>Function descriptions</b>	enable secure area protection
<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:

```
fmc_unlock();
```

```
/* enable secure area protection */
```

```
fmc_scr_area_enable();
```

### ob\_unlock

The description of ob\_unlock is shown as below:

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

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

Example:

```
fmc_unlock();
```

```
/* unlock the option bytes operation */
```

```
ob_unlock();
```

## ob\_lock

The description of ob\_lock is shown as below:

**Table 3-183. Function ob\_lock**

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

Example:

```
fmc_unlock();
```

```
/*lock the option bytes operation */
```

```
ob_lock();
```

## ob\_reload

The description of ob\_reload is shown as below:

**Table 3-184. Function ob\_reload**

<b>Function name</b>	ob_reload
<b>Function prototype</b>	void ob_reload(void);
<b>Function descriptions</b>	reload the option bytes operation
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* reload the option bytes operation */
```

```
ob_reload( );
```

### ob\_user\_write

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

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

Function name	ob_user_write
Function prototype	fmc_state_enum ob_user_write(uint32_t ob_user, uint32_t ob_user_mask);
Function descriptions	program the option bytes USER
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
<b>ob_user</b>	user option bytes. When write one paramter, the corresponding mask should be set.
OB_NRST_PIN_INPUT_MODE	a low level on the NRST pin can reset system, internal reset can not drive NRST pin
OB_NRST_PIN_NORMAL_GPIO	NRST pin function as normal GPIO
OB_NRST_PIN_INPUT_OUTPUT_MODE	NRST pin configure as input/output mode
OB_NBOOT0_VALUE_0	option byte BOOT0 is value 1
OB_NBOOT0_VALUE_1	option byte BOOT0 is value 0
OB_NBOOT1_VALUE_0	option byte BOOT1 is value 1
OB_NBOOT1_VALUE_1	option byte BOOT1 is value 0
OB_SWBOOT0_FROM_OB_BOOT0	BOOT0 taken from the option bit NBOOT0
OB_SWBOOT0_FROM_PIN	BOOT0 taken from PB8/BOOT0 pin
OB_SRAM_ECC_ENABLE	SRAM ECC enable
OB_SRAM_ECC_DISABLE	SRAM ECC disable
OB_HXTAL_REMAP_ENABLE	HXTAL remapping enable

<i>NABLE</i>	
<i>OB_HXTAL_REMAP_DISABLE</i>	HXTAL remapping disable
<i>OB_WWDGT_HW</i>	hardware window watchdog
<i>OB_WWDGT_SW</i>	software window watchdog
<i>OB_FWDGT_HW</i>	hardware free watchdog
<i>OB_FWDGT_SW</i>	software free watchdog
<i>OB_STDBY_RST</i>	generate a reset instead of entering standby mode
<i>OB_STDBY_NRST</i>	no reset when entering deepsleep mode
<i>OB_DEEPSLEEP_RST</i>	generate a reset instead of entering deepsleep mode
<i>OB_DEEPSLEEP_NRS T</i>	no reset when entering deepsleep mode
<i>OB_BORR_TH_VALUE 0</i>	BOR rising level 1
<i>OB_BORR_TH_VALUE 1</i>	BOR rising level 2
<i>OB_BORR_TH_VALUE 2</i>	BOR rising level 3
<i>OB_BORR_TH_VALUE 3</i>	BOR rising level 4
<i>OB_BORF_TH_VALUE 0</i>	BOR falling level 1
<i>OB_BORF_TH_VALUE 1</i>	BOR falling level 2
<i>OB_BORF_TH_VALUE 2</i>	BOR falling level 3
<i>OB_BORF_TH_VALUE 3</i>	BOR falling level 4
<i>OB_BORST_DISABLE</i>	brown out reset disable
<i>OB_BORST_ENABLE</i>	brown out reset enable
<b>Input parameter{in}</b>	
<b>ob_user_mask</b>	user bits mask
<i>FMC_OBCTL_NRST_M DSEL</i>	reset pin mode bit
<i>FMC_OBCTL_NBOOT0</i>	NBOOT0 option bit
<i>FMC_OBCTL_NBOOT1</i>	NBOOT1 option bit
<i>FMC_OBCTL_SWBT0</i>	software BOOT0 bit
<i>FMC_OBCTL_SRAM_E CC_EN</i>	SRAM ECC disable bit
<i>FMC_OBCTL_HXTAL_ REMAP</i>	HXTAL remapping bit
<i>FMC_OBCTL_NWWDG _HW</i>	window watchdog configuration bit

<i>FMC_OBCTL_NFWDG_HW</i>	free watchdog configuration bit
<i>FMC_OBCTL_NRST_STDBY</i>	option byte standby reset value bit
<i>FMC_OBCTL_NRST_DPSLP</i>	option byte deepsleep reset value bit
<i>FMC_OBCTL_BORF_TH</i>	BOR threshold at rising VDD supply
<i>FMC_OBCTL_BORR_TH</i>	BOR threshold at falling VDD supply
<i>FMC_OBCTL_BORSTEN</i>	brown out reset enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	FMC status, refers to <a href="#">Table 3-165. fmc_state_enum</a>

Example:

```
fmc_state_enum fmc_state;

fmc_unlock();

ob_unlock();

/* modify option byte */

fmc_state = ob_user_write (OB_NRST_PIN_INPUT_MODE, FMC_OBCTL_NRST_MDSEL);
```

### ob\_security\_protection\_level\_config

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

**Table 3-186. Function ob\_security\_protection\_level\_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>	FMC status, refers to <a href="#">Table 3-165. fmc_state_enum</a>
-----------------------	---

Example:

```
fmc_state_enum fmc_state;

fmc_unlock();

ob_unlock();

/* disable security protection */

fmc_state = ob_security_protection_level_config (FMC_NSPC);
```

### ob\_dcrp\_area\_config

The description of ob\_dcrp\_area\_config is shown as below:

**Table 3-187. Function ob\_dcrp\_area\_config**

<b>Function name</b>	ob_dcrp_area_config
<b>Function prototype</b>	fmc_state_enum ob_dcrp_config(uint32_t dcrp_area, uint32_t dcrp_eren, uint32_t dcrp_start, uint32_t dcrp_end);
<b>Function descriptions</b>	configure the option byte DCRP area
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>dcrp_area</b>	DCRP area
<i>DCRP_AREA0</i>	the first DCRP area
<i>DCRP_AREA1</i>	the second DCRP area
<b>Input parameter{in}</b>	
<b>dcrp_eren</b>	DCRP area erase enable bit
<i>OB_DCRP_AREA_ERASE_DISABLE</i>	DCRP area is not erased when low security protection to no security protection
<i>OB_DCRP_AREA_ERASE_ENABLE</i>	DCRP area is erased when low security protection to no security protection
<b>Input parameter{in}</b>	
<b>dcrp_start</b>	first page of DCRP area
<b>Input parameter{in}</b>	
<b>dcrp_end</b>	last page of DCRP area
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	FMC status, refers to <a href="#">Table 3-165. fmc_state_enum</a>

Example:

```
fmc_state_enum fmc_state;
```



```
fmc_unlock();
```

```
ob_unlock();
```

```
/* configure DCRP area */
```

```
fmc_state = fmc_state_enum ob_dcrp_area_config(DCRP_AREA0, OB_DCRP_AREA_E
RASE_ENABLE, 0x05, 0x0B);
```

### ob\_write\_protection\_config

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

**Table 3-188. Function ob\_write\_protection\_config**

<b>Function name</b>	ob_write_protection_config
<b>Function prototype</b>	fmc_state_enum ob_write_protection_config(uint32_t wp_area, uint32_t wp_start, uint32_t wp_end);
<b>Function descriptions</b>	configure the option byte write protection area
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>wp_area</b>	write protection area
WP_AREA0	the first write protection area
WP_AREA1	the second write protection area
<b>Input parameter{in}</b>	
<b>wp_start</b>	first page of write protection area
<b>Input parameter{in}</b>	
<b>wp_end</b>	last page of write protection area
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	FMC status, refers to <a href="#">Table 3-165. fmc_state_enum</a>

Example:

```
fmc_state_enum fmc_state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* configure write protection area */
```

```
fmc_state = fmc_state_enum ob_write_protection_config(WP_AREA0, 0x08, 0x09);
```

### ob\_scr\_area\_config

The description of ob\_scr\_area\_config is shown as below:

Table 3-189. Function ob\_scr\_area\_config

Function name	ob_scr_area_config
Function prototype	fmc_state_enum ob_scr_area_config(uint32_t secure_size);
Function descriptions	configure the option byte secure area
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
secure_size	size of secure area in page unit
Output parameter{out}	
-	-
Return value	
fmc_state_enum	FMC status, refers to <a href="#">Table 3-165. fmc_state_enum</a>

Example:

```
fmc_unlock();

ob_unlock();

/* configure the option byte secure area */

fmc_state_enum fmc_state = ob_scr_area_config(0x02);
```

### ob\_boot\_lock\_config

The description of ob\_boot\_lock\_config is shown as below:

Table 3-190. Function ob\_boot\_lock\_config

Function name	ob_boot_lock_config
Function prototype	fmc_state_enum ob_boot_lock_config(uint32_t boot_config);
Function descriptions	配置boot锁定
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
boot_config	boot configuration
OB_BOOT_LOCK_FR OM_MAIN_FLASH	boot from main flash
OB_BOOT_UNLOCK	unlock boot
Output parameter{out}	
-	-
Return value	
fmc_state_enum	FMC status, refers to <a href="#">Table 3-165. fmc_state_enum</a>

Example:

```
fmc_unlock();

ob_unlock();
```

```
/* unlock boot */
```

```
fmc_state_enum fmc_state = ob_boot_lock_config(OB_BOOT_UNLOCK);
```

## ob\_user\_get

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

**Table 3-191. Function ob\_user\_get**

<b>Function name</b>	ob_user_get
<b>Function prototype</b>	uint32_t ob_user_get(ob_user_data_extract_info_enum user_data_extract_info, uint8_t * ob_user_data);
<b>Function descriptions</b>	get the value of option bytes USER
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
ob_user_data_extract_info_enum	user data extract information which include mask and start bit position, refers to <a href="#">Table 3-166. ob_user_data_extract_info_enum</a>
<b>Output parameter{out}</b>	
ob_user_data	the value of option bytes USER
<b>Return value</b>	
uint32_t	state of the return value
INVLD_RETURN_VALUE	the return value is invalid
VLD_RETURN_VALUE	the return value is valid

Example:

```
uint32_t user_value, ;

ob_user_data_extract_info_enum user_data_extract_info;

uint8_t ob_user_data;

user_value = ob_user_get(OBCTL_USER_DATA_BORST_EN, ob_user_data);
```

## ob\_security\_protection\_level\_get

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

**Table 3-192. Function ob\_security\_protection\_level\_get**

<b>Function name</b>	ob_security_protection_level_get
<b>Function prototype</b>	uint8_t ob_security_protection_level_get(void);
<b>Function descriptions</b>	get the value of FMC option bytes security protection level in FMC_OBCTL register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

-	-
Output parameter{out}	
-	-
Return value	
uint8_t	protection level
FMC_NSPC	no protection
FMC_LSPC	protection level low
FMC_HSPC	protection level high

Example:

```
/* get the value of FMC option bytes security protection level */
```

```
uint8_t user = ob_security_protection_level_get();
```

### ob\_dcrp\_area\_get

The description of ob\_dcrp\_area\_get is shown as below:

**Table 3-193. Function ob\_dcrp\_area\_get**

<b>Function name</b>	ob_dcrp_area_get
<b>Function prototype</b>	uint32_t ob_dcrp_area_get(uint32_t dcrp_area, uint32_t *dcrp_erase_option, uint32_t *dcrp_start, uint32_t *dcrp_end);
<b>Function descriptions</b>	get configuration of DCRP area
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
dcrp_area	DCRP area
DCRP_AREA0	the first DCRP area
DCRP_AREA1	the second DCRP area
Output parameter{out}	
dcrp_erase_option	erase option of DCRP area
Output parameter{out}	
dcrp_start	first page of DCRP area
Output parameter{out}	
dcrp_end	last page of DCRP area
Return value	
uint32_t	state of the return value
INVLD_RETURN_VAL UE	the return value is invalid
VLD_RETURN_VALUE	the return value is valid

Example:

```
/* get configuration of DCRP area */
```

```
uint32_t start, end, erase ,flag;
```

```
flag = ob_dcrp_area_get(DCRP_AREA0, &erase, &start, &end)
```

### ob\_write\_protection\_area\_get

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

**Table 3-194. Function ob\_write\_protection\_area\_get**

<b>Function name</b>	ob_write_protection_area_get
<b>Function prototype</b>	uint32_t ob_write_protection_area_get(uint32_t wp_area, uint32_t *wp_start, uint32_t *wp_end);
<b>Function descriptions</b>	get address of write protection area
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wp_area</b>	write protection area
<i>WP_AREA0</i>	the first write protection area
<i>WP_AREA1</i>	the second write protection area
<b>Output parameter{out}</b>	
<b>wp_start</b>	first page of write protection area
<b>Output parameter{out}</b>	
<b>wp_end</b>	last page of write protection area
<b>Return value</b>	
<b>uint32_t</b>	state of the return value
<i>INVLD_RETURN_VAL</i> <i>UE</i>	the return value is invalid
<i>VLD_RETURN_VALUE</i>	the return value is valid

Example:

```
/* get configuration of WP area */
```

```
uint32_t start, end, flag;
```

```
flag = ob_write_protection_area_get(WP_AREA0, &start, &end);
```

### ob\_scr\_area\_get

The description of ob\_scr\_area\_get is shown as below:

**Table 3-195. Function ob\_scr\_area\_get**

<b>Function name</b>	ob_scr_area_get
<b>Function prototype</b>	uint32_t ob_scr_area_get(uint32_t scr_area, uint32_t *scr_area_byte_cnt);
<b>Function descriptions</b>	get size of secure area
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Output parameter{out}</b>	
<b>scr_area_byte_cnt</b>	secure area size in byte unit

Return value	
<b>uint32_t</b>	state of the return value
<i>INVLD_RETURN_VAL</i> <i>UE</i>	the return value is invalid
<i>VLD_RETURN_VALUE</i>	the return value is valid

Example:

```
/* get size of secure area */

uint32_t size, flag;

flag = ob_scr_area_get(&size);
```

### ob\_boot\_lock\_get

The description of ob\_boot\_lock\_get is shown as below:

**Table 3-196. Function ob\_boot\_lock\_get**

<b>Function name</b>	ob_boot_lock_get
<b>Function prototype</b>	uint32_t ob_boot_lock_get(void);
<b>Function descriptions</b>	get boot configuration
<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>	boot configuration

Example:

```
/* get boot value */

uint32_t boot_value;

boot_value = ob_boot_lock_get();
```

### fmc\_flag\_get

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

**Table 3-197. Function fmc\_flag\_get**

<b>Function name</b>	fmc_flag_get
<b>Function prototype</b>	FlagStatus fmc_flag_get(uint32_t flag);
<b>Function descriptions</b>	get FMC flag status
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
flag	FMC flag
<i>FMC_FLAG_BUSY</i>	FMC operation is in progress
<i>FMC_FLAG_OBERR</i>	option byte read error
<i>FMC_FLAG_WPERR</i>	erase/program protection error
<i>FMC_FLAG_PGSERR</i>	program sequence error
<i>FMC_FLAG_PGMERR</i>	program size error
<i>FMC_FLAG_PGAERR</i>	program alignment error
<i>FMC_FLAG_FSTPERR</i>	fast program error
<i>FMC_FLAG_RPERR</i>	read protection error
<i>FMC_FLAG_PGERR</i>	program error
<i>FMC_FLAG_OPRERR</i>	operation error
<i>FMC_FLAG_ENDF</i>	FMC end of operation flag
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	status of flag
<i>RESET</i>	reset
<i>SET</i>	set

Example:

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

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

### fmc\_flag\_clear

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

**Table 3-198. Function fmc\_flag\_clear**

Function name	fmc_flag_clear
Function prototype	void fmc_flag_clear(uint32_t flag);
Function descriptions	clear the FMC flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	FMC flag
<i>FMC_FLAG_OBERR</i>	option byte read error
<i>FMC_FLAG_WPERR</i>	erase/program protection error
<i>FMC_FLAG_PGSERR</i>	program sequence error
<i>FMC_FLAG_PGMERR</i>	program size error
<i>FMC_FLAG_PGAERR</i>	program alignment error
<i>FMC_FLAG_FSTPERR</i>	fast program error
<i>FMC_FLAG_RPERR</i>	read protection error

<i>FMC_FLAG_PGERR</i>	program error
<i>FMC_FLAG_OPRERR</i>	operation error
<i>FMC_FLAG_ENDF</i>	FMC end of operation flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
fmc_flag_clear(FMC_FLAG_ENDF);
```

### fmc\_interrupt\_enable

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

**Table 3-199. Function fmc\_interrupt\_enable**

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

Example:

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

```
fmc_interrupt_enable(FMC_INT_END);
```

### fmc\_interrupt\_disable

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

**Table 3-200. Function fmc\_interrupt\_disable**

<b>Function name</b>	fmc_interrupt_disable
<b>Function prototype</b>	void fmc_interrupt_disable(uint32_t interrupt);



<b>Function descriptions</b>	disable FMC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the FMC interrupt
<i>FMC_INT_END</i>	FMC end of operation interrupt
<i>FMC_INT_OPERR</i>	FMC error interrupt
<i>FMC_INT_RPERR</i>	read protection error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
fmc_interrupt_disable(FMC_INT_END);
```

### fmc\_interrupt\_flag\_get

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

**Table 3-201. Function fmc\_interrupt\_flag\_get**

<b>Function name</b>	fmc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus fmc_interrupt_flag_get(fmc_interrupt_flag_enum flag);
<b>Function descriptions</b>	get FMC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	FMC interrupt flag
<i>FMC_INT_END</i>	FMC end of operation interrupt flag
<i>FMC_INT_ERR</i>	FMC error interrupt flag
<i>FMC_INT_RPERR</i>	read protection error interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	status of flag
<i>RESET</i>	reset
<i>SET</i>	set

Example:

```
/* get FMC RPERR error flag */
```

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

## fmc\_interrupt\_flag\_clear

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

**Table 3-202. Function fmc\_interrupt\_flag\_clear**

<b>Function name</b>	fmc_interrupt_flag_clear
<b>Function prototype</b>	void fmc_interrupt_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear FMC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	FMC interrupt flag
<i>FMC_INT_END</i>	FMC end of operation interrupt flag
<i>FMC_INT_ERR</i>	FMC error interrupt flag
<i>FMC_INT_RPERR</i>	read protection error interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear FMC RPERR error flag */
```

```
fmc_interrupt_flag_clear(FMC_INT_FLAG_RPERR);
```

## fmc\_state\_get

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

**Table 3-203. Function fmc\_state\_get**

<b>Function name</b>	fmc_state_get
<b>Function prototype</b>	fmc_state_enum fmc_state_get(void);
<b>Function descriptions</b>	get the FMC state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

<b>fmc_state_enum</b>	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-165. fmc_state_enum</a>
-----------------------	--

Example:

```
/* get the FMC state */

fmc_state_enum state = fmc_state_get( );
```

### fmc\_ready\_wait

The description of fmc\_ready\_wait is shown as below:

**Table 3-204. Function fmc\_ready\_wait**

<b>Function name</b>	fmc_ready_wait
<b>Function prototype</b>	fmc_state_enum fmc_ready_wait(uint32_t timeout);
<b>Function descriptions</b>	check whether FMC is ready or not
<b>Precondition</b>	-
<b>The called functions</b>	fmc_state_get()
<b>Input parameter{in}</b>	
<b>timeout</b>	timeout count
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC,the enum members can refer to members of the enum <a href="#">Table 3-165. fmc_state_enum</a>

Example:

```
/* check whether FMC is ready or not */

fmc_state_enum state = fmc_ready_wait (0x00001000 );
```

## 3.9. 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 [3.9.1](#) the FWDGT firmware functions are introduced in chapter [3.9.2](#).

### 3.9.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

**Table 3-205. FWDGT Registers**

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

### 3.9.2. Descriptions of Peripheral functions

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

**Table 3-206. FWDGT firmware function**

Function name	Function description
fdwgt_write_enable	enable write access to FWDGT_PSC and FWDGT_RLD and FWDGT_WND
fdwgt_write_disable	disable write access to FWDGT_PSC and FWDGT_RLD and FWDGT_WND
fdwgt_enable	start the FWDGT counter
fdwgt_prescaler_value_config	configure the FWDGT counter prescaler value
fdwgt_reload_value_config	configure the FWDGT counter reload value
fdwgt_window_value_config	configure the FWDGT counter window value
fdwgt_counter_reload	reload the counter of FWDGT
fdwgt_config	configure counter reload value, and prescaler divider value
fdwgt_flag_get	get flag state of FWDGT

#### fdwgt\_write\_enable

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

**Table 3-207. Function fdwgt\_write\_enable**

Function name	fdwgt_write_enable
Function prototype	void fdwgt_write_enable(void);
Function descriptions	enable write access to FWDGT_PSC and FWDGT_RLD and FWDGT_WND
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable write access to FWDGT_PSC and FWDGT_RLD and FWDGT_WND */
```

```
fwdgt_write_enable();
```

### fwdgt\_write\_disable

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

**Table 3-208. Function fwdgt\_write\_disable**

Function name	fwdgt_write_disable
Function prototype	void fwdgt_write_disable(void);
Function descriptions	disable write access to FWDGT_PSC,FWDGT_RLD and FWDGT_WND
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable write access to FWDGT_PSC,FWDGT_RLD and FWDGT_WND */
```

```
fwdgt_write_disable();
```

### fwdgt\_enable

The description of fwdgt\_enable is shown as below:

**Table 3-209. Function fwdgt\_enable**

Function name	fwdgt_enable
Function prototype	void fwdgt_enable(void);
Function descriptions	start the FWDGT counter
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
-	-

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 3-210. Function fwdgt\_prescaler\_value\_config**

<b>Function name</b>	fwdgt_prescaler_value_config
<b>Function prototype</b>	ErrStatus fwdgt_prescaler_value_config(uint16_t prescaler_value);
<b>Function descriptions</b>	configure the FWDGT counter clock prescaler value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>prescaler_value</b>	specify prescaler value
<i>FWDGT_PSC_DIVx</i>	FWDGT prescaler set to x(x=4,8,16,32,64,128,256)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR / SUCCESS

Example:

```
/* set FWDGT prescaler to 4 */
```

```
ErrStatus flag;
```

```
flag = fwdgt_prescaler_value_config(FWDGT_PSC_DIV4);
```

### fwdgt\_reload\_value\_config

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

**Table 3-211. Function fwdgt\_reload\_value\_config**

<b>Function name</b>	fwdgt_reload_value_config
<b>Function prototype</b>	ErrStatus fwdgt_reload_value_config(uint16_t reload_value);
<b>Function descriptions</b>	configure the FWDGT counter reload value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>reload_value</b>	reload_value, specify reload value(0x0000 - 0x0FFF)
<b>Output parameter{out}</b>	

-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR / SUCCESS

Example:

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

```
ErrStatus flag;
```

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

### fwdgt\_window\_value\_config

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

**Table 3-212. 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>The called functions</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 0xFF */
```

```
ErrStatus flag;
```

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

### fwdgt\_counter\_reload

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

**Table 3-213. Function fwdgt\_counter\_reload**

<b>Function name</b>	fwdgt_counter_reload
<b>Function prototype</b>	void fwdgt_counter_reload(void);
<b>Function descriptions</b>	reload the counter of FWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reload FWDGT counter */
fwdgt_counter_reload();
```

## fwdgt\_config

The description of fwdgt\_config is shown as below:

**Table 3-214. Function fwdgt\_config**

Function name	fwdgt_config
Function prototype	ErrStatus fwdgt_config(uint16_t reload_value, uint8_t prescaler_div);
Function descriptions	configure counter reload value, and prescaler divider value
Precondition	-
The called functions	-
Input parameter{in}	
reload_value	specify reload value(0x0000 - 0x0FFF)
Input parameter{in}	
prescaler_div	FWDGT prescaler value-
FWDGT_PSC_DIV4	FWDGT prescaler set to 4
FWDGT_PSC_DIV8	FWDGT prescaler set to 8
FWDGT_PSC_DIV16	FWDGT prescaler set to 16
FWDGT_PSC_DIV32	FWDGT prescaler set to 32
FWDGT_PSC_DIV64	FWDGT prescaler set to 64
FWDGT_PSC_DIV128	FWDGT prescaler set to 128
FWDGT_PSC_DIV256	FWDGT prescaler set to 256
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* confiure FWDGT counter clock: 40KHz(IRC40K) / 64 = 0.625 KHz */
fwdgt_config(2*500, FWDGT_PSC_DIV64);
```

## fwdgt\_flag\_get

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



Table 3-215. Function fwdgt\_flag\_get

Function name	fwdgt_flag_get
Function prototype	FlagStatus fwdgt_flag_get(uint16_t flag);
Function descriptions	get flag state of FWDGT
Precondition	-
The called functions	-
Input parameter{in}	
flag	flag to get
FWDGT_FLAG_PUD	a write operation to FWDGT_PSC register is on going
FWDGT_FLAG_RUD	a write operation to FWDGT_RLD register is on going
FWDGT_FLAG_WUD	a write operation to FWDGT_WND register is on going
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

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

```
FlagStatus status;
```

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

## 3.10. GPIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter [3.10.1](#), the GPIO firmware functions are introduced in chapter [3.10.2](#).

### 3.10.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

Table 3-216. GPIO Registers

Registers	Descriptions
GPIOx_CTL	GPIO port control register
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
GPIOx_AFSEL0	GPIO alternate function selected register 0

Registers	Descriptions
GPIOx_AFSEL1	GPIO alternate function selected register 1
GPIOx_BC	GPIO bit clear register
GPIOx_TG	GPIO port bit toggle register

### 3.10.2. Descriptions of Peripheral functions

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

**Table 3-217. GPIO firmware function**

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

#### gpio\_deinit

The description of gpio\_deinit is shown as below:

**Table 3-218. Function gpio\_deinit**

Function name	gpio_deinit
Function prototype	void gpio_deinit(uint32_t gpio_periph);
Function descriptions	reset GPIO port
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C,D,F)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset GPIOA */

gpio_deinit(GPIOA);
```

## gpio\_mode\_set

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

**Table 3-219. 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>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
GPIOx	GPIOx(x = A,B,C,D,F)
<b>Input parameter{in}</b>	
<b>mode</b>	gpio pin mode
GPIO_MODE_INPUT	input mode
GPIO_MODE_OUTPU T	output mode
GPIO_MODE_AF	alternate function mode
GPIO_MODE_ANALO G	analog mode
<b>Input parameter{in}</b>	
<b>pull_up_down</b>	gpio pin with pull-up or pull-down resistor
GPIO_PUPD_NONE	floating mode, no pull-up and pull-down resistors
GPIO_PUPD_PULLUP	with pull-up resistor
GPIO_PUPD_PULLDO WN	with pull-down resistor
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* 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 3-220. Function gpio\_output\_options\_set**

<b>Function name</b>	gpio_output_options_set
<b>Function prototype</b>	void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t pin);
<b>Function descriptions</b>	set GPIO output type and speed
<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>otype</b>	gpio pin output mode
<i>GPIO_OTYPE_PP</i>	push pull mode
<i>GPIO_OTYPE_OD</i>	open drain mode
<b>Input parameter{in}</b>	
<b>speed</b>	gpio pin output max speed
<i>GPIO_OSPEED_LEVE L0</i>	output max speed level 0
<i>GPIO_OSPEED_LEVE L1</i>	output max speed level 1
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config PA0 as 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 3-221. Function gpio\_bit\_set

Function name	gpio_bit_set
Function prototype	void gpio_bit_set(uint32_t gpio_periph,uint32_t pin);
Function descriptions	set GPIO pin bit
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C,D,F)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set PA0 */
```

```
gpio_bit_set(GPIOA, GPIO_PIN_0);
```

### gpio\_bit\_reset

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

Table 3-222. Function gpio\_bit\_reset

Function name	gpio_bit_reset
Function prototype	void gpio_bit_reset(uint32_t gpio_periph,uint32_t pin);
Function descriptions	reset GPIO pin
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C,D,F)
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:

```
/* reset PA0 */

gpio_bit_set(GPIOA, GPIO_PIN_0);
```

## gpio\_bit\_write

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

**Table 3-223. Function gpio\_bit\_write**

Function name	gpio_bit_write
Function prototype	void gpio_bit_write(uint32_t gpio_periph,uint32_t pin,bit_status bit_value);
Function descriptions	write data to the specified GPIO pin
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C,D,F)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	All pins
Input parameter{in}	
bit_value	SET or RESET
RESET	clear the port pin
SET	set the port pin
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write 1 to PA0 */

gpio_bit_write(GPIOA, GPIO_PIN_0, SET);
```

## gpio\_port\_write

The description of gpio\_port\_write is shown as below:

**Table 3-224. Function gpio\_port\_write**

Function name	gpio_port_write
Function prototype	void gpio_port_write(uint32_t gpio_periph,uint16_t data);
Function descriptions	write data to the specified GPIO port
Precondition	-

The called functions	-
Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,F)
Input parameter{in}	
<b>data</b>	specify the value to be written to the port output data register
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

### gpio\_input\_bit\_get

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

**Table 3-225. Function gpio\_input\_bit\_get**

<b>Function name</b>	gpio_input_bit_get
<b>Function prototype</b>	FlagStatus gpio_input_bit_get(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	get GPIO pin input status
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,F)
Input parameter{in}	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get status of PA0 */
```

```
FlagStatus bit_state;
```

```
bit_state = gpio_input_bit_get(GPIOA, GPIO_PIN_0);
```

## gpio\_input\_port\_get

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

**Table 3-226. Function gpio\_input\_port\_get**

<b>Function name</b>	gpio_input_port_get
<b>Function prototype</b>	uint16_t gpio_input_port_get(uint32_t gpio_periph);
<b>Function descriptions</b>	get GPIO all pins 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>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	0x0000-0xFFFF

Example:

```
/* get input value of Port A */

uint16_t port_state;

port_state = gpio_input_port_get(GPIOA);
```

## gpio\_output\_bit\_get

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

**Table 3-227. Function gpio\_output\_bit\_get**

<b>Function name</b>	gpio_output_bit_get
<b>Function prototype</b>	FlagStatus gpio_output_bit_get(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	get GPIO pin output status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,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 3-228. 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 3-229. Function gpio\_af\_set**

<b>Function name</b>	gpio_af_set
<b>Function prototype</b>	void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);
<b>Function descriptions</b>	set GPIO alternate function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x =A,B,C,D,F)

Input parameter{in}	
<b>alt_func_num</b>	GPIO pin af function, please refer to specific device datasheet
<i>GPIO_AF_0</i>	<i>RTC_OUT, CK_OUT0, SWDIO, SWCLK, SPI0, SPI1, USART0, EVENTOUT</i>
<i>GPIO_AF_1</i>	<i>USART0, USART1, USART2, TIMER0, TIMER2, SPI1</i>
<i>GPIO_AF_2</i>	<i>TIMER0, TIMER13, TIMER15, TIMER16</i>
<i>GPIO_AF_3</i>	<i>SPI1, TIMER2, CK_OUT1, USART2</i>
<i>GPIO_AF_4</i>	<i>USART0, USART1, SPI1, TIMER13</i>
<i>GPIO_AF_5</i>	<i>TIMER0, TIMER16, I2S, USART0, SPI1</i>
<i>GPIO_AF_6</i>	<i>I2C0, I2C1, USART1, USART2</i>
<i>GPIO_AF_7</i>	<i>CMP0, CMP1, EVENTOUT, I2C0</i>
<i>GPIO_AF_8</i>	<i>SPI0</i>
<i>GPIO_AF_9</i>	<i>TIMER0, USART1</i>
<i>GPIO_AF_10</i>	<i>SPI0, TIMER0, TIMER15, TIMER16</i>
<i>GPIO_AF_11</i>	<i>CK_OUT1, TIMER0, TIMER2</i>
<i>GPIO_AF_12</i>	<i>USART0, TIMER0, TIMER2</i>
<i>GPIO_AF_13</i>	<i>TIMER2, TIMER13</i>
<i>GPIO_AF_14</i>	<i>USART0, TIMER15, I2C0</i>
<i>GPIO_AF_15</i>	<i>CK_OUT1, TIMER16, EVENTOUT</i>
Input parameter{in}	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	<i>GPIO_PIN_x(x=0..15)</i>
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*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 3-230. Function gpio\_pin\_lock**

<b>Function name</b>	gpio_pin_lock
<b>Function prototype</b>	void gpio_pin_lock(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	lock GPIO pin bit
<b>Precondition</b>	-
<b>The called functions</b>	-
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>	
-	-

Example:

```
/* lock PA0 */
```

```
gpio_pin_lock(GPIOA, GPIO_PIN_0);
```

### gpio\_bit\_toggle

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

**Table 3-231. Function gpio\_bit\_toggle**

<b>Function name</b>	gpio_bit_toggle
<b>Function prototype</b>	void gpio_bit_toggle(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	toggle GPIO pin status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,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>	GPIO_PIN_ALL
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* toggle PA0 */
```

```
gpio_bit_toggle(GPIOA, GPIO_PIN_0);
```

### gpio\_port\_toggle

The description of gpio\_port\_toggle is shown as below:

Table 3-232. Function gpio\_port\_toggle

Function name	gpio_port_toggle
Function prototype	void gpio_port_toggle(uint32_t gpio_periph);
Function descriptions	toggle GPIO port 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	
-	-

Example:

```
/* toggle GPIOA */
gpio_port_toggle(GPIOA);
```

## 3.11. 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 [3.11.1](#), the I2C firmware functions are introduced in chapter [3.11.2](#).

### 3.11.1. Descriptions of Peripheral registers

I2C registers are listed in the table shown as below:

Table 3-233. 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_TIMING	Timing register
I2C_TIMEOUT	Timeout register
I2C_STAT	Status register
I2C_STATC	I2C status clear register
I2C_PEC	PEC register
I2C_RDATA	Receive data register
I2C_TDATA	Transmit data register

Registers	Descriptions
I2C_CTL2	Control register 2

### 3.11.2. Descriptions of Peripheral functions

I2C firmware functions are listed in the table shown as below:

**Table 3-234. I2C firmware function**

Function name	Function description
i2c_deinit	reset I2C
i2c_timing_config	configure the timing parameters
i2c_digital_noise_filter_config	configure digital noise filter
i2c_analog_noise_filter_enable	enable analog noise filter
i2c_analog_noise_filter_disable	disable analog noise filter
i2c_master_clock_config	configure the SCL high and low period of clock in master mode
i2c_master_addressing	configure i2c slave addresss and transfer direction in master mode
i2c_address10_header_enable	10-bit address header executes read direction only in master receive mode
i2c_address10_header_disable	10-bit address header executes complete sequence in master receive mode
i2c_address10_enable	enable 10-bit addressing mode in master mode
i2c_address10_disable	disable 10-bit addressing mode in master mode
i2c_automatic_end_enable	enable I2C automatic end mode in master mode
i2c_automatic_end_disable	disable I2C automatic end mode in master mode
i2c_slave_response_to_gcall_enable	enable the response to a general call
i2c_slave_response_to_gcall_disable	disable the response to a general call
i2c_stretch_scl_low_enable	enable to stretch SCL low when data is not ready in slave mode
i2c_stretch_scl_low_disable	disable to stretch SCL low when data is not ready in slave mode
i2c_address_config	configure i2c slave address
i2c_address_bit_compare_config	define which bits of ADDRESS[7:1] need to compare with the incoming address byte
i2c_address_disable	disable i2c address in slave mode
i2c_second_address_config	configure i2c second slave address
i2c_second_address_disable	disable i2c second address in slave mode
i2c_received_address_get	get received match address in slave mode
i2c_slave_byte_control_enable	enable slave byte control
i2c_slave_byte_control_disable	disable slave byte control
i2c_nack_enable	generate a NACK in slave mode
i2c_wakeup_from_deepsleep_enable	enable wakeup from Deep-sleep mode

Function name	Function description
i2c_wakeup_from_deepsleep_disable	disable wakeup from Deep-sleep 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
i2c_data_receive	I2C receive data
i2c_reload_enable	enable I2C reload mode
i2c_reload_disable	disable I2C reload mode
i2c_transfer_byte_number_config	configure number of bytes to be transferred
i2c_dma_enable	enable I2C DMA for transmission or reception
i2c_dma_disable	disable I2C DMA for transmission or reception
i2c_pec_transfer	I2C transfers PEC value
i2c_pec_enable	enable I2C PEC calculation
i2c_pec_disable	disable I2C PEC calculation
i2c_pec_value_get	get packet error checking value
i2c_smbus_alert_enable	enable SMBus Alert
i2c_smbus_alert_disable	disable SMBus Alert
i2c_smbus_default_addr_enable	enable SMBus device default address
i2c_smbus_default_addr_disable	disable SMBus device default address
i2c_smbus_host_addr_enable	enable SMBus Host address
i2c_smbus_host_addr_disable	disable SMBus Host address
i2c_extended_clock_timeout_enable	enable extended clock timeout detection
i2c_extended_clock_timeout_disable	disable extended clock timeout detection
i2c_clock_timeout_enable	enable clock timeout detection
i2c_clock_timeout_disable	disable clock timeout detection
i2c_bus_timeout_b_config	configure bus timeout B
i2c_bus_timeout_a_config	configure bus timeout A
i2c_idle_clock_timeout_config	configure idle clock timeout detection
i2c_flag_get	get I2C flag status
i2c_flag_clear	clear I2C flag status
i2c_interrupt_enable	enable I2C interrupt
i2c_interrupt_disable	disable I2C interrupt
i2c_interrupt_flag_get	get I2C interrupt flag status
i2c_interrupt_flag_clear	clear I2C interrupt flag status

## Enum i2c\_interrupt\_flag\_enum

Table 3-235. i2c\_interrupt\_flag\_enum

Member name	Function description
I2C_INT_FLAG_TI	transmit interrupt flag
I2C_INT_FLAG_RBNE	I2C_RDATA is not empty during receiving interrupt flag

Member name	Function description
I2C_INT_FLAG_ADDSEND	address received matches in slave mode interrupt flag
I2C_INT_FLAG_NACK	not acknowledge interrupt flag
I2C_INT_FLAG_STPDET	stop condition detected in slave mode interrupt flag
I2C_INT_FLAG_TC	transfer complete in master mode interrupt flag
I2C_INT_FLAG_TCR	transfer complete reload interrupt flag
I2C_INT_FLAG_BERR	bus error interrupt flag
I2C_INT_FLAG_LOSTARB	arbitration lost interrupt flag
I2C_INT_FLAG_OUERR	overrun/underrun error in slave mode interrupt flag
I2C_INT_FLAG_PECERR	PEC error interrupt flag
I2C_INT_FLAG_TIMEOUT	timeout interrupt flag
I2C_INT_FLAG_SMBALT	SMBus Alert interrupt flag

## i2c\_deinit

The description of i2c\_deinit is shown as below:

**Table 3-236. Function i2c\_deinit**

<b>Function name</b>	i2c_deinit
<b>Function prototype</b>	void i2c_deinit(uint32_t i2c_periph);
<b>Function descriptions</b>	reset I2C
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset I2C0 */
i2c_deinit(I2C0);
```

## i2c\_timing\_config

The description of i2c\_timing\_config is shown as below:

**Table 3-237. Function i2c\_timing\_config**

<b>Function name</b>	i2c_timing_config
<b>Function prototype</b>	void i2c_timing_config(uint32_t i2c_periph, uint32_t psc, uint32_t scl_dely, uint32_t sda_dely);
<b>Function descriptions</b>	configure the timing parameters

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>psc</b>	0-0xf, timing prescaler
<b>Input parameter{in}</b>	
<b>scl_dely</b>	0-0xf,data setup time
<b>Input parameter{in}</b>	
<b>sda_dely</b>	0-0xf,data hold time
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the timing parameters */
```

```
i2c_timing_config(I2C0, 0x1, 0x2, 0x1);
```

### i2c\_digital\_noise\_filter\_config

The description of i2c\_digital\_noise\_filter\_config is shown as below:

**Table 3-238. Function i2c\_digital\_noise\_filter\_config**

<b>Function name</b>	i2c_digital_noise_filter_config
<b>Function prototype</b>	void i2c_digital_noise_filter_config(uint32_t i2c_periph, uint32_t filter_length);
<b>Function descriptions</b>	configure digital noise filter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>filter_length</b>	filter_length
<i>FILTER_DISABLE</i>	digital filter is disabled
<i>FILTER_LENGTH_1</i>	digital filter is enabled and filter spikes with a length of up to 1 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_2</i>	digital filter is enabled and filter spikes with a length of up to 2 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_3</i>	digital filter is enabled and filter spikes with a length of up to 3 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_4</i>	digital filter is enabled and filter spikes with a length of up to 4 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_5</i>	digital filter is enabled and filter spikes with a length of up to 5 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_6</i>	digital filter is enabled and filter spikes with a length of up to 6 t <sub>I2CCLK</sub>



<i>FILTER_LENGTH_7</i>	digital filter is enabled and filter spikes with a length of up to 7 $t_{I2CCLK}$
<i>FILTER_LENGTH_8</i>	digital filter is enabled and filter spikes with a length of up to 8 $t_{I2CCLK}$
<i>FILTER_LENGTH_9</i>	digital filter is enabled and filter spikes with a length of up to 9 $t_{I2CCLK}$
<i>FILTER_LENGTH_10</i>	digital filter is enabled and filter spikes with a length of up to 10 $t_{I2CCLK}$
<i>FILTER_LENGTH_11</i>	digital filter is enabled and filter spikes with a length of up to 11 $t_{I2CCLK}$
<i>FILTER_LENGTH_12</i>	digital filter is enabled and filter spikes with a length of up to 12 $t_{I2CCLK}$
<i>FILTER_LENGTH_13</i>	digital filter is enabled and filter spikes with a length of up to 13 $t_{I2CCLK}$
<i>FILTER_LENGTH_14</i>	digital filter is enabled and filter spikes with a length of up to 14 $t_{I2CCLK}$
<i>FILTER_LENGTH_15</i>	digital filter is enabled and filter spikes with a length of up to 15 $t_{I2CCLK}$
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 digital filter filters spikes with a length of up to 1  $t_{I2CCLK}$  */
```

```
i2c_digital_noise_filter_config(I2C0, FILTER_LENGTH_1);
```

### **i2c\_analog\_noise\_filter\_enable**

The description of i2c\_analog\_noise\_filter\_enable is shown as below:

**Table 3-239. Function i2c\_analog\_noise\_filter\_enable**

<b>Function name</b>	i2c_analog_noise_filter_enable
<b>Function prototype</b>	void i2c_analog_noise_filter_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable analog noise filter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable analog noise filter */
```

```
i2c_analog_noise_filter_enable(I2C0);
```

### **i2c\_analog\_noise\_filter\_disable**

The description of i2c\_analog\_noise\_filter\_disable is shown as below:

Table 3-240. Function i2c\_analog\_noise\_filter\_disable

Function name	i2c_analog_noise_filter_disable
Function prototype	void i2c_analog_noise_filter_disable(uint32_t i2c_periph);
Function descriptions	disable analog noise filter
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable analog noise filter */
i2c_analog_noise_filter_disable(I2C0);
```

### i2c\_master\_clock\_config

The description of i2c\_master\_clock\_config is shown as below:

Table 3-241. Function i2c\_master\_clock\_config

Function name	i2c_master_clock_config
Function prototype	void i2c_master_clock_config(uint32_t i2c_periph, uint32_t sclh, uint32_t scll);
Function descriptions	configure the SCL high and low period of clock in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
sclh	0-0xff, SCL high period
Input parameter{in}	
scll	0-0xff, SCL low period
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the SCL and SDA period of clock in master mode */
```

```
i2c_master_clock_config(I2C0, 0x0f, 0x0f);
```

## i2c\_master\_addressing

The description of i2c\_master\_addressing is shown as below:

**Table 3-242. 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 address, uint32_t trans_direction);
<b>Function descriptions</b>	configure i2c slave addresss and transfer direction in master mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>address</b>	0-0x3ff except reserved address, I2C slave address to be sent
<b>Input parameter{in}</b>	
<b>trans_direction</b>	I2C transfer direction in master mode
<i>I2C_MASTER_TRANSMIT</i>	master transmit
<i>I2C_MASTER_RECEIVE</i>	master receive
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* send slave address to I2C bus */
```

```
i2c_master_addressing(I2C0, 0x82, I2C_MASTER_TRANSMIT);
```

## i2c\_address10\_header\_enable

The description of i2c\_address10\_header\_enable is shown as below:

**Table 3-243. Function i2c\_address10\_header\_enable**

<b>Function name</b>	i2c_address10_header_enable
<b>Function prototype</b>	void i2c_address10_header_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	10-bit address header executes read direction only in master receive mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* 10-bit address header executes read direction only in master receive mode */
```

```
i2c_address10_header_enable(I2C0);
```

### i2c\_address10\_header\_disable

The description of i2c\_address10\_header\_disable is shown as below:

**Table 3-244. Function i2c\_address10\_header\_disable**

<b>Function name</b>	i2c_address10_header_disable
<b>Function prototype</b>	void i2c_address10_header_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	10-bit address header executes complete sequence in master receive mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* 10-bit address header executes complete sequence in master receive mode */
```

```
i2c_address10_header_disable(I2C0);
```

### i2c\_address10\_enable

The description of i2c\_address10\_enable is shown as below:

**Table 3-245. Function i2c\_address10\_enable**

<b>Function name</b>	i2c_address10_enable
<b>Function prototype</b>	void i2c_address10_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable 10-bit addressing mode in master mode
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable 10-bit addressing mode in master mode */
```

```
i2c_address10_enable(I2C0);
```

### i2c\_address10\_disable

The description of i2c\_address10\_disable is shown as below:

**Table 3-246. Function i2c\_address10\_disable**

<b>Function name</b>	i2c_address10_disable
<b>Function prototype</b>	void i2c_address10_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable 10-bit addressing mode in master mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable 10-bit addressing mode in master mode */
```

```
i2c_address10_disable(I2C0);
```

### i2c\_automatic\_end\_enable

The description of i2c\_automatic\_end\_enable is shown as below:

**Table 3-247. Function i2c\_automatic\_end\_enable**

<b>Function name</b>	i2c_automatic_end_enable
<b>Function prototype</b>	void i2c_automatic_end_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable I2C automatic end mode in master mode
<b>Precondition</b>	-

The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_enable(I2C0);
```

### i2c\_automatic\_end\_disable

The description of i2c\_automatic\_end\_disable is shown as below:

**Table 3-248. Function i2c\_automatic\_end\_disable**

Function name	i2c_automatic_end_disable
Function prototype	void i2c_automatic_end_disable(uint32_t i2c_periph);
Function descriptions	disable I2C automatic end mode in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_disable(I2C0);
```

### i2c\_slave\_response\_to\_gcall\_enable

The description of i2c\_slave\_response\_to\_gcall\_enable is shown as below:

**Table 3-249. Function i2c\_slave\_response\_to\_gcall\_enable**

Function name	i2c_slave_response_to_gcall_enable
Function prototype	void i2c_slave_response_to_gcall_enable(uint32_t i2c_periph);
Function descriptions	enable the response to a general call

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the response to a general call */
```

```
i2c_slave_response_to_gcall_enable(I2C0);
```

### **i2c\_slave\_response\_to\_gcall\_disable**

The description of i2c\_slave\_response\_to\_gcall\_disable is shown as below:

**Table 3-250. Function i2c\_slave\_response\_to\_gcall\_disable**

<b>Function name</b>	i2c_slave_response_to_gcall_disable
<b>Function prototype</b>	void i2c_slave_response_to_gcall_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable the response to a general call
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the response to a general call */
```

```
i2c_slave_response_to_gcall_disable(I2C0);
```

### **i2c\_stretch\_scl\_low\_enable**

The description of i2c\_stretch\_scl\_low\_enable is shown as below:

**Table 3-251. Function i2c\_stretch\_scl\_low\_enable**

<b>Function name</b>	i2c_stretch_scl_low_enable
<b>Function prototype</b>	void i2c_stretch_scl_low_enable(uint32_t i2c_periph);

<b>Function descriptions</b>	enable to stretch SCL low when data is not ready in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable to stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_enable(I2C0);
```

### i2c\_stretch\_scl\_low\_disable

The description of i2c\_stretch\_scl\_low\_disable is shown as below:

**Table 3-252. Function i2c\_stretch\_scl\_low\_disable**

<b>Function name</b>	i2c_stretch_scl_low_disable
<b>Function prototype</b>	void i2c_stretch_scl_low_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable to stretch SCL low when data is not ready in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable to stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_disable(I2C0);
```

### i2c\_address\_config

The description of i2c\_address\_config is shown as below:

**Table 3-253. Function i2c\_address\_config**

<b>Function name</b>	i2c_address_config
----------------------	--------------------



<b>Function prototype</b>	void i2c_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_format);
<b>Function descriptions</b>	configure i2c slave address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>address</b>	I2C address
<b>Input parameter{in}</b>	
<b>addr_format</b>	7bits or 10bits
<i>I2C_ADDFORMAT_7BITS</i>	7bits
<i>I2C_ADDFORMAT_10BITS</i>	10bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure i2c slave address */
```

```
i2c_address_config(I2C0, 0x82, I2C_ADDFORMAT_7BITS);
```

### i2c\_address\_bit\_compare\_config

The description of i2c\_address\_bit\_compare\_config is shown as below:

**Table 3-254. Function i2c\_address\_bit\_compare\_config**

<b>Function name</b>	i2c_address_bit_compare_config
<b>Function prototype</b>	void i2c_address_bit_compare_config(uint32_t i2c_periph, uint32_t compare_bits);
<b>Function descriptions</b>	define which bits of ADDRESS[7:1] need to compare with the incoming address byte
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>compare_bits</b>	the bits need to compare
<i>ADDRESS_BIT1_COM</i>	address bit1 needs compare

<i>PARE</i>	
<i>ADDRESS_BIT2_COM</i> <i>PARE</i>	address bit2 needs compare
<i>ADDRESS_BIT3_COM</i> <i>PARE</i>	address bit3 needs compare
<i>ADDRESS_BIT4_COM</i> <i>PARE</i>	address bit4 needs compare
<i>ADDRESS_BIT5_COM</i> <i>PARE</i>	address bit5 needs compare
<i>ADDRESS_BIT6_COM</i> <i>PARE</i>	address bit6 needs compare
<i>ADDRESS_BIT7_COM</i> <i>PARE</i>	address bit7 needs compare
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* bit 1 of ADDRESS[7:1] need to compare with the incoming address byte */
i2c_address_bit_compare_config(I2C0, ADDRESS_BIT1_COMPARE);
```

### i2c\_address\_disable

The description of i2c\_address\_disable is shown as below:

**Table 3-255. Function i2c\_address\_disable**

<b>Function name</b>	i2c_address_disable
<b>Function prototype</b>	void i2c_address_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable i2c address in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable i2c address in slave mode */
i2c_address_disable(I2C0);
```

## i2c\_second\_address\_config

The description of i2c\_second\_address\_config is shown as below:

**Table 3-256. Function i2c\_second\_address\_config**

<b>Function name</b>	i2c_second_address_config
<b>Function prototype</b>	void i2c_second_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_mask);
<b>Function descriptions</b>	configure i2c second slave address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>address</b>	I2C address
<b>Input parameter{in}</b>	
<b>addr_mask</b>	the bits not need to compare
ADDRESS2_NO_MAS K	no mask, all the bits must be compared
ADDRESS2_MASK_BIT1	ADDRESS2[1] is masked, only ADDRESS2[7:2] are compared
ADDRESS2_MASK_BIT1_2	ADDRESS2[2:1] is masked, only ADDRESS2[7:3] are compared
ADDRESS2_MASK_BIT1_3	ADDRESS2[3:1] is masked, only ADDRESS2[7:4] are compared
ADDRESS2_MASK_BIT1_4	ADDRESS2[4:1] is masked, only ADDRESS2[7:5] are compared
ADDRESS2_MASK_BIT1_5	ADDRESS2[5:1] is masked, only ADDRESS2[7:6] are compared
ADDRESS2_MASK_BIT1_6	ADDRESS2[6:1] is masked, only ADDRESS2[7] are compared
ADDRESS2_MASK_ALL	all the ADDRESS2[7:1] bits are masked
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure i2c second slave address */
i2c_second_address_config(I2C0, 0x82, ADDRESS2_MASK_BIT1_2);
```

## i2c\_second\_address\_disable

The description of i2c\_second\_address\_disable is shown as below:

**Table 3-257. Function i2c\_second\_address\_disable**

<b>Function name</b>	i2c_second_address_disable
<b>Function prototype</b>	void i2c_second_address_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable i2c second address in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable i2c second address in slave mode */
```

```
i2c_second_address_disable(I2C0);
```

## i2c\_received\_address\_get

The description of i2c\_received\_address\_get is shown as below:

**Table 3-258. Function i2c\_received\_address\_get**

<b>Function name</b>	i2c_received_address_get
<b>Function prototype</b>	uint32_t i2c_received_address_get(uint32_t i2c_periph);
<b>Function descriptions</b>	get received match address in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	0x00..0x7f

Example:

```
/* get received match address in slave mode */
```

```
uint32_t address;
```

```
address = i2c_received_address_get(I2C0);
```

### i2c\_slave\_byte\_control\_enable

The description of i2c\_slave\_byte\_control\_enable is shown as below:

**Table 3-259. Function i2c\_slave\_byte\_control\_enable**

<b>Function name</b>	i2c_slave_byte_control_enable
<b>Function prototype</b>	void i2c_slave_byte_control_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable slave byte control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable slave byte control */
```

```
i2c_slave_byte_control_enable(I2C0);
```

### i2c\_slave\_byte\_control\_disable

The description of i2c\_slave\_byte\_control\_disable is shown as below:

**Table 3-260. Function i2c\_slave\_byte\_control\_disable**

<b>Function name</b>	i2c_slave_byte_control_disable
<b>Function prototype</b>	void i2c_slave_byte_control_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable slave byte control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable slave byte control */
```

```
i2c_slave_byte_control_disable(I2C0);
```

## i2c\_nack\_enable

The description of i2c\_nack\_enable is shown as below:

**Table 3-261. Function i2c\_nack\_enable**

<b>Function name</b>	i2c_nack_enable
<b>Function prototype</b>	void i2c_nack_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a NACK in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* generate a NACK in slave mode */
```

```
i2c_nack_enable(I2C0);
```

## i2c\_wakeup\_from\_deepsleep\_enable

The description of i2c\_wakeup\_from\_deepsleep\_enable is shown as below:

**Table 3-262. Function i2c\_wakeup\_from\_deepsleep\_enable**

<b>Function name</b>	i2c_wakeup_from_deepsleep_enable
<b>Function prototype</b>	void i2c_wakeup_from_deepsleep_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable wakeup from Deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable wakeup from Deep-sleep mode */
```

```
i2c_wakeup_from_deepsleep_enable(I2C0);
```

### i2c\_wakeup\_from\_deepsleep\_disable

The description of i2c\_wakeup\_from\_deepsleep\_disable is shown as below:

**Table 3-263. Function i2c\_wakeup\_from\_deepsleep\_disable**

<b>Function name</b>	i2c_wakeup_from_deepsleep_disable
<b>Function prototype</b>	void i2c_wakeup_from_deepsleep_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable wakeup from Deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable wakeup from Deep-sleep mode */
```

```
i2c_wakeup_from_deepsleep_disable(I2C0);
```

### i2c\_enable

The description of i2c\_enable is shown as below:

**Table 3-264. Function i2c\_enable**

<b>Function name</b>	i2c_enable
<b>Function prototype</b>	void i2c_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 */
```

```
i2c_enable(I2C0);
```

## i2c\_disable

The description of i2c\_disable is shown as below:

**Table 3-265. 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)
<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 3-266. 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)
<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 3-267. 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)
<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 3-268. 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
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>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 3-269. 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
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>Uint8_t</b>	0x00..0xff

Example:

```
/* I2C0 receive data */
```

```
Uint8_t i2c_receiver;
```

```
i2c_receiver = i2c_data_receive(I2C0);
```

## i2c\_reload\_enable

The description of i2c\_reload\_enable is shown as below:

**Table 3-270. Function i2c\_reload\_enable**

<b>Function name</b>	i2c_reload_enable
<b>Function prototype</b>	void i2c_reload_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable I2C reload mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C reload mode */
```

```
i2c_reload_enable(I2C0);
```

### i2c\_reload\_disable

The description of i2c\_reload\_disable is shown as below:

**Table 3-271. Function i2c\_reload\_disable**

<b>Function name</b>	i2c_reload_disable
<b>Function prototype</b>	void i2c_reload_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable I2C reload mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C reload mode */
```

```
i2c_reload_disable(I2C0);
```

### i2c\_transfer\_byte\_number\_config

The description of i2c\_transfer\_byte\_number\_config is shown as below:

**Table 3-272. Function i2c\_transfer\_byte\_number\_config**

<b>Function name</b>	i2c_transfer_byte_number_config
<b>Function prototype</b>	void i2c_transfer_byte_number_config(uint32_t i2c_periph, uint8_t byte_number);
<b>Function descriptions</b>	configure number of bytes to be transferred
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>byte_number</b>	0x00-0xff, number of bytes to be transferred
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* configure number of bytes to be transferred */
```

```
i2c_transfer_byte_number_config(I2C0, 0xff);
```

### i2c\_dma\_enable

The description of i2c\_dma\_enable is shown as below:

**Table 3-273. Function i2c\_dma\_enable**

<b>Function name</b>	i2c_dma_enable
<b>Function prototype</b>	void i2c_dma_enable(uint32_t i2c_periph, uint8_t dma);
<b>Function descriptions</b>	enable I2C DMA for transmission or reception
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>dma</b>	I2C DMA
<i>I2C_DMA_TRANSMIT</i>	transmit data using DMA
<i>I2C_DMA_RECEIVE</i>	receive data using DMA
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C DMA for transmission or reception */
```

```
i2c_dma_enable(I2C0, I2C_DMA_RECEIVE);
```

### i2c\_dma\_disable

The description of i2c\_dma\_disable is shown as below:

**Table 3-274. Function i2c\_dma\_disable**

<b>Function name</b>	i2c_dma_disable
<b>Function prototype</b>	void i2c_dma_disable(uint32_t i2c_periph, uint8_t dma);
<b>Function descriptions</b>	disable I2C DMA for transmission or reception
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>dma</b>	I2C DMA
<i>I2C_DMA_TRANSMIT</i>	transmit data using DMA
<i>I2C_DMA_RECEIVE</i>	receive data using DMA
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C DMA for transmission or reception */
```

```
i2c_dma_disable(I2C0, I2C_DMA_RECEIVE);
```

### i2c\_pec\_transfer

The description of i2c\_pec\_transfer is shown as below:

**Table 3-275. Function i2c\_pec\_transfer**

<b>Function name</b>	i2c_pec_transfer
<b>Function prototype</b>	void i2c_pec_transfer(uint32_t i2c_periph);
<b>Function descriptions</b>	I2C transfers PEC value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C transfers PEC value */
```

```
i2c_pec_transfer(I2C0);
```

### i2c\_pec\_enable

The description of i2c\_pec\_enable is shown as below:

Table 3-276. Function i2c\_pec\_enable

Function name	i2c_pec_enable
Function prototype	void i2c_pec_enable(uint32_t i2c_periph);
Function descriptions	enable I2C PEC calculation
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C PEC calculation */
i2c_pec_enable(I2C0);
```

### i2c\_pec\_disable

The description of i2c\_pec\_disable is shown as below:

Table 3-277. Function i2c\_pec\_disable

Function name	i2c_pec_disable
Function prototype	void i2c_pec_disable(uint32_t i2c_periph);
Function descriptions	disable I2C PEC calculation
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C PEC calculation */
i2c_pec_disable(I2C0);
```

### i2c\_pec\_value\_get

The description of i2c\_pec\_value\_get is shown as below:

Table 3-278. Function i2c\_pec\_value\_get

Function name	i2c_pec_value_get
Function prototype	uint32_t i2c_pec_value_get(uint32_t i2c_periph);
Function descriptions	get packet error checking value
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0)
Output parameter{out}	
-	-
Return value	
uint32_t	PEC value

Example:

```
/* I2C0 get packet error checking value */
uint32_t pec_value;
pec_value = i2c_pec_value_get(I2C0);
```

### i2c\_smbus\_alert\_enable

The description of i2c\_smbus\_alert\_enable is shown as below:

Table 3-279. Function i2c\_smbus\_alert\_enable

Function name	i2c_smbus_alert_enable
Function prototype	void i2c_smbus_alert_enable(uint32_t i2c_periph);
Function descriptions	enable SMBus Alert
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SMBus Alert */
i2c_smbus_alert_enable(I2C0);
```

## i2c\_smbus\_alert\_disable

The description of i2c\_smbus\_alert\_disable is shown as below:

**Table 3-280. Function i2c\_smbus\_alert\_disable**

<b>Function name</b>	i2c_smbus_alert_disable
<b>Function prototype</b>	void i2c_smbus_alert_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable SMBus Alert
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SMBus Alert */
i2c_smbus_alert_disable(I2C0);
```

## i2c\_smbus\_default\_addr\_enable

The description of i2c\_smbus\_default\_addr\_enable is shown as below:

**Table 3-281. Function i2c\_smbus\_default\_addr\_enable**

<b>Function name</b>	i2c_smbus_default_addr_enable
<b>Function prototype</b>	void i2c_smbus_default_addr_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable SMBus device default address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SMBus device default address */
i2c_smbus_default_addr_enable(I2C0);
```



## i2c\_smbus\_default\_addr\_disable

The description of i2c\_smbus\_default\_addr\_disable is shown as below:

**Table 3-282. Function i2c\_smbus\_default\_addr\_disable**

<b>Function name</b>	i2c_smbus_default_addr_disable
<b>Function prototype</b>	void i2c_smbus_default_addr_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable SMBus device default address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SMBus device default address */
```

```
i2c_smbus_default_addr_disable(I2C0);
```

## i2c\_smbus\_host\_addr\_enable

The description of i2c\_smbus\_host\_addr\_enable is shown as below:

**Table 3-283. Function i2c\_smbus\_host\_addr\_enable**

<b>Function name</b>	i2c_smbus_host_addr_enable
<b>Function prototype</b>	void i2c_smbus_host_addr_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable SMBus Host address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SMBus Host address */
```

```
i2c_smbus_host_addr_enable(I2C0);
```

## i2c\_smbus\_host\_addr\_disable

The description of i2c\_smbus\_host\_addr\_disable is shown as below:

**Table 3-284. Function i2c\_smbus\_host\_addr\_disable**

<b>Function name</b>	i2c_smbus_host_addr_disable
<b>Function prototype</b>	void i2c_smbus_host_addr_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable SMBus Host address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SMBus Host address */
```

```
i2c_smbus_host_addr_disable(I2C0);
```

## i2c\_extended\_clock\_timeout\_enable

The description of i2c\_extended\_clock\_timeout\_enable is shown as below:

**Table 3-285. Function i2c\_extended\_clock\_timeout\_enable**

<b>Function name</b>	i2c_extended_clock_timeout_enable
<b>Function prototype</b>	void i2c_extended_clock_timeout_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable extended clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable extended clock timeout detection */
```

```
i2c_extended_clock_timeout_enable(I2C0);
```

## i2c\_extended\_clock\_timeout\_disable

The description of i2c\_extended\_clock\_timeout\_disable is shown as below:

**Table 3-286. Function i2c\_extended\_clock\_timeout\_disable**

<b>Function name</b>	i2c_extended_clock_timeout_disable
<b>Function prototype</b>	void i2c_extended_clock_timeout_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable extended clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable extended clock timeout detection */
```

```
i2c_extended_clock_timeout_disable(I2C0);
```

## i2c\_clock\_timeout\_enable

The description of i2c\_clock\_timeout\_enable is shown as below:

**Table 3-287. Function i2c\_clock\_timeout\_enable**

<b>Function name</b>	i2c_clock_timeout_enable
<b>Function prototype</b>	void i2c_clock_timeout_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable clock timeout detection */
```

```
i2c_clock_timeout_enable(I2C0);
```

## i2c\_clock\_timeout\_disable

The description of i2c\_clock\_timeout\_disable is shown as below:

**Table 3-288. Function i2c\_clock\_timeout\_disable**

<b>Function name</b>	i2c_clock_timeout_disable
<b>Function prototype</b>	void i2c_clock_timeout_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable clock timeout detection */
```

```
i2c_clock_timeout_disable(I2C0);
```

## i2c\_bus\_timeout\_b\_config

The description of i2c\_bus\_timeout\_b\_config is shown as below:

**Table 3-289. Function i2c\_bus\_timeout\_b\_config**

<b>Function name</b>	i2c_bus_timeout_b_config
<b>Function prototype</b>	void i2c_bus_timeout_b_config(uint32_t i2c_periph, uint32_t timeout);
<b>Function descriptions</b>	configure bus timeout B
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>timeout</b>	0-0xffff, bus timeout B
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure bus timeout B */
```

```
i2c_bus_timeout_b_config(I2C0, 0xff);
```

## i2c\_bus\_timeout\_a\_config

The description of i2c\_bus\_timeout\_a\_config is shown as below:

**Table 3-290. Function i2c\_bus\_timeout\_a\_config**

<b>Function name</b>	i2c_bus_timeout_a_config
<b>Function prototype</b>	void i2c_bus_timeout_a_config(uint32_t i2c_periph, uint32_t timeout);
<b>Function descriptions</b>	configure bus timeout A
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>timeout</b>	0-0xffff, bus timeout A
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure bus timeout A */
```

```
i2c_bus_timeout_a_config(I2C0, 0xff);
```

## i2c\_idle\_clock\_timeout\_config

The description of i2c\_idle\_clock\_timeout\_config is shown as below:

**Table 3-291. Function i2c\_idle\_clock\_timeout\_config**

<b>Function name</b>	i2c_idle_clock_timeout_config
<b>Function prototype</b>	void i2c_idle_clock_timeout_config(uint32_t i2c_periph, uint32_t timeout);
<b>Function descriptions</b>	configure idle clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>timeout</b>	bus timeout A
<i>BUSTOA_DETECT_SCL_LOW</i>	BUSTOA is used to detect SCL low timeout
<i>BUSTOA_DETECT_ID</i>	BUSTOA is used to detect both SCL and SDA high timeout when the bus is

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

Example:

```
/* configure idle clock timeout detection */
```

```
i2c_idle_clock_timeout_config(I2C0, BUSTOA_DETECT_SCL_LOW);
```

### i2c\_flag\_get

The description of i2c\_flag\_get is shown as below:

**Table 3-292. Function i2c\_flag\_get**

<b>Function name</b>	i2c_flag_get
<b>Function prototype</b>	FlagStatus i2c_flag_get(uint32_t i2c_periph, uint32_t flag);
<b>Function descriptions</b>	get I2C flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>flag</b>	I2C flags
<i>I2C_FLAG_TBE</i>	I2C_TDATA is empty during transmitting
<i>I2C_FLAG_TI</i>	transmit interrupt
<i>I2C_FLAG_RBNE</i>	I2C_RDATA is not empty during receiving
<i>I2C_FLAG_ADDSEND</i>	address received matches in slave mode
<i>I2C_FLAG_NACK</i>	not acknowledge flag
<i>I2C_FLAG_STPDET</i>	STOP condition detected in slave mode
<i>I2C_FLAG_TC</i>	transfer complete in master mode
<i>I2C_FLAG_TCR</i>	transfer complete reload
<i>I2C_FLAG_BERR</i>	bus error
<i>I2C_FLAG_LOSTARB</i>	arbitration Lost
<i>I2C_FLAG_OUERR</i>	overrun/underrun error in slave mode
<i>I2C_FLAG_PECERR</i>	PEC error
<i>I2C_FLAG_TIMEOUT</i>	timeout flag
<i>I2C_FLAG_SMBALT</i>	SMBus Alert
<i>I2C_FLAG_I2CBSY</i>	busy flag
<i>I2C_FLAG_TR</i>	whether the I2C is a transmitter or a receiver in slave mode
<b>Output parameter{out}</b>	
-	-

Return value	
FlagStatus	SET / RESET

Example:

```
/* get I2C flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_flag_get(I2C0, I2C_FLAG_TBE);
```

### i2c\_flag\_clear

The description of i2c\_flag\_clear is shown as below:

**Table 3-293. Function i2c\_flag\_clear**

<b>Function name</b>	i2c_flag_clear
<b>Function prototype</b>	void i2c_flag_clear(uint32_t i2c_periph, uint32_t flag);
<b>Function descriptions</b>	clear I2C flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
<b>flag</b>	I2C flags
I2C_FLAG_ADDSEND	address received matches in slave mode
I2C_FLAG_NACK	not acknowledge flag
I2C_FLAG_STPDET	STOP condition detected in slave mode
I2C_FLAG_BERR	bus error
I2C_FLAG_LOSTARB	arbitration Lost
I2C_FLAG_OUERR	overflow/underrun error in slave mode
I2C_FLAG_PECERR	PEC error
I2C_FLAG_TIMEOUT	timeout flag
I2C_FLAG_SMBALT	SMBus Alert
<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 3-294. Function i2c\_interrupt\_enable**

<b>Function name</b>	i2c_interrupt_enable
<b>Function prototype</b>	void i2c_interrupt_enable(uint32_t i2c_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable I2C interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>interrupt</b>	I2C interrupts
<i>I2C_INT_ERR</i>	error interrupt
<i>I2C_INT_TC</i>	transfer complete interrupt
<i>I2C_INT_STPDET</i>	stop detection interrupt
<i>I2C_INT_NACK</i>	not acknowledge received interrupt
<i>I2C_INT_ADDM</i>	address match interrupt
<i>I2C_INT_RBNE</i>	receive interrupt
<i>I2C_INT_TI</i>	transmit interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 transmit interrupt */
```

```
i2c_interrupt_enable(I2C0, I2C_INT_TI);
```

## i2c\_interrupt\_disable

The description of i2c\_interrupt\_disable is shown as below:

**Table 3-295. Function i2c\_interrupt\_disable**

<b>Function name</b>	i2c_interrupt_disable
<b>Function prototype</b>	void i2c_interrupt_disable(uint32_t i2c_periph, uint32_t interrupt);
<b>Function descriptions</b>	disable I2C interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)



Input parameter{in}	
<b>interrupt</b>	I2C interrupts
<i>I2C_INT_ERR</i>	error interrupt
<i>I2C_INT_TC</i>	transfer complete interrupt
<i>I2C_INT_STPDET</i>	stop detection interrupt
<i>I2C_INT_NACK</i>	not acknowledge received interrupt
<i>I2C_INT_ADDM</i>	address match interrupt
<i>I2C_INT_RBNE</i>	receive interrupt
<i>I2C_INT_TI</i>	transmit interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C0 transmit interrupt */
```

```
i2c_interrupt_disable(I2C0, I2C_INT_TI);
```

### i2c\_interrupt\_flag\_get

The description of i2c\_interrupt\_flag\_get is shown as below:

**Table 3-296. Function i2c\_interrupt\_flag\_get**

<b>Function name</b>	i2c_interrupt_flag_get
<b>Function prototype</b>	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	get I2C interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
<b>int_flag</b>	I2C interrupt flags, refer to <a href="#">Table 3-235. i2c_interrupt_flag_enum</a> .
<i>I2C_INT_FLAG_TI</i>	transmit interrupt flag
<i>I2C_INT_FLAG_RBNE</i>	I2C_RDATA is not empty during receiving interrupt flag
<i>I2C_INT_FLAG_ADDS END</i>	address received matches in slave mode interrupt flag
<i>I2C_INT_FLAG_NACK</i>	not acknowledge interrupt flag
<i>I2C_INT_FLAG_STPD ET</i>	stop condition detected in slave mode interrupt flag
<i>I2C_INT_FLAG_TC</i>	transfer complete in master mode interrupt flag
<i>I2C_INT_FLAG_TCR</i>	transfer complete reload interrupt flag

<i>I2C_INT_FLAG_BERR</i>	bus error interrupt flag
<i>I2C_INT_FLAG_LOSTARB</i>	arbitration lost interrupt flag
<i>I2C_INT_FLAG_OVERR</i>	overflow/underrun error in slave mode interrupt flag
<i>I2C_INT_FLAG_PECERR</i>	PEC error interrupt flag
<i>I2C_INT_FLAG_TIMEOUT</i>	timeout interrupt flag
<i>I2C_INT_FLAG_SMBALERT</i>	SMBus Alert interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get I2C interrupt flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_interrupt_flag_get(I2C0, I2C_INT_FLAG_TI);
```

### i2c\_interrupt\_flag\_clear

The description of i2c\_interrupt\_flag\_clear is shown as below:

**Table 3-297. 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)
<b>Input parameter{in}</b>	
<b>int_flag</b>	I2C interrupt flags, refer to <a href="#">Table 3-235. i2c_interrupt_flag_enum</a> .
<i>I2C_INT_FLAG_ADDS</i> <i>END</i>	address received matches in slave mode interrupt flag
<i>I2C_INT_FLAG_NACK</i>	not acknowledge interrupt flag
<i>I2C_INT_FLAG_STPDET</i> <i>ET</i>	stop condition detected in slave mode interrupt flag
<i>I2C_INT_FLAG_BERR</i>	bus error interrupt flag

<i>I2C_INT_FLAG_LOSTA</i> <i>RB</i>	arbitration lost interrupt flag
<i>I2C_INT_FLAG_OUER</i> <i>R</i>	overflow/underrun error in slave mode interrupt flag
<i>I2C_INT_FLAG_PEC</i> <i>RR</i>	PEC error interrupt flag
<i>I2C_INT_FLAG_TIMEO</i> <i>UT</i>	timeout interrupt flag
<i>I2C_INT_FLAG_SMBA</i> <i>LT</i>	SMBus Alert interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear a bus error flag */
```

```
i2c_interrupt_flag_clear(I2C0, I2C_INT_FLAG_BERR);
```

## 3.12. MISC

MISC is a software package that provide the interfaces for NVIC and SysTick. The NVIC and SysTick registers are listed in chapter [3.12.1](#), the MISC firmware functions are introduced in chapter [3.12.2](#).

### 3.12.1. Descriptions of Peripheral registers

**Table 3-298. NVIC Registers**

Registers	Descriptions
ISER <sup>(1)</sup>	Interrupt Set Enable Register
ICER <sup>(1)</sup>	Interrupt Clear Enable Register
ISPR <sup>(1)</sup>	Interrupt Set Pending Register
ICPR <sup>(1)</sup>	Interrupt Clear Pending Register
IABR <sup>(1)</sup>	Interrupt Active bit Register
ITNS <sup>(1)</sup>	Interrupt Non-Secure State Register
IPR <sup>(1)</sup>	Interrupt Priority Register
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

Registers	Descriptions
SHPR <sup>(2)</sup>	System Handlers Priority Registers
SHCSR <sup>(2)</sup>	System Handler Control and State Register

1. refer to the structure NVIC\_Type, is defined in the core\_cm23.h file

2. refer to the structure SCB\_Type, is defined in the core\_cm23.h file

**Table 3-299. 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\_cm23.h file

### 3.12.2. Descriptions of Peripheral functions

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

**Table 3-300. MISC firmware function**

Function name	Function description
nvic_irq_enable	enable NVIC interrupt request
nvic_irq_disable	disable NVIC interrupt request
nvic_vector_table_set	set the NVIC vector table base address
nvic_system_reset	initiates a system reset request to reset the MCU
system_lowpower_set	set the state of the low power mode
system_lowpower_reset	reset the state of the low power mode
systick_clksource_set	set the systick clock source

### Enum IRQn\_Type

**Table 3-301. IRQn\_Type**

Member name	Function description
WWDGT_IRQn	window watchDog timer interrupt
TIMESTAMP_IRQn	RTC TimeStamp interrupt
FMC_IRQn	FMC interrupt
RCU_IRQn	RCU interrupt
EXTI0_IRQn	EXTI line 0 interrupts
EXTI1_IRQn	EXTI line 1 interrupts
EXTI2_IRQn	EXTI line 2 interrupts
EXTI3_IRQn	EXTI line 3 interrupts
EXTI4_IRQn	EXTI line 4 interrupts
DMA_Channel0_IRQn	DMA channel 0 interrupt
DMA_Channel1_IRQn	DMA channel 1 interrupts
DMA_Channel2_IRQn	DMA channel 2 interrupts

Member name	Function description
ADC_IRQn	ADC interrupts
USART0_IRQn	USART0 interrupt
USART1_IRQn	USART1 interrupt
USART2_IRQn	USART2 interrupt
I2C0_EV_IRQn	I2C0 event interrupt
I2C0_ER_IRQn	I2C0 error interrupt
I2C1_EV_IRQn	I2C1 event interrupt
I2C1_ER_IRQn	I2C1 error interrupt
SPI0_IRQn	SPI0 interrupt
SPI1_IRQn	SPI1 interrupt
RTC_Alarm_IRQn	RTC Alarm interrupt
EXTI5_9_IRQn	EXTI line 5 to 9 interrupts
TIMER0_TRG_CMT_UP_BRK_IRQn	TIMER0 Trigger, Commutation, Update, Break interrupt
TIMER0_Channel_IRQn	TIMER0 capture compare interrupt
TIMER2_IRQn	TIMER2 interrupt
TIMER13_IRQn	TIMER13 interrupt
TIMER15_IRQn	TIMER15 interrupt
TIMER16_IRQn	TIMER16 interrupt
EXTI10_15_IRQn	EXTI line 10 to 15 interrupts
DMAMUX_IRQn	DMAMUX interrupt
CMP0_IRQn	Comparator 0 interrupt
CMP1_IRQn	Comparator 1 interrupt
I2C0_WKUP_IRQn	I2C0 Wakeup interrupt
I2C1_WKUP_IRQn	I2C1 Wakeup interrupt
USART0_WKUP_IRQn	USART0 Wakeup interrupt

### nvic\_irq\_enable

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

**Table 3-302. Function nvic\_irq\_enable**

<b>Function name</b>	nvic_irq_enable
<b>Function prototype</b>	void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority);
<b>Function descriptions</b>	enable NVIC interrupt request
<b>Precondition</b>	-
<b>The called functions</b>	NVIC_SetPriority、NVIC_EnableIRQ
<b>Input parameter{in}</b>	
nvic_irq	NVIC interrupt, refer to enum <a href="#">Table 3-301. IRQn Type</a>
<b>Input parameter{in}</b>	
nvic_irq_pre_priority	the pre-emption priority needed to set
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* enable window watchDog timer interrupt, priority is 1 */
nvic_irq_enable(WWDGT_IRQn, 1);
```

### **nvic\_irq\_disable**

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

**Table 3-303. Function nvic\_irq\_disable**

<b>Function name</b>	nvic_irq_disable
<b>Function prototype</b>	void nvic_irq_disable(uint8_t nvic_irq);
<b>Function descriptions</b>	disable NVIC interrupt request
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>nvic_irq</b>	NVIC interrupt, refer to enum <a href="#">Table 3-301. IRQn Type</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable window watchDog timer interrupt */
nvic_irq_disable(WWDGT_IRQn);
```

### **nvic\_vector\_table\_set**

The description of nvic\_vector\_table\_set is shown as below:

**Table 3-304. Function nvic\_vector\_table\_set**

<b>Function name</b>	nvic_vector_table_set
<b>Function prototype</b>	void nvic_vector_table_set(uint32_t nvic_vect_tab, uint32_t offset);
<b>Function descriptions</b>	set the NVIC vector table base address
<b>Precondition</b>	-
<b>The called functions</b>	-
<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
<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);
```

### nvic\_system\_reset

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

**Table 3-305. Function nvic\_system\_reset**

<b>Function name</b>	nvic_system_reset
<b>Function prototype</b>	void nvic_system_reset(void);
<b>Function descriptions</b>	initiates a system reset request to reset the MCU
<b>Precondition</b>	-
<b>The called functions</b>	NVIC_SystemReset
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset the MCU */
nvic_system_reset();
```

### system\_lowpower\_set

The description of system\_lowpower\_set is shown as below:

**Table 3-306. 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
<b>SCB_LPM_SLEEP_EXI</b>	if chose this para, the system always enter low power mode by exiting from

<i>T_ISR</i>	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_</i> <i>ALL_INT</i>	if chose this para, the lowpower mode can be woke up by all the enable and disable interrupts
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* the system always enter low power mode by exiting from ISR */
system_lowpower_set(SCB_LPM_SLEEP_EXIT_ISR);
```

### system\_lowpower\_reset

The description of system\_lowpower\_reset is shown as below:

**Table 3-307. Function system\_lowpower\_reset**

<b>Function name</b>	system_lowpower_reset
<b>Function prototype</b>	void system_lowpower_reset(uint8_t lowpower_mode);
<b>Function descriptions</b>	reset the state of the low power mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lowpower_mode</b>	the low power mode state
<i>SCB_LPM_SLEEP_EXIT_ISR</i>	if chose this para, the system will exit low power mode by exiting from ISR
<i>SCB_LPM_DEEPSLEEP</i> <i>P</i>	if chose this para, the system will enter the SLEEP mode
<i>SCB_LPM_WAKE_BY_</i> <i>ALL_INT</i>	if chose this para, the lowpower mode only can be woke up by the enable interrupts
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* the system will exit low power mode by exiting from ISR */
system_lowpower_reset(SCB_LPM_SLEEP_EXIT_ISR);
```

### systick\_clksource\_set

The description of systick\_clksource\_set is shown as below:



Table 3-308. Function `systick_clksource_set`

<b>Function name</b>	<code>systick_clksource_set</code>
<b>Function prototype</b>	<code>void systick_clksource_set(uint32_t systick_clksource);</code>
<b>Function descriptions</b>	set the systick clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>systick_clksource</b>	the systick clock source needed to choose
<code>SYSTICK_CLKSOURC E_HCLK</code>	systick clock source is from HCLK
<code>SYSTICK_CLKSOURC E_HCLK_DIV8</code>	systick clock source is from HCLK/8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* systick clock source is HCLK/8 */
```

```
systick_clksource_set(SYSTICK_CLKSOURCE_HCLK_DIV8);
```

## 3.13. PMU

According to the Power management unit (PMU), provides six types of power saving modes, including Run1, Sleep, Sleep1, Deep-sleep, Deep-sleep1 and Standby mode. The PMU registers are listed in chapter [3.13.1](#), the PMU firmware functions are introduced in chapter [3.13.2](#).

### 3.13.1. Descriptions of Peripheral registers

PMU registers are listed in the table shown as below:

Table 3-309. PMU Registers

Registers	Descriptions
PMU_CTL0	control 0 register
PMU_CS	control and status register
PMU_CTL1	control 1 register
PMU_STAT	status register
PMU_PAR	parameter register

### 3.13.2. Descriptions of Peripheral functions

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

Table 3-310. PMU firmware function

Function name	Function description
pmu_deinit	reset PMU registers
pmu_deepsleep_voltage_select	select deepsleep mode voltage
pmu_low_power_enable	enable low power LDO
pmu_low_power_disable	disable low power LDO
pmu_to_sleepmode	PMU work in Sleep mode
pmu_to_deepsleepmode	PMU work in Deep-sleep mode
pmu_to_standbymode	PMU work in standby mode
pmu_wakeup_pin_enable	enable PMU wakeup pin
pmu_wakeup_pin_disable	disable PMU wakeup pin
pmu_backup_write_enable	enable backup domain write
pmu_backup_write_disable	disable backup domain write
pmu_eflash_run_power_config	configure power state of eflash domain in run/run1 mode
pmu_eflash_deepsleep_power_config	configure power state of eflash domain when enter deepsleep/deepsleep1
pmu_eflash_wakeup_time_config	configure eflash wakeup time, using IRC48M counter
pmu_deepsleep_wait_time_config	configure IRC48M counter before enter Deepsleep/Deepsleep1 mode
pmu_flag_get	get PMU flag status
pmu_flag_clear	clear PMU flag status

### pmu\_deinit

The description of pmu\_deinit is shown as below:

Table 3-311. Function pmu\_deinit

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

Example:

```

/* reset PMU */
pmu_deinit();

```

## pmu\_deepsleep\_voltage\_select

The description of pmu\_deepsleep\_voltage\_select is shown as below:

**Table 3-312. pmu\_deepsleep\_voltage\_select**

<b>Function name</b>	pmu_deepsleep_voltage_select
<b>Function prototype</b>	void pmu_deepsleep_voltage_select(uint32_t dsv_n);
<b>Function descriptions</b>	select deepsleep mode voltage
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>deepsleepvol</b>	deepsleep voltage
PMU_DSV_0	0.9V (only valid when NPLDO is off)
PMU_DSV_1	1.0V
PMU_DSV_2	1.1V
PMU_DSV_3	1.2V
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select deepsleep mode voltage as 1.0V */
pmu_deepsleep_voltage_select(PMU_DSV_1);
```

## pmu\_low\_power\_ldo\_enable

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

**Table 3-313. Function pmu\_low\_power\_ldo\_enable**

<b>Function name</b>	pmu_low_power_ldo_enable
<b>Function prototype</b>	void pmu_low_power_ldo_enable(void);
<b>Function descriptions</b>	enable low power LDO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable low power LDO */
```

```
pmu_low_power_ldo_enable();
```

### pmu\_low\_power\_ldo\_disable

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

**Table 3-314. Function pmu\_low\_power\_ldo\_disable**

<b>Function name</b>	pmu_low_power_ldo_disable
<b>Function prototype</b>	void pmu_low_power_ldo_disable(void);
<b>Function descriptions</b>	disable low power LDO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable low power LDO */
pmu_low_power_ldo_disable();
```

### pmu\_to\_sleepmode

The description of pmu\_to\_sleepmode is shown as below:

**Table 3-315. Function pmu\_to\_sleepmode**

<b>Function name</b>	pmu_to_sleepmode
<b>Function prototype</b>	void pmu_to_sleepmode(uint8_t sleepmodecmd);
<b>Function descriptions</b>	PMU work in sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sleepmodecmd</b>	command to enter sleep mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work in sleep mode */
```

```
pmu_to_sleepmode(WFI_CMD);
```

## pmu\_to\_deepsleepmode

The description of pmu\_to\_deepsleepmode is shown as below:

**Table 3-316. Function pmu\_to\_deepsleepmode**

<b>Function name</b>	pmu_to_deepsleepmode
<b>Function prototype</b>	void pmu_to_deepsleepmode(uint8_t deepsleepmodecmd, uint8_t deepsleepmode);
<b>Function descriptions</b>	PMU work in Deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>deepsleepmodecmd</b>	Deep-sleep mode command
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
<b>Input parameter{in}</b>	
<b>deepsleepmode</b>	Deep-sleep mode
<i>PMU_DEEPSLEEP</i>	Deep-sleep mode
<i>PMU_DEEPSLEEP1</i>	Deep-sleep mode 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work in Deep-sleep mode */
```

```
pmu_to_deepsleepmode(WFI_CMD, PMU_DEEPSLEEP);
```

## pmu\_to\_standbymode

The description of pmu\_to\_standbymode is shown as below:

**Table 3-317. Function pmu\_to\_standbymode**

<b>Function name</b>	pmu_to_standbymode
<b>Function prototype</b>	void pmu_to_standbymode();
<b>Function descriptions</b>	pmu work in standby mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* PMU work in standby mode */
```

```
pmu_to_standby(WFI_CMD);
```

### pmu\_wakeup\_pin\_enable

The description of pmu\_wakeup\_pin\_enable is shown as below:

**Table 3-318. Function pmu\_wakeup\_pin\_enable**

Function name	pmu_wakeup_pin_enable
Function prototype	void pmu_wakeup_pin_enable(uint32_t wakeup_pin);
Function descriptions	enable wakeup pin
Precondition	-
The called functions	-
Input parameter{in}	
wakeup_pin	Wakeup pin
PMU_WAKEUP_PIN0	WKUP Pin 0 (PA0)
PMU_WAKEUP_PIN1	WKUP Pin 1 (PC13 / PA4)
PMU_WAKEUP_PIN2	WKUP Pin 2 (PB6)
PMU_WAKEUP_PIN3	WKUP Pin 3 (PA2)
PMU_WAKEUP_PIN5	WKUP Pin 5 (PB5)
Output parameter{out}	
-	-
Return value	
-	-

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 3-319. Function pmu\_wakeup\_pin\_disable**

Function name	pmu_wakeup_pin_disable
Function prototype	void pmu_wakeup_pin_disable(uint32_t wakeup_pin);
Function descriptions	disable wakeup pin
Precondition	-
The called functions	-

Input parameter{in}	
<b>wakeup_pin</b>	Wakeup pin
<i>PMU_WAKEUP_PIN0</i>	WKUP Pin 0 (PA0)
<i>PMU_WAKEUP_PIN1</i>	WKUP Pin 1 (PC13 / PA4)
<i>PMU_WAKEUP_PIN2</i>	WKUP Pin 2 (PB6)
<i>PMU_WAKEUP_PIN3</i>	WKUP Pin 3 (PA2)
<i>PMU_WAKEUP_PIN5</i>	WKUP Pin 5 (PB5)
Output parameter{out}	
-	-
Return value	
-	-

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 3-320. 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>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

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 3-321. 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\_eflash\_run\_power\_config

The description of pmu\_eflash\_run\_power\_config is shown as below:

**Table 3-322. Function pmu\_eflash\_run\_power\_config**

<b>Function name</b>	pmu_eflash_run_power_config
<b>Function prototype</b>	void pmu_eflash_run_power_config(ControlStatus state);
<b>Function descriptions</b>	configure power state of eflash domain in run/run1 mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>state</b>	power state
<i>ENABLE</i>	eflash domain power on
<i>DISABLE</i>	eflash domain power off
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure power state of EFLASH */
pmu_eflash_run_power_config(ENABLE);
```

### pmu\_eflash\_deepsleep\_power\_config

The description of pmu\_eflash\_deepsleep\_power\_config is shown as below:



Table 3-323. Function pmu\_eflash\_deepsleep\_power\_config

<b>Function name</b>	pmu_eflash_deepsleep_power_config
<b>Function prototype</b>	void pmu_eflash_deepsleep_power_config(ControlStatus state);
<b>Function descriptions</b>	configure power state of eflash domain when enter deepsleep/deepsleep1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>state</b>	power state
<i>ENABLE</i>	eflash domain power on
<i>DISABLE</i>	eflash domain power off
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure power state of EFLASH */
pmu_eflash_deepsleep_power_config(DISABLE);
```

### pmu\_eflash\_wakeup\_time\_config

The description of pmu\_eflash\_wakeup\_time\_config is shown as below:

Table 3-324. Function pmu\_eflash\_wakeup\_time\_config

<b>Function name</b>	pmu_eflash_wakeup_time_config
<b>Function prototype</b>	void pmu_eflash_wakeup_time_config(uint32_t wakeup_time);
<b>Function descriptions</b>	configure eflash wakeup time, using IRC48M counter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_time</b>	IRC48M counter for eflash wakeup time
<i>uint32_t</i>	0x0~0xFF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure IRC48M counter for eflash wakeup time */
pmu_eflash_wakeup_time_config(100);
```

## pmu\_deepsleep\_wait\_time\_config

The description of pmu\_deepsleep\_wait\_time\_config is shown as below:

**Table 3-325. Function pmu\_deepsleep\_wait\_time\_config**

<b>Function name</b>	pmu_deepsleep_wait_time_config
<b>Function prototype</b>	void pmu_deepsleep_wait_time_config(uint32_t wait_time);
<b>Function descriptions</b>	configure IRC48M counter before enter Deepsleep/Deepsleep1 mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_time</b>	wait time,using IRC48M counter
<i>uint32_t</i>	0x0~0x1F
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure IRC48M counter before enter Deepsleep/Deepsleep1 mode */
```

```
pmu_deepsleep_wait_time_config(20);
```

```
pmu_to_deepsleepmode(WFI_CMD, PMU_DEEPSLEEP);
```

## pmu\_flag\_get

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

**Table 3-326. 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 PMU flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flag
<i>PMU_FLAG_WAKEUP</i>	wakeup flag
<i>PMU_FLAG_STANDBY</i>	standby flag
<i>PMU_FLAG_LDOVSRF</i>	LDO voltage select ready flag
<i>PMU_FLAG_NPRDY</i>	normal-power LDO ready flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get flag state */

FlagStatus status;

status = pmu_flag_get(PMU_FLAG_WAKEUP);
```

### pmu\_flag\_clear

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

**Table 3-327. Function pmu\_flag\_clear**

<b>Function name</b>	pmu_flag_clear
<b>Function prototype</b>	void pmu_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear PMU flag status
<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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear flag bit */

pmu_flag_clear(PMU_FLAG_STANDBY);
```

## 3.14. 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 [3.14.1](#), the RCU firmware functions are introduced in chapter [3.14.2](#).

### 3.14.1. Descriptions of Peripheral registers

RCU registers are listed in the table shown as below:

**Table 3-328. RCU Registers**

Registers	Descriptions
RCU_CTL0	control 0 register

RCU_CFG0	configuration register 0
RCU_INT	interrupt register
RCU_AHB1RST	AHB1 reset register
RCU_AHB2RST	AHB1 reset register
RCU_APBRSRST	APB reset register
RCU_AHB1EN	AHB1 enable register
RCU_AHB2EN	AHB2 enable register
RCU_APBEN	APB enable register
RCU_AHB1SPDPEN	AHB1 sleep and deep sleep mode enable register
RCU_AHB2SPDPEN	AHB2 sleep and deep sleep mode enable register
RCU_APBSPDPEN	APB sleep and deep sleep mode enable register
RCU_CTL1	control 1 register
RCU_RSTSCK	reset source /clock register
RCU_CFG1	configuration register 1

### 3.14.2. Descriptions of Peripheral functions

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

**Table 3-329. RCU firmware function**

Function name	Function description
rcu_deinit	deinitialize the RCU
rcu_periph_clock_enable	enable the peripherals clock
rcu_periph_clock_disable	disable the peripherals clock
rcu_periph_clock_sleep_enable	enable the peripherals clock when sleep mode
rcu_periph_clock_sleep_disable	disable the peripherals clock when sleep mode
rcu_periph_reset_enable	reset the peripherals
rcu_periph_reset_disable	disable reset the peripheral
rcu_bkp_reset_enable	reset the BKP (for GD32C231 and GD32C221)
rcu_bkp_reset_disable	disable the BKP reset (for GD32C231 and GD32C221)
rcu_rtc_reset_enable	reset the RTC (for GD32C211)
rcu_rtc_reset_disable	disable the RTC reset (for GD32C211)
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_apb_clock_config	configure the APB clock prescaler selection
rcu_adc_clock_config	configure the ADC clock source and prescaler selection (for GD32C231 and GD32C221)
rcu_adc_clock_config	configure the ADC clock source and prescaler selection (for GD32C211)
rcu_ckout0_config	configure the CK_OUT0 clock source and divider
rcu_ckout1_config	configure the CK_OUT1 clock source and divider
rcu_lsckout_enable	enable the low speed clock output
rcu_lsckout_disable	disable the low speed clock output
rcu_lsckout_config	configure the LSCKOUT clock source
rcu_usart_clock_config	configure the USART clock source selection
rcu_i2c_clock_config	configure the I2Cx(x=0,1) clock source selection
rcu_i2s_clock_config	configure the I2S clock source selection
rcu_irc48mdiv_sys_clock_config	configure the IRC48MDIV_SYS clock selection
rcu_irc48mdiv_per_clock_config	configure the IRC48MDIV clock selection
rcu_rtc_clock_config	configure the RTC clock source selection
rcu_lxtal_drive_capability_config	configure the LXTAL drive capability
rcu_osci_stab_wait	wait until oscillator stabilization flags is SET
rcu_osci_on	turn on the oscillator
rcu_osci_off	turn off the oscillator
rcu_osci_start_enable	The startup circuit enable when osci is ready (for GD32C211)
rcu_osci_start_disable	The startup circuit disable when osci is ready (for GD32C211)
rcu_osci_bypass_mode_enable	enable the oscillator bypass mode, HXTALEN or LXTALEN must be reset before it
rcu_osci_bypass_mode_disable	disable the oscillator bypass mode, HXTALEN or LXTALEN must be reset before it
rcu_irc48m_adjust_value_set	set the IRC48M adjust value
rcu_lxtal_stab_reset_enable	enable LXTAL stabilization reset
rcu_lxtal_stab_reset_disable	disable LXTAL stabilization reset
rcu_hxtal_clock_monitor_enable	enable the HXTAL clock monitor
rcu_hxtal_clock_monitor_disable	disable the HXTAL clock monitor
rcu_lxtal_clock_monitor_enable	enable the LXTAL clock monitor
rcu_lxtal_clock_monitor_disable	disable the LXTAL clock monitor
rcu_clock_freq_get	get the system clock, bus and peripheral clock frequency

rcu_flag_get	get the clock stabilization and peripheral reset flags
rcu_all_reset_flag_clear	clear the reset flag
rcu_interrupt_enable	enable the stabilization interrupt
rcu_interrupt_disable	disable the stabilization interrupt
rcu_interrupt_flag_get	get the clock stabilization interrupt and ckm flags
rcu_interrupt_flag_clear	clear the interrupt flags

## Enum rcu\_periph\_enum

**Table 3-330. Enum rcu\_periph\_enum**

Member name	Descriptions
RCU_FMC	FMC clock
RCU_CRC	CRC clock
RCU_DMA	DMA clock
RCU_DMAMUX	DMAMUX clock
RCU_GPIOA	GPIOA clock
RCU_GPIOB	GPIOB clock
RCU_GPIOC	GPIOC clock
RCU_GPIOD	GPIOD clock
RCU_GPIOF	GPIOF clock
RCU_SYSCFG	SYSCFG clock
RCU_CMP	CMP clock
RCU_WWDGT	WWDGT clock
RCU_ADC	ADC clock
RCU_TIMER0	TIMER0 clock
RCU_TIMER2	TIMER2 clock
RCU_SPI0	SPI0 clock
RCU_SPI1	SPI1 clock
RCU_USART0	USART0 clock
RCU_USART1	USART1 clock
RCU_TIMER13	TIMER13 clock
RCU_TIMER15	TIMER15 clock
RCU_TIMER16	TIMER16 clock
RCU_USART2	USART2 clock
RCU_I2C0	I2C0 clock
RCU_I2C1	I2C1 clock
RCU_DBGMCU	DBGMCU clock
RCU_PMU	PMU clock
RCU_RTC	RTC clock

## Enum rcu\_periph\_sleep\_enum

Table 3-331. Enum rcu\_periph\_sleep\_enum

Member name	Descriptions
RCU_SRAM_SLP	SRAM clock
RCU_FMC_SLP	FMC clock
RCU_CRC_SLP	CRC clock
RCU_DMA_SLP	DMA clock
RCU_DMAMUX_SLP	DMAMUX clock
RCU_GPIOA_SLP	GPIOA clock
RCU_GPIOB_SLP	GPIOB clock
RCU_GPIOC_SLP	GPIOC clock
RCU_GPIOD_SLP	GPIOD clock
RCU_GPIOF_SLP	GPIOF clock
RCU_SYSCFG_SLP	SYSCFG clock
RCU_CMP_SLP	CMP clock
RCU_WWDGT_SLP	WWDGT clock
RCU_ADC_SLP	ADC clock
RCU_TIMER0_SLP	TIMER0 clock
RCU_TIMER2_SLP	TIMER2 clock
RCU_SPI0_SLP	SPI0 clock
RCU_SPI1_SLP	SPI1 clock
RCU_USART0_SLP	USART0 clock
RCU_USART1_SLP	USART1 clock
RCU_TIMER13_SLP	TIMER13 clock
RCU_TIMER15_SLP	TIMER15 clock
RCU_TIMER16_SLP	TIMER16 clock
RCU_USART2_SLP	USART2 clock
RCU_I2C0_SLP	I2C0 clock
RCU_I2C1_SLP	I2C1 clock
RCU_PMU_SLP	PMU clock

## Enum rcu\_periph\_reset\_enum

Table 3-332. Enum rcu\_periph\_reset\_enum

Member name	Descriptions
RCU_CRCCRST	CRC clock
RCU_DMARST	DMA clock
RCU_DMAMUXRST	DMA clock
RCU_GPIOARST	GPIOA clock
RCU_GPIOBRST	GPIOB clock
RCU_GPIOCRST	GPIOC clock

RCU_GPIODRST	GPIOD clock
RCU_GPIOFRST	GPIOF clock
RCU_SYSCFGRST	SYSCFG clock
RCU_CMPRST	CMP clock
RCU_WWDGTRST	WWDGT clock
RCU_ADCRST	ADC clock
RCU_TIMER0RST	TIMER0 clock
RCU_TIMER2RST	TIMER2 clock
RCU_SPI0RST	SPI0 clock
RCU_SPI1RST	SPI1 clock
RCU_USART0RST	USART0 clock
RCU_USART1RST	USART1 clock
RCU_TIMER13RST	TIMER13 clock
RCU_TIMER15RST	TIMER15 clock
RCU_TIMER16RST	TIMER16 clock
RCU_USART2RST	USART2 clock
RCU_I2C0RST	I2C0 clock
RCU_I2C1RST	I2C1 clock
RCU_PMURST	PMU clock

## Enum rcu\_flag\_enum

**Table 3-333. Enum rcu\_flag\_enum**

Member name	Descriptions
RCU_FLAG_IRC32KST B	IRC32K stabilization flag
RCU_FLAG_LXTALST B	LXTAL stabilization flag
RCU_FLAG_IRC48MS TB	IRC48M stabilization flag
RCU_FLAG_HXTALST B	HXTAL stabilization flag
RCU_FLAG_OBLRST	option byte loader reset flag
RCU_FLAG_BORRST	BOR reset flag
RCU_FLAG_EPRST	External PIN reset flag
RCU_FLAG_PORRST	power reset flag
RCU_FLAG_SWRST	SW reset flag
RCU_FLAG_FWDGTR ST	FWDGT reset flag
RCU_FLAG_WWDGTR ST	WWDGT reset flag
RCU_FLAG_LPRST	low-power reset flag



RCU_FLAG_LCKCMD	LXTAL clock failure detection flag
-----------------	------------------------------------

## Enum rcu\_int\_flag\_enum

Table 3-334. Enum rcu\_int\_flag\_enum

Member name	Descriptions
RCU_INT_FLAG_IRC3 2KSTB	IRC32K stabilization interrupt flag
RCU_INT_FLAG_LXTA LSTB	LXTAL stabilization interrupt flag
RCU_INT_FLAG_IRC4 8MSTB	IRC48M stabilization interrupt flag
RCU_INT_FLAG_LXTA LCKM	LXTAL clock stuck interrupt flag
RCU_INT_FLAG_HXTA LSTB	HXTAL stabilization interrupt flag
RCU_INT_FLAG_CKM	CKM interrupt flag

## Enum rcu\_int\_flag\_clear\_enum

Table 3-335. Enum rcu\_int\_flag\_clear\_enum

Member name	Descriptions
RCU_INT_FLAG_IRC3 2KSTB_CLR	IRC32K stabilization interrupt flags clear
RCU_INT_FLAG_LXTA LSTB_CLR	LXTAL stabilization interrupt flags clear
RCU_INT_FLAG_IRC4 8MSTB_CLR	IRC48M stabilization interrupt flags clear
RCU_INT_FLAG_LXTA LCKM_CLR	LXTAL clock stuck interrupt flag clear
RCU_INT_FLAG_HXTA LSTB_CLR	HXTAL stabilization interrupt flags clear
RCU_INT_FLAG_CKM _CLR	CKM interrupt flags clear

## Enum rcu\_int\_enum

Table 3-336. Enum rcu\_int\_enum

Member name	Descriptions
RCU_INT_IRC32KSTB	IRC32K stabilization interrupt
RCU_INT_LXTALSTB	LXTAL stabilization interrupt
RCU_INT_IRC48MSTB	IRC48M stabilization interrupt

RCU\_INT\_HXTALSTB

HXTAL stabilization interrupt

## Enum rcu\_osci\_type\_enum

Table 3-337. Enum rcu\_osci\_type\_enum

Member name	Descriptions
RCU_HXTAL	HXTAL
RCU_LXTAL	LXTAL
RCU_IRC48M	IRC48M
RCU_IRC32K	IRC32K

## Enum rcu\_clock\_freq\_enum

Table 3-338. Enum rcu\_clock\_freq\_enum

Member name	Descriptions
CK_SYS	system clock

## Enum usart\_idx\_enum

Table 3-339. Enum usart\_idx\_enum

Member name	Descriptions
IDX_USART0	index of USART0
IDX_USART1	index of USART1

## Enum i2c\_idx\_enum

Table 3-340. Enum i2c\_idx\_enum

Member name	Descriptions
IDX_I2C0	index of I2C0

## rcu\_deinit

The description of rcu\_deinit is shown as below:

Table 3-341. Function rcu\_deinit

Function name	rcu_deinit
Function prototype	void rcu_deinit(void)
Function descriptions	deinitialize the RCU
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize the RCU */
```

```
rcu_deinit();
```

## rcu\_periph\_clock\_enable

The description of rcu\_periph\_clock\_enable is shown as below:

**Table 3-342. Function rcu\_periph\_clock\_enable**

Function name	rcu_periph_clock_enable
Function prototype	void rcu_periph_clock_enable(rcu_periph_enum periph)
Function descriptions	enable the peripherals clock
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to <a href="#">Table 3-330. Enum rcu_periph_enum</a>
RCU_FMC	FMC clock
RCU_CRC	CRC clock
RCU_DMA	DMA clock
RCU_DMAMUX	DMAMUX clock
RCU_GPIOx (x=A,B,C,D,F)	GPIO ports clock
RCU_SYSCFG	SYSCFG clock
RCU_CMP	CMP clock
RCU_WWDGT	WWDGT clock
RCU_ADC	ADC clock
RCU_TIMERx (x=0,2,13,15,16)	TIMER clock
RCU_SPIx (x=0,1)	SPI clock
RCU_USARTx (x=0,1,2)	USART clock
RCU_I2Cx (x=0,1)	I2C clock
RCU_DBGMCU	DBGMCU clock
RCU_PMU	PMU clock
RCU_RTC	RTC clock
Output parameter{out}	
-	-
Return value	

Example:

```
/* CMP peripherals clock enable*/

rcu_periph_clock_enable(RCU_CMP);
```

## rcu\_periph\_clock\_disable

The description of rcu\_periph\_clock\_disable is shown as below:

**Table 3-343. 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 3-330. Enum rcu_periph_enum</a>
RCU_CRC	CRC clock
RCU_DMA	DMA clock
RCU_DMAMUX	DMAMUX clock
RCU_GPIOx (x=A,B,C,D,F)	GPIO ports clock
RCU_SYSCFG	SYSCFG clock
RCU_CMP	CMP clock
RCU_WWDGT	WWDGT clock
RCU_ADC	ADC clock
RCU_TIMERx (x=0,1,2,5,13,15,16)	TIMER clock
RCU_SPIx (x=0,1)	SPI clock
RCU_USARTx (x=0,1,2)	USART clock
RCU_I2Cx (x=0,1)	I2C clock
RCU_DBGMCU	DBGMCU clock
RCU_PMU	PMU clock
RCU_RTC	RTC clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CMP peripherals clock disable*/
```

```
rcu_periph_clock_disable(RCU_CMP);
```

## rcu\_periph\_clock\_sleep\_enable

The description of rcu\_periph\_clock\_sleep\_enable is shown as below:

**Table 3-344. 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 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 3-331. Enum rcu_periph_sleep_enum</a>
<i>RCU_SRAM_SLP</i>	SRAM clock
<i>RCU_FMC_SLP</i>	FMC clock
<i>RCU_CRC_SLP</i>	CRC clock
<i>RCU_DMA_SLP</i>	DMA clock
<i>RCU_DMAMUX_SLP</i>	DMAMUX clock
<i>RCU_GPIOx_SLP</i> (x = A,B,C,D,F)	GPIO ports clock
<i>RCU_SYSCFG_SLP</i>	SYSCFG clock
<i>RCU_CMP_SLP</i>	CMP clock
<i>RCU_WWDGT_SLP</i>	WWDGT clock
<i>RCU_ADC_SLP</i>	ADC clock
<i>RCU_TIMERx_SLP</i> (x=0,2,13,15,16)	TIMER clock
<i>RCU_SPIx_SLP</i> (x = 0,1)	SPI clock
<i>RCU_USARTx_SLP</i> (x = 0,1,2)	USART clock
<i>RCU_I2Cx_SLP</i> (x = 0,1)	I2C clock
<i>RCU_PMU_SLP</i>	PMU clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* CMP peripherals clock enable when sleep mode */
rcu_periph_clock_sleep_enable(RCU_CMP_SLP);
```

## rcu\_periph\_clock\_sleep\_disable

The description of rcu\_periph\_clock\_sleep\_disable is shown as below:

**Table 3-345. 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 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 3-331. Enum rcu_periph_sleep_enum</a>
<i>RCU_SRAM_SLP</i>	SRAM clock
<i>RCU_FMC_SLP</i>	FMC clock
<i>RCU_CRC_SLP</i>	CRC clock
<i>RCU_DMA_SLP</i>	DMA clock
<i>RCU_DMAMUX_SLP</i>	DMAMUX clock
<i>RCU_GPIOx_SLP</i> (x = A,B,C,D,F)	GPIO ports clock
<i>RCU_SYSCFG_SLP</i>	SYSCFG clock
<i>RCU_CMP_SLP</i>	CMP clock
<i>RCU_WWDGT_SLP</i>	WWDGT clock
<i>RCU_ADC_SLP</i>	ADC clock
<i>RCU_TIMERx_SLP</i> (x=0,2,13,15,16)	TIMER clock
<i>RCU_SPIx_SLP</i> (x = 0,1)	SPI clock
<i>RCU_USARTx_SLP</i> (x = 0,1,2)	USART clock
<i>RCU_I2Cx_SLP</i> (x = 0,1)	I2C clock
<i>RCU_PMU_SLP</i>	PMU clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* CMP peripherals clock disable when sleep mode */
rcu_periph_clock_sleep_disable(RCU_CMP_SLP);
```

## rcu\_periph\_reset\_enable

The description of rcu\_periph\_reset\_enable is shown as below:

**Table 3-346. 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>	reset the peripherals
<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 3-332. Enum rcu_periph_reset_enum</a>
RCU_CRCRST	reset CRC
RCU_DMARST	reset DMA
RCU_DMAMUXRST	reset DMAMUX
RCU_GPIOxRST (x = A,B,C,D,F)	reset GPIO ports
RCU_SYSCFGRST	reset SYSCFG
RCU_CMPRST	reset CMP
RCU_WWDGTRST	reset WWDGT
RCU_ADCRST	reset ADC
RCU_TIMERxRST (x=0,2,13,15,16)	reset TIMER
RCU_SPIxRST (x = 0,1)	reset SPI
RCU_USARTxRST (x = 0,1,2)	reset USART
RCU_I2CxRST (x = 0,1)	reset I2C
RCU_PMURST	reset PMU
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* CMP peripherals reset enable */
rcu_periph_reset_enable(RCU_CMPRST);
```

## rcu\_periph\_reset\_disable

The description of rcu\_periph\_reset\_disable is shown as below:

Table 3-347. Function rcu\_periph\_reset\_disable

Function name	rcu_periph_reset_disable
Function prototype	void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset)
Function descriptions	disable reset the peripheral
Precondition	-
The called functions	-
Input parameter{in}	
periph_reset	RCU peripherals reset, refer to <a href="#">Table 3-332. Enum rcu_periph_reset_enum</a>
RCU_CRCRST	reset CRC
RCU_DMARST	reset DMA
RCU_DMAMUXRST	reset DMAMUX
RCU_GPIOxRST (x = A,B,C,D,F)	reset GPIO ports
RCU_SYSCFGRST	reset SYSCFG
RCU_CMPRST	reset CMP
RCU_WWDGTRST	reset WWDGT
RCU_ADCRST	reset ADC
RCU_TIMERxRST (x=0,2,13,15,16)	reset TIMER
RCU_SPIxRST (x = 0,1)	reset SPI
RCU_USARTxRST (x = 0,1,2)	reset USART
RCU_I2CxRST (x = 0,1)	reset I2C
RCU_PMURST	reset PMU
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CMP peripherals reset disable */
rcu_periph_reset_disable(RCU_CMPRST);
```

## rcu\_bkp\_reset\_enable (for GD32C231 and GD32C221)

The description of rcu\_bkp\_reset\_enable is shown as below:

Table 3-348. Function rcu\_bkp\_reset\_enable

Function name	rcu_bkp_reset_enable
Function prototype	void rcu_bkp_reset_enable(void)
Function descriptions	reset the BKP



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

Example:

```
/* reset the BKP domain */
rcu_bkp_reset_enable();
```

## rcu\_bkp\_reset\_disable (for GD32C231 and GD32C221)

The description of rcu\_bkp\_reset\_disable is shown as below:

**Table 3-349. 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 reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the BKP domain reset */
rcu_bkp_reset_disable();
```

## rcu\_rtc\_reset\_enable (for GD32C211)

The description of rcu\_rtc\_reset\_enable is shown as below:

**Table 3-350. Function rcu\_rtc\_reset\_enable**

<b>Function name</b>	rcu_rtc_reset_enable
<b>Function prototype</b>	void rcu_rtc_reset_enable(void)
<b>Function descriptions</b>	reset the RTC
<b>Precondition</b>	-

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

Example:

```
/* reset RTC enable */
rcu_rtc_reset_enable();
```

## rcu\_rtc\_reset\_disable (for GD32C211)

The description of rcu\_rtc\_reset\_disable is shown as below:

**Table 3-351. Function rcu\_rtc\_reset\_disable**

Function name	rcu_rtc_reset_disable
Function prototype	void rcu_rtc_reset_disable(void)
Function descriptions	disable the RTC reset
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset RTC disable */
rcu_rtc_reset_disable();
```

## rcu\_system\_clock\_source\_config

The description of rcu\_system\_clock\_source\_config is shown as below:

**Table 3-352. Function rcu\_system\_clock\_source\_config**

Function name	rcu_system_clock_source_config
Function prototype	void rcu_system_clock_source_config(uint32_t ck_sys)
Function descriptions	configure the system clock source
Precondition	-
The called functions	-

Input parameter{in}	
<b>ck_sys</b>	system clock source select
<i>RCU_CKSYSSRC_IRC48MDIV_SYS</i>	select IRC48MDIV_SYS as the CK_SYS source
<i>RCU_CKSYSSRC_HXTAL</i>	select CK_HXTAL as the CK_SYS source
<i>RCU_CKSYSSRC_IRC32K</i>	select IRC32K as the CK_SYS source
<i>RCU_CKSYSSRC_LXTAL</i>	select LXTLA as the CK_SYS source
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select 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 3-353. 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>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
<b>uint32_t</b>	Description
<i>RCU_CKSYSSRC_IRC48MDIV_SYS</i>	select IRC48MDIV_SYS as the CK_SYS source
<i>RCU_CKSYSSRC_HXTAL</i>	select CK_HXTAL as the CK_SYS source
<i>RCU_CKSYSSRC_IRC32K</i>	select IRC32K as the CK_SYS source
<i>RCU_CKSYSSRC_LXTAL</i>	select LXTLA as the CK_SYS source

Example:

```
uint32_t temp_cksys_status;

/* get the CK_SYS source */

temp_cksys_status = rcu_system_clock_source_get();
```

## rcu\_ahb\_clock\_config

The description of rcu\_ahb\_clock\_config is shown as below:

**Table 3-354. 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, x=1, 2, 4, 8, 16, 64, 128, 256, 512</i>	select CK_SYS/x as CK_AHB
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure CK_SYS / 128 */

rcu_ahb_clock_config(RCU_AHB_CKSYS_DIV128);
```

## rcu\_apb\_clock\_config

The description of rcu\_apb\_clock\_config is shown as below:

**Table 3-355. Function rcu\_apb\_clock\_config**

<b>Function name</b>	rcu_apb_clock_config
<b>Function prototype</b>	void rcu_apb_clock_config(uint32_t ck_apb)
<b>Function descriptions</b>	configure the APB clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_apb</b>	APB clock prescaler selection
<i>RCU_APB_CKAHB_DI</i>	select CK_AHB as CK_APB

V1	
RCU_APB_CKAHB_DIV2	select CK_AHB/2 as CK_APB
RCU_APB_CKAHB_DIV4	select CK_AHB/4 as CK_APB
RCU_APB_CKAHB_DIV8	select CK_AHB/8 as CK_APB
RCU_APB_CKAHB_DIV16	select CK_AHB/16 as CK_APB
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure CK_AHB / 16 as CK_APB1 */
```

```
rcu_apb1_clock_config(RCU_APB1_CKAHB_DIV16);
```

## rcu\_adc\_clock\_config (for GD32C231 and GD32C221)

The description of rcu\_adc\_clock\_config is shown as below:

**Table 3-356. Function rcu\_adc\_clock\_config**

<b>Function name</b>	rcu_adc_clock_config
<b>Function prototype</b>	void rcu_adc_clock_config(uint32_t adc_clock_source, uint32_t ck_adc)
<b>Function descriptions</b>	configure the ADC clock source and prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>adc_clock_source</b>	ADC clock source selection
RCU_ADCSRC_CKSYS	ADC clock source select CK_SYS
RCU_ADCSRC_IRC48M_PER	ADC clock source select CK_IRC48MDIV_PER
Input parameter{in}	
<b>ck_adc</b>	ADC clock prescaler selection
RCU_ADCCCK_DIV1	ADC clock prescaler select not divided
RCU_ADCCCK_DIV2	ADC clock prescaler select divided by 2
RCU_ADCCCK_DIV4	ADC clock prescaler select divided by 4
RCU_ADCCCK_DIV6	ADC clock prescaler select divided by 6
RCU_ADCCCK_DIV8	ADC clock prescaler select divided by 8
RCU_ADCCCK_DIV10	ADC clock prescaler select divided by 10
RCU_ADCCCK_DIV12	ADC clock prescaler select divided by 12

<i>RCU_ADCCCK_DIV16</i>	ADC clock prescaler select divided by 16
<i>RCU_ADCCCK_DIV32</i>	ADC clock prescaler select divided by 32
<i>RCU_ADCCCK_DIV64</i>	ADC clock prescaler select divided by 64
<i>RCU_ADCCCK_DIV128</i>	ADC clock prescaler select divided by 128
<i>RCU_ADCCCK_DIV256</i>	ADC clock prescaler select divided by 256
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the ADC clock */
```

```
rcu_adc_clock_config(RCU_ADCSRC_CKSYS, RCU_ADCCCK_DIV8);
```

## rcu\_adc\_clock\_config (for GD32C211)

The description of rcu\_adc\_clock\_config is shown as below:

**Table 3-357. Function rcu\_adc\_clock\_config**

<b>Function name</b>	rcu_adc_clock_config
<b>Function prototype</b>	void rcu_adc_clock_config(uint32_t adc_clock_source, uint32_t ck_adc)
<b>Function descriptions</b>	configure the ADC clock source and prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_clock_source</b>	ADC clock source selection
<i>RCU_ADCSRC_CKSYS</i>	ADC clock source select CK_SYS
<i>RCU_ADCSRC_IRC48M_PER</i>	ADC clock source select CK_IRC48MDIV_PER
<i>RCU_ADCSRC_HXTAL</i>	ADC clock source select CK_HXTAL
<b>Input parameter{in}</b>	
<b>ck_adc</b>	ADC clock prescaler selection
<i>RCU_ADCCCK_DIV2</i>	ADC clock prescaler select divided by 2
<i>RCU_ADCCCK_DIV3</i>	ADC clock prescaler select divided by 3
<i>RCU_ADCCCK_DIV4</i>	ADC clock prescaler select divided by 4
<i>RCU_ADCCCK_DIV5</i>	ADC clock prescaler select divided by 5
<i>RCU_ADCCCK_DIV6</i>	ADC clock prescaler select divided by 6
<i>RCU_ADCCCK_DIV7</i>	ADC clock prescaler select divided by 7
<i>RCU_ADCCCK_DIV8</i>	ADC clock prescaler select divided by 8
<i>RCU_ADCCCK_DIV9</i>	ADC clock prescaler select divided by 9
<i>RCU_ADCCCK_DIV10</i>	ADC clock prescaler select divided by 10
<i>RCU_ADCCCK_DIV11</i>	ADC clock prescaler select divided by 11

<i>RCU_ADCCCK_DIV12</i>	ADC clock prescaler select divided by 12
<i>RCU_ADCCCK_DIV13</i>	ADC clock prescaler select divided by 13
<i>RCU_ADCCCK_DIV14</i>	ADC clock prescaler select divided by 14
<i>RCU_ADCCCK_DIV15</i>	ADC clock prescaler select divided by 15
<i>RCU_ADCCCK_DIV16</i>	ADC clock prescaler select divided by 16
<i>RCU_ADCCCK_DIV17</i>	ADC clock prescaler select divided by 17
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the ADC clock */
```

```
rcu_adc_clock_config(RCU_ADCSRC_CKSYS, RCU_ADCCCK_DIV8);
```

## rcu\_ckout0\_config

The description of rcu\_ckout0\_config is shown as below:

**Table 3-358. 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 divider
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ckout_src</b>	CK_OUT0 clock source selection
<i>RCU_CKOUT0SRC_NONE</i>	no clock selected
<i>RCU_CKOUT0SRC_CKSYS</i>	CK_OUT0 clock source select CKSYS
<i>RCU_CKOUT0SRC_IRC48M</i>	CK_OUT0 clock source select IRC48M
<i>RCU_CKOUT0SRC_HXTAL</i>	CK_OUT0 clock source select HXTAL
<i>RCU_CKOUT0SRC_IRC32K</i>	CK_OUT0 clock source select IRC32K
<i>RCU_CKOUT0SRC_LXTAL</i>	CK_OUT0 clock source select LXTAL
<b>Input parameter{in}</b>	
<b>ckout_div</b>	CK_OUT0 divider
<i>RCU_CKOUT0_DIVx(x = 1, 2, 4, 8, 16, 32, 64, 128)</i>	CK_OUT is divided by x

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the HXTAL as CK_OUT0 clock source */
rcu_ckout0_config(RCU_CKOUT0SRC_HXTAL, RCU_CKOUT0_DIV1);
```

## rcu\_ckout1\_config

The description of rcu\_ckout1\_config is shown as below:

**Table 3-359. Function rcu\_ckout1\_config**

Function name	rcu_ckout1_config
Function prototype	void rcu_ckout1_config(uint32_t ckout1_src, uint32_t ckout1_div)
Function descriptions	configure the CK_OUT clock source and divider
Precondition	-
The called functions	-
Input parameter{in}	
ckout_src	CK_OUT clock source selection
RCU_CKOUT1SRC_NONE	no clock selected
RCU_CKOUT1SRC_CKSYS	CK_OUT1 clock source select CKSYS
RCU_CKOUT1SRC_IRC48M	CK_OUT1 clock source select IRC48M
RCU_CKOUT1SRC_HXTAL	CK_OUT1 clock source select HXTAL
RCU_CKOUT1SRC_IRC32K	CK_OUT1 clock source select IRC32K
RCU_CKOUT1SRC_LXTAL	CK_OUT1 clock source select LXTAL
Input parameter{in}	
ckout_div	CK_OUT divider
RCU_CKOUT1_DIVx(x = 1, 2, 4, 8, 16, 32, 64, 128)	CK_OUT is divided by x
Output parameter{out}	
-	-
Return value	
-	-

Example:



```
/* configure the HXTAL as CK_OUT1 clock source */
```

```
rcu_ckout1_config(RCU_CKOUT1SRC_HXTAL, RCU_CKOUT1_DIV1);
```

## rcu\_lsckout\_enable

The description of rcu\_lsckout\_enable is shown as below:

**Table 3-360. Function rcu\_lsckout\_enable**

<b>Function name</b>	rcu_lsckout_enable
<b>Function prototype</b>	void rcu_lsckout_enable(void)
<b>Function descriptions</b>	enable the low speed clock output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the low speed clock output */
```

```
rcu_lsckout_enable ();
```

## rcu\_lsckout\_disable

The description of rcu\_lsckout\_disable is shown as below:

**Table 3-361. Function rcu\_lsckout\_disable**

<b>Function name</b>	rcu_lsckout_disable
<b>Function prototype</b>	void rcu_lsckout_disable(void)
<b>Function descriptions</b>	disable the low speed clock output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the low speed clock output */
```

```
rcu_lsckout_disable ();
```

## rcu\_lsckout\_config

The description of rcu\_lsckout\_config is shown as below:

**Table 3-362. Function rcu\_lsckout\_config**

<b>Function name</b>	rcu_lsckout_config
<b>Function prototype</b>	void rcu_lsckout_config(uint32_t lsckout_src)
<b>Function descriptions</b>	configure the LCKOUT clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lsckout_src</b>	LCKOUT clock source selection
<i>RCU_LCKOUTSRC_IRC32K</i>	IRC32K clock selected
<i>RCU_LCKOUTSRC_LXTAL</i>	LXTAL selected
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure IRC32K as LCKOUT clock source */
```

```
rcu_lsckout_config (RCU_LCKOUTSRC_IRC32K);
```

## rcu\_usart\_clock\_config

The description of rcu\_usart\_clock\_config is shown as below:

**Table 3-363. Function rcu\_usart\_clock\_config**

<b>Function name</b>	rcu_usart_clock_config
<b>Function prototype</b>	void rcu_usart_clock_config(usart_idx_enum usart_idx, 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>usart_idx</b>	IDX_USART0
<b>Input parameter{in}</b>	
<b>ck_usart</b>	USART clock source selection
<i>RCU_USART0SRC_CK_APB</i>	CK_USART select CK_APB

<i>RCU_USART0SRC_CKSYS</i>	CK_USART select CK_SYS
<i>RCU_USART0SRC_IRC48MDIV_PER</i>	CK_USART select CK_IRC48MDIV_PER
<i>RCU_USART0SRC_LXTAL</i>	CK_USART select LXTAL
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the LXTAL as USART0 clock */
```

```
rcu_usart_clock_config(IDX_USART0,RCU_USART0SRC_LXTAL);
```

## rcu\_i2c\_clock\_config

The description of rcu\_i2c\_clock\_config is shown as below:

**Table 3-364. Function rcu\_i2c\_clock\_config**

<b>Function name</b>	rcu_i2c_clock_config
<b>Function prototype</b>	void rcu_i2c_clock_config(i2c_idx_enum i2c_idx, uint32_t ck_i2c)
<b>Function descriptions</b>	configure the I2Cx(x=0,1) clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_idx</b>	IDX_I2Cx(x=0,1)
<b>Input parameter{in}</b>	
<b>ck_i2c</b>	I2C clock source selection
<i>RCU_I2CSRC_CKAPB</i>	CK_I2C select CK_APB
<i>RCU_I2CSRC_CKSYS</i>	CK_I2C select CK_SYS
<i>RCU_I2CSRC_IRC48MDIV_PER</i>	CK_I2C select CK_IRC48MDIV_PER
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the CKAPB as I2C0 clock */
```

```
rcu_usart_clock_config(IDX_I2C0, RCU_I2CSRC_CKAPB);
```

## rcu\_i2s\_clock\_config

The description of rcu\_i2s\_clock\_config is shown as below:

**Table 3-365. Function rcu\_i2s\_clock\_config**

<b>Function name</b>	rcu_i2s_clock_config
<b>Function prototype</b>	void rcu_i2s_clock_config(uint32_t ck_i2s)
<b>Function descriptions</b>	configure the I2S clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_i2s</b>	I2S clock source selection
<i>RCU_I2SSRC_CKSYS</i>	CK_I2S select CK_SYS
<i>RCU_I2SSRC_IRC48M DIV_PER</i>	CK_I2S select IRC48MDIV_PER
<i>RCU_I2SSRC_CKIN</i>	CK_I2S select I2S_CKIN
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the CKSYS as I2S clock */
rcu_i2s_clock_config (RCU_I2SSRC_CKSYS);
```

## rcu\_irc48mdiv\_sys\_clock\_config

The description of rcu\_irc48mdiv\_sys\_clock\_config is shown as below:

**Table 3-366. Function rcu\_irc48mdiv\_sys\_clock\_config**

<b>Function name</b>	rcu_irc48mdiv_sys_clock_config
<b>Function prototype</b>	void rcu_irc48mdiv_sys_clock_config(uint32_t ck_irc48mdiv_sys)
<b>Function descriptions</b>	configure the IRC48MDIV_SYS clock selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_irc48mdiv_sys</b>	IRC48MDIV clock selection
<i>RCU_IRC48MDIV_SYS _1</i>	CK_IRC48MDIV_SYS select CK_IRC48M
<i>RCU_IRC48MDIV_SYS _2</i>	CK_IRC48MDIV_SYS select CK_IRC48M divided by 2
<i>RCU_IRC48MDIV_SYS _4</i>	CK_IRC48MDIV_SYS select CK_IRC48M divided by 4

<i>RCU_IRC48MDIV_SYS</i> _8	CK_IRC48MDIV_SYS select CK_IRC48M divided by 8
<i>RCU_IRC48MDIV_SYS</i> _16	CK_IRC48MDIV_SYS select CK_IRC48M divided by 16
<i>RCU_IRC48MDIV_SYS</i> _32	CK_IRC48MDIV_SYS select CK_IRC48M divided by 32
<i>RCU_IRC48MDIV_SYS</i> _64	CK_IRC48MDIV_SYS select CK_IRC48M divided by 64
<i>RCU_IRC48MDIV_SYS</i> _128	CK_IRC48MDIV_SYS select CK_IRC48M divided by 128
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the IRC48MDIV_SYS clock selection */
```

```
rcu_irc48mdiv_sys_clock_config (RCU_IRC48MDIV_SYS_2);
```

## rcu\_irc48mdiv\_per\_clock\_config

The description of rcu\_irc48mdiv\_per\_clock\_config is shown as below:

**Table 3-367. Function rcu\_irc48mdiv\_per\_clock\_config**

<b>Function name</b>	rcu_irc48mdiv_per_clock_config
<b>Function prototype</b>	void rcu_irc48mdiv_per_clock_config(uint32_t ck_irc48mdiv_per)
<b>Function descriptions</b>	configure the IRC48MDIV clock selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_irc48mdiv_per</b>	IRC48MDIV clock selection
<i>RCU_IRC48MDIV_PER</i> _1	CK_IRC48MDIV_PER select CK_IRC48M
<i>RCU_IRC48MDIV_PER</i> _2	CK_IRC48MDIV_PER select CK_IRC48M divided by 2
<i>RCU_IRC48MDIV_PER</i> _3	CK_IRC48MDIV_PER select CK_IRC48M divided by 3
<i>RCU_IRC48MDIV_PER</i> _4	CK_IRC48MDIV_PER select CK_IRC48M divided by 4
<i>RCU_IRC48MDIV_PER</i> _5	CK_IRC48MDIV_PER select CK_IRC48M divided by 5
<i>RCU_IRC48MDIV_PER</i> _6	CK_IRC48MDIV_PER select CK_IRC48M divided by 6

<i>RCU_IRC48MDIV_PER</i> _7	CK_IRC48MDIV_PER select CK_IRC48M divided by 7
<i>RCU_IRC48MDIV_PER</i> _8	CK_IRC48MDIV_PER select CK_IRC48M divided by 8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the IRC48MDIV clock selection */
```

```
rcu_irc48mdiv_per_clock_config(RCU_IRC48MDIV_PER_8);
```

## rcu\_rtc\_clock\_config

The description of rcu\_rtc\_clock\_config is shown as below:

**Table 3-368. 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>	CK_LXTAL selected as RTC source clock
<i>RCU_RTCSRC_IRC32</i> <i>K</i>	CK_IRC32K selected as RTC source clock
<i>RCU_RTCSRC_HXTAL</i> _DIV32	CK_HXTAL/32 selected 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_IRC32K);
```

## rcu\_lxtal\_drive\_capability\_config

The description of rcu\_lxtal\_drive\_capability\_config is shown as below:

**Table 3-369. Function rcu\_lxtal\_drive\_capability\_config**

<b>Function name</b>	rcu_lxtal_drive_capability_config
<b>Function prototype</b>	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap)
<b>Function descriptions</b>	configure the LXTAL drive capability
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lxtal_dricap</b>	drive capability of LXTAL
<i>RCU_LXTAL_LOWDRI</i>	lower driving capability
<i>RCU_LXTAL_HIGHDRI</i>	higher driving capability
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the LXTAL lower driving capability */
```

```
rcu_lxtal_drive_capability_config(RCU_LXTAL_LOWDRI);
```

## rcu\_osci\_stab\_wait

The description of rcu\_osci\_stab\_wait is shown as below:

**Table 3-370. 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 until oscillator stabilization flags is SET
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-337. Enum rcu_osci_type_enum</a>
<i>RCU_HXTAL</i>	HXTAL
<i>RCU_LXTAL</i>	LXTAL
<i>RCU_IRC48M</i>	IRC48M
<i>RCU_IRC32K</i>	IRC32K
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* wait for oscillator stabilization flag */

if(SUCCESS == rcu_osc_stab_wait(RCU_HXTAL)){

}
```

## rcu\_osc\_on

The description of rcu\_osc\_on is shown as below:

**Table 3-371. Function rcu\_osc\_on**

<b>Function name</b>	rcu_osc_on
<b>Function prototype</b>	void rcu_osc_on(rcu_osc_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 3-337. Enum rcu_osc_type_enum</a>
RCU_HXTAL	HXTAL
RCU_LXTAL	LXTAL
RCU_IRC48M	IRC48M
RCU_IRC32K	IRC32K
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn on the high speed crystal oscillator */

rcu_osc_on(RCU_HXTAL);
```

## rcu\_osc\_off

The description of rcu\_osc\_off is shown as below:

**Table 3-372. Function rcu\_osc\_off**

<b>Function name</b>	rcu_osc_off
<b>Function prototype</b>	void rcu_osc_off(rcu_osc_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 3-337. Enum rcu_osc_type_enum</a>



<i>RCU_HXTAL</i>	HXTAL
<i>RCU_LXTAL</i>	LXTAL
<i>RCU_IRC48M</i>	IRC48M
<i>RCU_IRC32K</i>	IRC32K
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn off the high speed crystal oscillator */
rcu_osci_off(RCU_HXTAL);
```

## rcu\_osci\_start\_enable (for GD32C211)

The description of rcu\_osci\_start\_enable is shown as below:

**Table 3-373. Function rcu\_osci\_start\_enable**

<b>Function name</b>	rcu_osci_start_enable
<b>Function prototype</b>	void rcu_osci_start_enable(rcu_osci_type_enum osci)
<b>Function descriptions</b>	The startup circuit enable when osci is ready
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-337. Enum rcu_osci_type_enum</a>
<i>RCU_LXTAL</i>	LXTAL
<i>RCU_IRC32K</i>	IRC32K
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* The startup circuit enable when IRC32K is ready */
rcu_osci_start_enable(RCU_IRC32K);
```

## rcu\_osci\_start\_disable (for GD32C211)

The description of rcu\_osci\_start\_disable is shown as below:

**Table 3-374. Function rcu\_osci\_start\_disable**

<b>Function name</b>	rcu_osci_start_disable
<b>Function prototype</b>	void rcu_osci_start_disable(rcu_osci_type_enum osci)

<b>Function descriptions</b>	The startup circuit disable when osci is ready
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-337. Enum rcu_osci_type_enum</a>
<i>RCU_LXTAL</i>	LXTAL
<i>RCU_IRC32K</i>	IRC32K
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* The startup circuit disable when IRC32K is ready */
rcu_osci_start_disable(RCU_IRC32K);
```

## rcu\_osci\_bypass\_mode\_enable

The description of rcu\_osci\_bypass\_mode\_enable is shown as below:

**Table 3-375. 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, HXTALEN or LXTALEN must be reset before it
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-337. Enum rcu_osci_type_enum</a>
<i>RCU_HXTAL</i>	HXTAL
<i>RCU_LXTAL</i>	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 3-376. 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, HXTALEN or LXTALEN must be reset before it
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-337. Enum rcu_osci_type_enum</a>
RCU_HXTAL	HXTAL
RCU_LXTAL	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\_irc48m\_adjust\_value\_set

The description of rcu\_irc48m\_adjust\_value\_set is shown as below:

**Table 3-377. Function rcu\_irc48m\_adjust\_value\_set**

<b>Function name</b>	rcu_irc48m_adjust_value_set
<b>Function prototype</b>	void rcu_irc48m_adjust_value_set(uint8_t irc48m_adjval)
<b>Function descriptions</b>	set the IRC48M adjust value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>irc48m_adjval</b>	IRC48M adjust value, must be between 0 and 0x3F
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the interrupt flags (API_ID(0x0032U)) */
```

```
rcu_irc48m_adjust_value_set();
```

## rcu\_lxtal\_stab\_reset\_enable

The description of rcu\_lxtal\_stab\_reset\_enable is shown as below:

**Table 3-378. Function rcu\_lxtal\_stab\_reset\_enable**

<b>Function name</b>	rcu_lxtal_stab_reset_enable
<b>Function prototype</b>	void rcu_lxtal_stab_reset_enable(void)
<b>Function descriptions</b>	enable LXTAL stabilization reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable LXTAL stabilization reset */
rcu_lxtal_stab_reset_enable();
```

## rcu\_lxtal\_stab\_reset\_disable

The description of rcu\_lxtal\_stab\_reset\_disable is shown as below:

**Table 3-379. Function rcu\_lxtal\_stab\_reset\_disable**

<b>Function name</b>	rcu_lxtal_stab_reset_disable
<b>Function prototype</b>	void rcu_lxtal_stab_reset_disable(void)
<b>Function descriptions</b>	disable LXTAL stabilization reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable LXTAL stabilization reset */
rcu_lxtal_stab_reset_disable();
```

## rcu\_hxtal\_clock\_monitor\_enable

The description of rcu\_hxtal\_clock\_monitor\_enable is shown as below:

**Table 3-380. 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 3-381. Function rcu\_hxtal\_clock\_monitor\_disable**

<b>Function name</b>	rcu_hxtal_clock_monitor_disable
<b>Function prototype</b>	void rcu_hxtal_clock_monitor_disable(void)
<b>Function descriptions</b>	disable the HXTAL clock monitor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the HXTAL clock monitor */
rcu_hxtal_clock_monitor_disable();
```

## rcu\_lxtal\_clock\_monitor\_enable

The description of rcu\_lxtal\_clock\_monitor\_enable is shown as below:

**Table 3-382. Function rcu\_lxtal\_clock\_monitor\_enable**

<b>Function name</b>	rcu_lxtal_clock_monitor_enable
<b>Function prototype</b>	void rcu_lxtal_clock_monitor_enable(void)
<b>Function descriptions</b>	enable the LXTAL 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 LXTAL clock monitor */
rcu_lxtal_clock_monitor_enable();
```

## rcu\_lxtal\_clock\_monitor\_disable

The description of rcu\_lxtal\_clock\_monitor\_disable is shown as below:

**Table 3-383. Function rcu\_lxtal\_clock\_monitor\_disable**

<b>Function name</b>	rcu_lxtal_clock_monitor_disable
<b>Function prototype</b>	void rcu_lxtal_clock_monitor_disable(void)
<b>Function descriptions</b>	disable the LXTAL clock monitor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the LXTAL clock monitor */
rcu_lxtal_clock_monitor_disable();
```

## rcu\_clock\_freq\_get

The description of rcu\_clock\_freq\_get is shown as below:

**Table 3-384. 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 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
CK_SYS	system clock frequency
CK_AHB	AHB clock frequency
CK_APB	APB clock frequency
CK_ADC	ADC clock frequency
CK_USART0	USART0 clock frequency
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
clock frequency of system, AHB, APB, ADC or USART0	clock frequency of system, AHB, APB, ADC or USART0

Example:

```
uint32_t temp_freq;

/* get the system clock frequency */

temp_freq = rcu_clock_freq_get(CK_SYS);
```

## rcu\_flag\_get

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

**Table 3-385. 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 3-333. Enum rcu_flag_enum</a>

<i>RCU_FLAG_IRC32KST</i> <i>B</i>	IRC32K stabilization flag
<i>RCU_FLAG_LXTALST</i> <i>B</i>	LXTAL stabilization flag
<i>RCU_FLAG_IRC48MS</i> <i>TB</i>	IRC48M stabilization flag
<i>RCU_FLAG_HXTALST</i> <i>B</i>	HXTAL stabilization flag
<i>RCU_FLAG_OBLRST</i>	option byte loader reset flag
<i>RCU_FLAG_BORRST</i>	BOR 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>	FWDGT reset flag
<i>RCU_FLAG_WWDGTR</i> <i>ST</i>	WWDGT reset flag
<i>RCU_FLAG_LPRST</i>	low-power reset flag
<i>RCU_FLAG_LCKCMD</i>	LXTAL clock failure detection 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 3-386. 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 the reset flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	



-	-
Return value	
-	-

Example:

```
/* clear all the reset flag */
```

```
rcu_all_reset_flag_clear();
```

## rcu\_interrupt\_enable

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

**Table 3-387. Function rcu\_interrupt\_enable**

<b>Function name</b>	rcu_interrupt_enable
<b>Function prototype</b>	void rcu_interrupt_enable(rcu_int_enum stab_int)
<b>Function descriptions</b>	enable the stabilization interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>stab_int</b>	clock stabilization interrupt, refer to <a href="#">Table 3-336. Enum rcu_int_enum</a>
<i>RCU_INT_IRC32KSTB</i>	IRC32K stabilization interrupt enable
<i>RCU_INT_LXTALSTB</i>	LXTAL stabilization interrupt enable
<i>RCU_INT_IRC48MSTB</i>	IRC48M stabilization interrupt enable
<i>RCU_INT_HXTALSTB</i>	HXTAL stabilization interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the HXTAL stabilization interrupt */
```

```
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

## rcu\_interrupt\_disable

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

**Table 3-388. Function rcu\_interrupt\_disable**

<b>Function name</b>	rcu_interrupt_disable
<b>Function prototype</b>	void rcu_interrupt_disable(rcu_int_enum stab_int)
<b>Function descriptions</b>	disable the stabilization interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>stab_int</b>	clock stabilization interrupt, refer to <a href="#">Table 3-336. Enum rcu_int_enum</a>
<i>RCU_INT_IRC32KSTB</i>	IRC32K stabilization interrupt disable
<i>RCU_INT_LXTALSTB</i>	LXTAL stabilization interrupt disable
<i>RCU_INT_IRC48MSTB</i>	IRC48M stabilization interrupt disable
<i>RCU_INT_HXTALSTB</i>	HXTAL stabilization interrupt disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the HXTAL stabilization interrupt */
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

## rcu\_interrupt\_flag\_get

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

**Table 3-389. Function rcu\_interrupt\_flag\_get**

<b>Function name</b>	rcu_interrupt_flag_get
<b>Function prototype</b>	FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag)
<b>Function descriptions</b>	get the clock stabilization interrupt and ckm flags
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>int_flag</b>	interrupt and ckm flags, refer to rcu_int_flag_enum
<i>RCU_INT_FLAG_IRC32KSTB</i>	IRC32K stabilization interrupt flag
<i>RCU_INT_FLAG_LXTALSTB</i>	LXTAL stabilization interrupt flag
<i>RCU_INT_FLAG_IRC48MSTB</i>	IRC48M stabilization interrupt flag
<i>RCU_INT_FLAG_HXTALSTB</i>	HXTAL stabilization interrupt flag
<i>RCU_INT_FLAG_LXTALCKM</i>	LXTAL clock stuck interrupt flag
<i>RCU_INT_FLAG_CKM</i>	HXTAL clock stuck interrupt flag
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the clock stabilization interrupt flag */
if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){
}
```

## rcu\_interrupt\_flag\_clear

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

**Table 3-390. Function rcu\_interrupt\_flag\_clear**

Function name	rcu_interrupt_flag_clear
Function prototype	void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag_clear)
Function descriptions	clear the interrupt flags
Precondition	-
The called functions	-
Input parameter{in}	
int_flag_clear	clock stabilization and stuck interrupt flags clear, refer to <a href="#">Table 3-335. Enum rcu_int_flag_clear_enum</a>
RCU_INT_FLAG_IRC3 2KSTB_CLR	IRC32K stabilization interrupt flag clear
RCU_INT_FLAG_LXTA LSTB_CLR	LXTAL stabilization interrupt flag clear
RCU_INT_FLAG_IRC4 8MSTB_CLR	IRC48M stabilization interrupt flag clear
RCU_INT_FLAG_HXTA LSTB_CLR	HXTAL stabilization interrupt flag clear
RCU_INT_FLAG_LXTA LCKM_CLR	LXTAL clock stuck interrupt flag clear
RCU_INT_FLAG_CKM _CLR	clock stuck interrupt flag clear
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the interrupt HXTAL stabilization interrupt flag */
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

## 3.15. 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 [3.15.1](#), the FWDGT firmware functions are introduced in chapter [3.15.2](#).

### 3.15.1. Descriptions of Peripheral registers

RTC registers are listed in the table shown as below:

**Table 3-391. 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_TYPE	RTC typeregister
RTC_ALRM0SS	RTC alarm 0 sub second register
RTC_BKP0	RTC backup 0 register
RTC_BKP1	RTC backup 1 register
RTC_BKP2	RTC backup 2 register
RTC_BKP3	RTC backup 3 register

### 3.15.2. Descriptions of Peripheral functions

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

**Table 3-392. RTC firmware function**

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_register_sync_wait	wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are

Function name	Function description
	updated
rtc_alarm_enable	enable RTC alarm
rtc_alarm_disable	disable RTC alarm
rtc_alarm_config	configure RTC alarm
rtc_alarm_subsecond_config	configure subsecond of RTC alarm
rtc_alarm_get	get RTC alarm
rtc_subsecond_get	get current subsecond value
rtc_init	initialize RTC registers
rtc_init_mode_enter	enter RTC init mode
rtc_init_mode_exit	exit RTC init mode
rtc_current_time_get	get current time and date
rtc_alarm_subsecond_get	get RTC alarm subsecond
rtc_deinit	reset most of the RTC registers
rtc_hour_adjust	ajust the daylight saving time by adding or subtracting one hour from the current time
rtc_second_adjust	ajust RTC second or subsecond value of current time
rtc_refclock_detection_enable	enable RTC reference clock detection function
rtc_refclock_detection_disable	disable RTC reference clock detection function
rtc_smooth_calibration_config	configure RTC smooth calibration
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_calibration_output_config	configure RTC calibration output source
rtc_output_pin_select	select the RTC output pin
rtc_alarm_output_config	configure RTC alarm output source
rtc_interrupt_enable	enable specified RTC interrupt
rtc_interrupt_disable	disble specified RTC interrupt
rtc_flag_get	check specified flag
rtc_flag_clear	clear specified flag

## Structure rtc\_parameter\_struct

Table 3-393. rtc\_parameter\_struct

Member name	Function description
year	RTC year value: 0x0 - 0x99(BCD format)
month	RTC month value (BCD format)
date	RTC date value: 0x1 - 0x31(BCD format)
day_of_week	RTC weekday value(BCD format)
hour	RTC hour value: 0x1 - 0x12(BCD format) or 0x0 - 0x23(BCD format)
minute	RTC minute value: 0x0 - 0x59(BCD format)
second	RTC second value: 0x0 - 0x59(BCD format)

factor_asyn	RTC asynchronous prescaler value: 0x0 - 0x7F
factor_syn	RTC synchronous prescaler value: 0x0 - 0x7FFF
am_pm	RTC AM/PM value
display_format	RTC time notation

### Structure rtc\_alarm\_struct

**Table 3-394. rtc\_alarm\_struct**

Member name	Function description
alarm_mask	RTC alarm mask
weekday_or_date	specify RTC alarm is on date or weekday
alarm_day	RTC alarm date or weekday value(BCD format)
alarm_hour	RTC alarm hour value: 0x1 - 0x12(BCD format) or 0x0 - 0x23(BCD format)
alarm_minute	RTC alarm minute value: 0x0 - 0x59(BCD format)
alarm_second	RTC alarm second value: 0x0 - 0x59(BCD format)
am_pm	RTC alarm AM/PM value

### Structure rtc\_timestamp\_struct

**Table 3-395. rtc\_timestamp\_struct**

Member name	Function description
timestamp_month	RTC time-stamp month value(BCD format)
timestamp_date	RTC time-stamp date value: 0x1 - 0x31(BCD format)
timestamp_day	RTC time-stamp weekday value(BCD format)
timestamp_hour	RTC time-stamp hour value(BCD format): 0x1 - 0x12(BCD format) or 0x0 - 0x23(BCD format)
timestamp_minute	RTC time-stamp minute value: 0x0 - 0x59(BCD format)
timestamp_second	RTC time-stamp second value: 0x0 - 0x59(BCD format)
am_pm	RTC time-stamp AM/PM value

### rtc\_bypass\_shadow\_enable

The description of rtc\_bypass\_shadow\_enable is shown as below:

**Table 3-396. rtc\_bypass\_shadow\_enable**

<b>Function name</b>	rtc_bypass_shadow_enable
<b>Function prototype</b>	void rtc_bypass_shadow_enable(void);
<b>Function descriptions</b>	enable RTC bypass shadow registers function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* enable RTC bypass shadow registers function*/
```

```
rtc_bypass_shadow_enable();
```

### rtc\_bypass\_shadow\_disable

The description of rtc\_bypass\_shadow\_disable shown as below:

**Table 3-397. rtc\_bypass\_shadow\_disable**

<b>Function name</b>	rtc_bypass_shadow_disable
<b>Function prototype</b>	void rtc_bypass_shadow_disable(void);
<b>Function descriptions</b>	disable RTC bypass shadow registers function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable RTC bypass shadow registers function*/
```

```
rtc_bypass_shadow_disable();
```

### rtc\_register\_sync\_wait

The description of rtc\_register\_sync\_wait is shown as below:

**Table 3-398. 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>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/*wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the
shadow registers are updated*/
```

```
ErrStatus error_status = rtc_register_sync_wait();
```

### rtc\_alarm\_enable

The description of rtc\_alarm\_enable is shown as below:

**Table 3-399. Function rtc\_alarm\_enable**

Function name	rtc_alarm_enable
Function prototype	void rtc_alarm_enable();
Function descriptions	enable RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*enable RTC alarm*/
```

```
rtc_alarm_enable();
```

### rtc\_alarm\_disable

The description of rtc\_alarm\_disable is shown as below:

**Table 3-400. Function rtc\_alarm\_disable**

Function name	rtc_alarm_disable
Function prototype	ErrStatus rtc_alarm_disable();
Function descriptions	disable RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-



Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/*disable RTC alarm*/
```

```
ErrStatus error_status = rtc_alarm_disable();
```

### rtc\_alarm\_config

The description of rtc\_alarm\_config is shown as below:

**Table 3-401. Function rtc\_alarm\_config**

Function name	rtc_alarm_config
Function prototype	void rtc_alarm_config(rtc_alarm_struct *rtc_alarm_time);
Function descriptions	configure RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Input parameter{in}	
rtc_alarm_time	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 3-394. rtc_alarm_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

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 3-402. Function rtc\_alarm\_subsecond\_config**

Function name	rtc_alarm_subsecond_config
Function prototype	void rtc_alarm_subsecond_config(uint32_t mask_subsecond, uint32_t subsecond)
Function descriptions	configure subsecond of RTC alarm
Precondition	-

<b>The called functions</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_ALARM0SS_SSC[14:1], and RTC_ALARM0SS_SSC[0] is to be compared
<i>RTC_MASKSSC_2_14</i>	mask RTC_ALARM0SS_SSC[14:2], and RTC_ALARM0SS_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 3-403. 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>The called functions</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 3-394. rtc_alarm_struct</a>
<b>Return value</b>	
-	-

Example:

```
/*disable RTC alarm*/
```

```
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 3-404. Function rtc\_alarm\_subsecond\_get**

<b>Function name</b>	rtc_alarm_subsecond_get
<b>Function prototype</b>	uint32_t rtc_alarm_subsecond_get();
<b>Function descriptions</b>	get RTC alarm subsecond
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	RTC alarm subsecond value(0x0-0x3FFF)

Example:

```
/*get RTC alarm subsecond*/
```

```
uint32_t subsecond = rtc_alarm_subsecond_get();
```

## rtc\_init\_mode\_enter

The description of rtc\_init\_mode\_enter is shown as below:

**Table 3-405. Function rtc\_init\_mode\_enter**

<b>Function name</b>	rtc_init_mode_enter
<b>Function prototype</b>	ErrStatus rtc_init_mode_enter(void);
<b>Function descriptions</b>	enter RTC init mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/*enter RTC init mode*/
```

```
ErrStatus error_status = rtc_init_mode_enter();
```

## rtc\_init

The description of rtc\_init is shown as below:

**Table 3-406. Function rtc\_init**

<b>Function name</b>	rtc_init
<b>Function prototype</b>	ErrStatus rtc_init(rtc_parameter_struct* rtc_initpara_struct);
<b>Function descriptions</b>	initialize RTC registers
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_initpara_struct</b>	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 3-393. rtc_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```

/* initialize RTC registers */

rtc_parameter_struct rtc_initpara;

rtc_interrupt_disable(RTC_INT_SECOND);

rtc_initpara.factor_asyn = prescaler_a;

rtc_initpara.factor_syn = prescaler_s;

rtc_initpara.year = 0x16;

rtc_initpara.day_of_week = RTC_SATURDAY;

rtc_initpara.month = RTC_APR;

rtc_initpara.date = 0x30;

rtc_initpara.display_format = RTC_24HOUR;

rtc_initpara.am_pm = RTC_AM;

rtc_init(&rtc_initpara);

```

### rtc\_init\_mode\_exit

The description of rtc\_init\_mode\_exit is shown as below:

**Table 3-407. Function rtc\_init\_mode\_exit**

<b>Function name</b>	rtc_init_mode_exit
<b>Function prototype</b>	void rtc_init_mode_exit(void);
<b>Function descriptions</b>	exit RTC init mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/*exit RTC init mode*/

rtc_init_mode_exit();

```

### rtc\_current\_time\_get

The description of rtc\_current\_time\_get is shown as below:

Table 3-408. Function rtc\_current\_time\_get

Function name	rtc_current_time_get
Function prototype	void rtc_current_time_get(rtc_parameter_struct* rtc_initpara_struct);
Function descriptions	get current time and date
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
rtc_initpara_struct	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 3-393. rtc_parameter_struct</a>
Return value	
-	-

Example:

```
/*get current time and date*/

rtc_parameter_struct rtc_initpara_struct;

rtc_current_time_get(&rtc_initpara_struct);
```

### rtc\_subsecond\_get

The description of rtc\_subsecond\_get is shown as below:

Table 3-409. Function rtc\_subsecond\_get

Function name	rtc_subsecond_get
Function prototype	uint32_t rtc_subsecond_get(void);
Function descriptions	get current subsecond value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	current subsecond value(0x00-0xFFFF)

Example:

```
/*get current subsecond value*/

uint32_t sub_second = rtc_subsecond_get();
```

## rtc\_deinit

The description of rtc\_deinit is shown as below:

**Table 3-410. Function rtc\_deinit**

<b>Function name</b>	rtc_deinit
<b>Function prototype</b>	ErrStatus rtc_deinit(void);
<b>Function descriptions</b>	reset most of the RTC registers
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable/ rcu_periph_reset_disable -
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* reset most of the RTC registers*/
ErrStatus error_status = rtc_deinit();
```

## rtc\_hour\_adjust

The description of rtc\_hour\_adjust is shown as below:

**Table 3-411. rtc\_hour\_adjust**

<b>Function name</b>	rtc_hour_adjust
<b>Function prototype</b>	void rtc_hour_adjust(uint32_t operation);
<b>Function descriptions</b>	adjust the daylight saving time by adding or subtracting one hour from the current time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>operation</b>	hour ajustment operation
<i>RTC_CTL_A1H</i>	add one hour
<i>RTC_CTL_S1H</i>	substract 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 3-412. rtc\_second\_adjust**

<b>Function name</b>	rtc_second_adjust
<b>Function prototype</b>	ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);
<b>Function descriptions</b>	adjust RTC second or subsecond value of current time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>add</b>	add 1s to current time or not
<i>RTC_SHIFT_ADD1S_RESET</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>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* adjust RTC second or subsecond value of current time */
```

```
ErrStatus error_status = rtc_second_adjust(RTC_SHIFT_ADD1S_SET, 0);
```

## rtc\_refclock\_detection\_enable

The description of rtc\_refclock\_detection\_enable shown as below:

**Table 3-413. rtc\_refclock\_detection\_enable**

<b>Function name</b>	rtc_refclock_detection_enable
<b>Function prototype</b>	ErrStatus rtc_refclock_detection_enable(void);
<b>Function descriptions</b>	enable RTC reference clock detection function
<b>Precondition</b>	-
<b>The called functions</b>	rtc_init_mode_enter/rtc_init_mode_exit
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	



<b>ErrStatus</b>	ERROR or SUCCESS
------------------	------------------

Example:

```
/* enable RTC reference clock detection function*/
```

```
ErrStatus error_status = rtc_refclock_detection_enable();
```

### rtc\_refclock\_detection\_disable

The description of rtc\_refclock\_detection\_disable shown as below:

**Table 3-414. rtc\_refclock\_detection\_disable**

<b>Function name</b>	rtc_refclock_detection_disable
<b>Function prototype</b>	ErrStatus rtc_refclock_detection_disable(void);
<b>Function descriptions</b>	disable RTC reference clock detection function
<b>Precondition</b>	-
<b>The called functions</b>	rtc_init_mode_enter/rtc_init_mode_exit
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* disable RTC reference clock detection function*/
```

```
ErrStatus error_status = rtc_refclock_detection_disable();
```

### rtc\_smooth\_calibration\_config

The description of rtc\_smooth\_calibration\_config is shown as below:

**Table 3-415. rtc\_smooth\_calibration\_config**

<b>Function name</b>	rtc_smooth_calibration_config
<b>Function prototype</b>	ErrStatus rtc_smooth_calibration_config(uint32_t window, uint32_t plus, uint32_t minus);
<b>Function descriptions</b>	configure RTC smooth calibration
<b>Precondition</b>	-
<b>The called functions</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 smooth calibration */
```

```
ErrStatus error_status;
```

```
error_status = rtc_smooth_calibration_config(RTC_CALIBRATION_WINDOW_32S,  
RTC_CALIBRATION_PLUS_SET, 0x10);
```

### rtc\_timestamp\_enable

The description of rtc\_timestamp\_enable is shown as below:

**Table 3-416. Function rtc\_timestamp\_enable**

<b>Function name</b>	rtc_timestamp_enable
<b>Function prototype</b>	void rtc_timestamp_enable(uint32_t edge);
<b>Function descriptions</b>	enable RTC time-stamp
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>edge</b>	specify which edge to detect of time-stamp
<i>RTC_TIMESTAMP_RISING_EDGE</i>	rising edge is valid event edge for timestamp event
<i>RTC_TIMESTAMP_FALLING_EDGE</i>	falling edge is valid event edge for timestamp event
<b>Output parameter{out}</b>	
-	-
<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 3-417. 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>The called functions</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 3-418. 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>The called functions</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="#">错误!未找到引用源。</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 3-419. Function rtc\_timestamp\_subsecond\_get**

<b>Function name</b>	rtc_timestamp_subsecond_get
<b>Function prototype</b>	uint32_t rtc_timestamp_subsecond_get(void);
<b>Function descriptions</b>	get RTC time-stamp subsecond
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	RTC time-stamp subsecond value

Example:

```
/* get RTC time-stamp subsecond */
```

```
uint32_t subsecond = rtc_timestamp_subsecond_get();
```

### rtc\_calibration\_output\_config

The description of rtc\_calibration\_output\_config is shown as below:

**Table 3-420. rtc\_calibration\_output\_config**

<b>Function name</b>	rtc_calibration_output_config
<b>Function prototype</b>	void rtc_calibration_output_config(uint32_t source);
<b>Function descriptions</b>	configure rtc calibration output source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
source	specify signal to output
RTC_CALIBRATION_5 12HZ	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 512Hz signal
RTC_CALIBRATION_1 HZ	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 1Hz signal
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* when the LSE frequency is 32768Hz and the RTC_PSC
```

```
is the default value, output 1Hz signal */
```

```
rtc_calibration_output_config(RTC_CALIBRATION_1HZ);
```

### rtc\_alarm\_output\_config

The description of rtc\_alarm\_output\_config is shown as below:

**Table 3-421. Function rtc\_alarm\_output\_config**

Function name	rtc_alarm_output_config
Function prototype	void rtc_alarm_output_config(uint32_t source, uint32_t mode);
Function descriptions	configure rtc alarm output source
Precondition	-
The called functions	-
Input parameter{in}	
source	specify signal to output
RTC_ALARM0_HIGH	when the alarm0 flag is set, the output pin is high
RTC_ALARM0_LOW	when the alarm0 flag is set, the output pin is low
Input parameter{in}	
mode	specify the output pin mode when output alarm signal or auto wakeup signal
RTC_ALARM_OUTPUTPU T_OD	open drain mode
RTC_ALARM_OUTPUTPU T_PP	push pull mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure rtc alarm0 output source */
```

```
rtc_alarm_output_config(RTC_ALARM0_LOW, RTC_ALARM_OUTPUT_PP);
```

### rtc\_output\_pin\_select

The description of rtc\_output\_pin\_select is shown as below:

**Table 3-422. Function rtc\_output\_pin\_select**

Function name	rtc_output_pin_select
---------------	-----------------------

<b>Function prototype</b>	void rtc_output_pin_select(uint32_t outputpin);
<b>Function descriptions</b>	select the RTC output pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>outputpin</b>	specify the rtc output pin is PC13 or not
<i>RTC_OUT_PC13</i>	the rtc output pin is PC13
<i>RTC_OUT_PB2_PB14</i>	the rtc output pin is PB2 or PB14
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* specify the rtc output pin is PC13 */
```

```
rtc_output_pin_select(RTC_OUT_PC13);
```

### rtc\_interrupt\_enable

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

**Table 3-423. Function rtc\_interrupt\_enable**

<b>Function name</b>	rtc_interrupt_enable
<b>Function prototype</b>	void rtc_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable specified RTC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which interrupt source to be enabled
<i>RTC_INT_TIMESTAMP</i>	timestamp interrupt
<i>RTC_INT_ALARM0</i>	Alarm0 interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable specified RTC interrupt*/
```

```
rtc_interrupt_enable(RTC_INT_ALARM0);
```

### rtc\_interrupt\_disable

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

Table 3-424. Function rtc\_interrupt\_disable

<b>Function name</b>	rtc_interrupt_disable
<b>Function prototype</b>	void rtc_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable specified RTC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which RTC interrupt to disable
<i>RTC_INT_TIMESTAMP</i>	timestamp interrupt
<i>RTC_INT_ALARM0</i>	Alarm0 interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable specified RTC interrupt */
rtc_interrupt_disable(RTC_INT_ALARM0);
```

### rtc\_flag\_get

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

Table 3-425. Function rtc\_flag\_get

<b>Function name</b>	rtc_flag_get
<b>Function prototype</b>	FlagStatus rtc_flag_get(uint32_t flag);
<b>Function descriptions</b>	check specified flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag to check
<i>RTC_FLAG_SCP</i>	smooth calibration pending flag
<i>RTC_FLAG_TSOVR</i>	time-stamp overflow event flag
<i>RTC_FLAG_TS</i>	time-stamp event flag
<i>RTC_FLAG_ALARM0</i>	Alarm0 event flag
<i>RTC_FLAG_INIT</i>	init mode event flag
<i>RTC_FLAG_RSYN</i>	time and date registers synchronized event flag
<i>RTC_FLAG_YCM</i>	year parameter configured event flag
<i>RTC_FLAG_SOP</i>	shift function operation pending flag
<i>RTC_FLAG_ALARM0</i> <i>W</i>	Alarm0 written available flag
<b>Output parameter{out}</b>	
-	-

Return value	
FlagStatus	SET or RESET

Example:

```
/* check time-stamp event flag */
```

```
FlagStatus = rtc_flag_get(RTC_FLAG_TS);
```

### rtc\_flag\_clear

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

**Table 3-426. Function rtc\_flag\_clear**

<b>Function name</b>	rtc_flag_clear
<b>Function prototype</b>	void rtc_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear specified flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag to clear
<i>RTC_FLAG_TSOVR</i>	time-stamp overflow event flag
<i>RTC_FLAG_TS</i>	time-stamp event flag
<i>RTC_FLAG_ALARM0</i>	Alarm0 event flag
<i>RTC_FLAG_RSYN</i>	time and date registers synchronized event flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* cleartime-stamp event flag */
```

```
rtc_flag_clear(RTC_FLAG_TS);
```

## 3.16. SPI

The SPI/I2S module can communicate with external devices using the SPI protocol or the I2S audio protocol. The SPI/I2S registers are listed in chapter [3.16.1](#), the SPI/I2S firmware functions are introduced in chapter [3.16.2](#).

### 3.16.1. Descriptions of Peripheral registers

SPI/I2S registers are listed in the table shown as below:



**Table 3-427. SPI/I2S Registers**

Registers	Descriptions
SPI_CTL0	SPI control register 0
SPI_CTL1	SPI control register 1
SPI_STAT	SPI status register
SPI_DATA	SPI data register
SPI_CRCPOLY	SPI CRC polynomial register
SPI_RCRC	SPI receive CRC register
SPI_TCRC	SPI transmit CRC register
SPI_I2SCTL	SPI/I2S control register
SPI_I2SPSC	SPI/I2S clock prescaler register
SPI_QCTL	Quad-SPI mode control register

### 3.16.2. Descriptions of Peripheral functions

SPI/I2S firmware functions are listed in the table shown as below:

**Table 3-428. 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
spi_init	initialize SPI parameter
spi_enable	enable SPI
spi_disable	disable SPI
i2s_init	initialize I2S parameter
i2s_psc_config	configure I2S prescaler
i2s_enable	enable I2S
i2s_disable	disable I2S
spi_nss_output_enable	enable SPI NSS output function
spi_nss_output_disable	disable SPI NSS output function
spi_nss_internal_high	SPI NSS pin high level in software mode
spi_nss_internal_low	SPI NSS pin low level in software mode
spi_dma_enable	enable SPI DMA function
spi_dma_disable	disable SPI DMA function
spi_transmit_odd_config	configure SPI total number of data transmitting by DMA is odd or not
spi_receive_odd_config	configure SPI total number of data receiving by DMA is odd or not
spi_i2s_data_frame_format_config	configure SPI data frame format
spi_fifo_access_size_config	configure SPI access size to FIFO(8-bit or 16-bit)
spi_bidirectional_transfer_config	configure SPI bidirectional transfer direction
spi_i2s_data_transmit	SPI transmit data
spi_i2s_data_receive	SPI receive data

Function name	Function description
spi_crc_polynomial_set	set SPI CRC polynomial
spi_crc_polynomial_get	get SPI CRC polynomial
spi_crc_length_set	set CRC length
spi_crc_on	turn on SPI CRC function
spi_crc_off	turn off SPI CRC function
spi_crc_next	SPI next data is CRC value
spi_crc_get	get SPI CRC send value or receive value
spi_crc_error_clear	clear SPI CRC error flag status
spi_ti_mode_enable	enable SPI TI mode
spi_ti_mode_disable	disable SPI TI mode
spi_nssp_mode_enable	enable SPI NSS pulse mode
spi_nssp_mode_disable	disable SPI NSS pulse mode
spi_quad_enable	enable quad wire SPI
spi_quad_disable	disable quad wire SPI
spi_quad_write_enable	enable quad wire SPI write
spi_quad_read_enable	enable quad wire SPI read
spi_i2s_format_error_clear	clear SPI/I2S format 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 3-429. spi\_parameter\_struct**

Member name	Function description
device_mode	SPI master or slave (SPI_MASTER, SPI_SLAVE)
trans_mode	SPI transtype (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	SPI frame size (SPI_FRAME_SIZE_16BIT, SPI_FRAME_SIZE_8BIT)
nss	SPI NSS control by hardware or software (SPI_NSS_SOFT, SPI_NSS_HARD)
endian	SPI big endian or little endian (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)
clock_polarity_phase	SPI clock phase and polarity (SPI_CLK_PL_LOW_PH_1EDGE, SPI_CLK_PL_HIGH_PH_1EDGE, SPI_CLK_PL_LOW_PH_2EDGE, SPI_CLK_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 3-430. Function spi\_i2s\_deinit**

<b>Function name</b>	spi_i2s_deinit
<b>Function prototype</b>	void spi_i2s_deinit(uint32_t spi_periph);
<b>Function descriptions</b>	reset SPI and I2S peripheral
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI/I2S peripheral
<b>SPIx(x=0,1)</b>	SPI/I2S peripheral selection
<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 3-431. 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 parameter struct, the structure members can refer to members of the structure <a href="#">Table 3-429. spi_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the parameters of SPI struct */
```

```

spi_parameter_struct spi_struct;

spi_struct->device_mode = SPI_SLAVE;

spi_struct->trans_mode = SPI_TRANSMODE_FULLDUPLEX;

spi_struct->frame_size = SPI_FRAME_SIZE_8BIT;

spi_struct->nss = SPI_NSS_HARD;

spi_struct->clock_polarity_phase = SPI_CK_PL_LOW_PH_1EDGE;

spi_struct->prescale = SPI_PSC_2;

spi_struct_para_init(&spi_struct);

```

## spi\_init

The description of spi\_init is shown as below:

**Table 3-432. Function spi\_init**

Function name	spi_init
Function prototype	void spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);
Function descriptions	initialize SPI peripheral parameter
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx(x=0, 1)	SPI peripheral selection
Input parameter{in}	
spi_struct	SPI parameter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-429. spi_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize SPI0 */

spi_parameter_struct spi_init_struct;

spi_init_struct.trans_mode = SPI_TRANSMODE_BDTRANSMIT;

spi_init_struct.device_mode = SPI_MASTER;

spi_init_struct.frame_size = SPI_FRAME_SIZE_8BIT;

spi_init_struct.clock_polarity_phase = SPI_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 3-433. Function spi\_enable**

<b>Function name</b>	spi_enable
<b>Function prototype</b>	void spi_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable SPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* enable SPI0 */

spi_enable(SPI0);

```

## spi\_disable

The description of spi\_disable is shown as below:

**Table 3-434. Function spi\_disable**

<b>Function name</b>	spi_disable
<b>Function prototype</b>	void spi_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable SPIx
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 */
spi_disable(SPI0);
```

## i2s\_init

The description of i2s\_init is shown as below:

**Table 3-435. 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 I2S peripheral parameter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S peripheral
<i>SPIx(x=0)</i>	I2S peripheral selection
<b>Input parameter{in}</b>	
<b>mode</b>	I2S operation mode
<i>I2S_MODE_SLAVETX</i>	I2S slave transmit mode
<i>I2S_MODE_SLAVEX</i>	I2S slave receive mode
<i>I2S_MODE_MASTERTX</i> <i>X</i>	I2S master transmit mode
<i>I2S_MODE_MASTERR</i> <i>X</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 3-436. 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 I2S prescaler
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S peripheral
<i>SPIx(x=0)</i>	I2S peripheral selection
<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_ENABLER</i>	I2S master clock output enable
<i>I2S_MCKOUT_DISABLE</i>	I2S master clock output disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure 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 3-437. Function i2s\_enable**

<b>Function name</b>	i2s_enable
<b>Function prototype</b>	void i2s_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable I2S
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S peripheral
<i>SPIx(x=0)</i>	I2S peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2S0*/
```

```
i2s_enable(SPI0);
```



## i2s\_disable

The description of i2s\_disable is shown as below:

**Table 3-438. Function i2s\_disable**

<b>Function name</b>	i2s_disable
<b>Function prototype</b>	void i2s_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable I2S
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S peripheral
<b>SPIx(x=0)</b>	I2S peripheral selection
<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 3-439. Function spi\_nss\_output\_enable**

<b>Function name</b>	spi_nss_output_enable
<b>Function prototype</b>	void spi_nss_output_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable SPI NSS output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx(x=0,1)</b>	SPI peripheral selection
<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 3-440. Function spi\_nss\_output\_disable**

<b>Function name</b>	spi_nss_output_disable
<b>Function prototype</b>	void spi_nss_output_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable SPI NSS output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<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 3-441. Function spi\_nss\_internal\_high**

<b>Function name</b>	spi_nss_internal_high
<b>Function prototype</b>	void spi_nss_internal_high(uint32_t spi_periph);
<b>Function descriptions</b>	SPI NSS pin high level in software mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<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 3-442. Function spi\_nss\_internal\_low**

<b>Function name</b>	spi_nss_internal_low
<b>Function prototype</b>	void spi_nss_internal_low(uint32_t spi_periph);
<b>Function descriptions</b>	SPI NSS pin low level in software mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 NSS pin is pulled low level in software mode */
```

```
spi_nss_internal_low(SPI0);
```

## spi\_dma\_enable

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

**Table 3-443. Function spi\_dma\_enable**

<b>Function name</b>	spi_dma_enable
<b>Function prototype</b>	void spi_dma_enable(uint32_t spi_periph, uint8_t dma);
<b>Function descriptions</b>	enable SPI DMA function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<b>Input parameter{in}</b>	
<b>dma</b>	SPI DMA mode
<i>SPI_DMA_TRANSMIT</i>	SPI transmit data use DMA
<i>SPI_DMA_RECEIVE</i>	SPI receive data use DMA
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 transmit data DMA function */

spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);
```

### spi\_dma\_disable

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

**Table 3-444. Function spi\_dma\_disable**

<b>Function name</b>	spi_dma_disable
<b>Function prototype</b>	void spi_dma_disable(uint32_t spi_periph, uint8_t dma);
<b>Function descriptions</b>	disable SPI DMA function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<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\_transmit\_odd\_config

The description of spi\_transmit\_odd\_config is shown as below:

**Table 3-445. Function spi\_transmit\_odd\_config**

<b>Function name</b>	spi_transmit_odd_config
<b>Function prototype</b>	void spi_transmit_odd_config(uint32_t spi_periph, uint16_t odd);
<b>Function descriptions</b>	configure SPI0 total number of data to transmit by DMA is odd or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=1)</i>	SPI peripheral selection

Input parameter{in}	
<b>dma</b>	SPI DMA mode
<i>SPI_TXDMA_EVEN</i>	number of byte in TX DMA channel is even
<i>SPI_TXDMA_ODD</i>	number of byte in TX DMA channel is odd
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SPI1 total number of data to transmit by DMA is odd */
```

```
spi_transmit_odd_config (SPI1, SPI_TXDMA_ODD);
```

### spi\_receive\_odd\_config

The description of spi\_receive\_odd\_config is shown as below:

**Table 3-446. Function spi\_receive\_odd\_config**

<b>Function name</b>	spi_receive_odd_config
<b>Function prototype</b>	void spi_receive_odd_config(uint32_t spi_periph, uint16_t odd);
<b>Function descriptions</b>	configure SPI0 total number of data to receive by DMA is odd or not
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=1)</i>	SPI peripheral selection
Input parameter{in}	
<b>dma</b>	SPI DMA mode
<i>SPI_RXDMA_EVEN</i>	number of byte in RX DMA channel is even
<i>SPI_RXDMA_ODD</i>	number of byte in RX DMA channel is odd
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SPI1 total number of data to receive by DMA is odd */
```

```
spi_receive_odd_config (SPI1, SPI_RXDMA_ODD);
```

### spi\_i2s\_data\_frame\_format\_config

The description of spi\_i2s\_data\_frame\_format\_config is shown as below:

Table 3-447. Function `spi_i2s_data_frame_format_config`

Function name	<code>spi_i2s_data_frame_format_config</code>
Function prototype	<code>void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);</code>
Function descriptions	configure SPI/I2S data frame format
Precondition	-
The called functions	-
Input parameter{in}	
<code>spi_periph</code>	SPI peripheral
<code>SPIx(x=0, 1)</code>	SPI peripheral selection
Input parameter{in}	
<code>frame_format</code>	SPI frame size
<code>SPI_FRAME_SIZE_xBIT</code>	SPI frame size is x bits, x=4,5,..16
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SPI1 data frame format size is 16 bits */
```

```
spi_i2s_data_frame_format_config(SPI1, SPI_FRAME_SIZE_16BIT);
```

### `spi_fifo_access_size_config`

The description of `spi_fifo_access_size_config` is shown as below:

Table 3-448. Function `spi_fifo_access_size_config`

Function name	<code>spi_fifo_access_size_config</code>
Function prototype	<code>void spi_fifo_access_size_config(uint32_t spi_periph, uint16_t fifo_access_size);</code>
Function descriptions	configure SPI1 access size to FIFO(8bit or 16bit)
Precondition	-
The called functions	-
Input parameter{in}	
<code>spi_periph</code>	SPI peripheral
<code>SPIx(x=1)</code>	SPI peripheral selection
Input parameter{in}	
<code>frame_format</code>	SPI frame size
<code>SPI_HALFWORD_ACCESS</code>	half-word access to FIFO
<code>SPI_BYTE_ACCESS</code>	byte access to FIFO
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure SPI1 access size half word */
```

```
spi_fifo_access_size_config (SPI1, SPI_HALFWORD_ACCESS);
```

### sbi\_bidirectional\_transfer\_config

The description of sbi\_bidirectional\_transfer\_config is shown as below:

**Table 3-449. Function sbi\_bidirectional\_transfer\_config**

<b>Function name</b>	sbi_bidirectional_transfer_config
<b>Function prototype</b>	void sbi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);
<b>Function descriptions</b>	configure SPI bidirectional transfer direction
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<b>Input parameter{in}</b>	
<b>transfer_direction</b>	SPI transfer direction
<i>SPI_BIDIRECTIONAL_TRANSMIT</i>	SPI work in transmit-only mode
<i>SPI_BIDIRECTIONAL_RECEIVE</i>	SPI work in receive-only mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 works in transmit-only mode */
```

```
sbi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

### sbi\_i2s\_data\_transmit

The description of sbi\_i2s\_data\_transmit is shown as below:

**Table 3-450. Function sbi\_i2s\_data\_transmit**

<b>Function name</b>	sbi_i2s_data_transmit
<b>Function prototype</b>	void sbi_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(x=0, 1)</i>	SPI peripheral selection
<b>Input parameter{in}</b>	
<b>data</b>	16-bit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 transmit data */

uint16_t spi0_send_array[10];

uint8_t send_n = 1;

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 3-451. 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(x=0, 1)</i>	SPI peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	16-bit data

Example:

```
/* SPI0 receive data */

uint16_t spi0_receive_array[10];

uint8_t receive_n = 1;

spi0_receive_array[receive_n] = spi_i2s_data_receive(SPI0);
```



## spi\_crc\_polynomial\_set

The description of spi\_crc\_polynomial\_set is shown as below:

**Table 3-452. 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(x=0,1)</i>	SPI peripheral selection
<b>Input parameter{in}</b>	
<b>crc_poly</b>	CRC polynomial value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set SPI0 CRC polynomial */
uint16_t CRC_VALUE = 0x8;
spi_crc_polynomial_set(SPI0,CRC_VALUE);
```

## spi\_crc\_polynomial\_get

The description of spi\_crc\_polynomial\_get is shown as below:

**Table 3-453. Function spi\_crc\_polynomial\_get**

<b>Function name</b>	spi_crc_polynomial_get
<b>Function prototype</b>	uint16_t spi_crc_polynomial_get(uint32_t spi_periph);
<b>Function descriptions</b>	get SPI CRC polynomial
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	16-bit CRC polynomial (0-0xFFFF)

Example:

```
/* get SPI0 CRC polynomial */
```

```
uint16_t CRC_VALUE;
```

```
CRC_VALUE = spi_crc_polynomial_get(SPI0);
```

### spi\_crc\_length\_set

The description of spi\_crc\_length\_set is shown as below:

**Table 3-454. Function spi\_crc\_length\_set**

<b>Function name</b>	spi_crc_length_set
<b>Function prototype</b>	void spi_crc_length_set(uint32_t spi_periph, uint16_t crc_length);
<b>Function descriptions</b>	set CRC length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=1)</i>	SPI peripheral selection
<b>Input parameter{in}</b>	
<b>crc_length</b>	CRC length
<i>SPI_CRC_8BIT</i>	CRC length is 8 bits
<i>SPI_CRC_16BIT</i>	CRC length is 16 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set CRC length is 8 bits*/
```

```
spi_crc_length_set(SPI1, SPI_CRC_8BIT);
```

### spi\_crc\_on

The description of spi\_crc\_on is shown as below:

**Table 3-455. Function spi\_crc\_on**

<b>Function name</b>	spi_crc_on
<b>Function prototype</b>	void spi_crc_on(uint32_t spi_periph);
<b>Function descriptions</b>	turn on SPI CRC function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection

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 3-456. Function spi\_crc\_off**

Function name	spi_crc_off
Function prototype	void spi_crc_off(uint32_t spi_periph);
Function descriptions	turn off SPI CRC function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx(x=0, 1)	SPI peripheral selection
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 3-457. 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(x=0,1)</i>	SPI peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 next data is CRC value */
```

```
spi_crc_next(SPI0);
```

### spi\_crc\_get

The description of spi\_crc\_get is shown as below:

**Table 3-458. Function spi\_crc\_get**

<b>Function name</b>	spi_crc_get
<b>Function prototype</b>	uint16_t spi_crc_get(uint32_t spi_periph, uint8_t crc);
<b>Function descriptions</b>	get SPI CRC send value or receive value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<b>Input parameter{in}</b>	
<b>crc</b>	SPI crc value
<i>SPI_CRC_TX</i>	get transmit crc value
<i>SPI_CRC_RX</i>	get receive crc value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	16-bit CRC value (0-0xFFFF)

Example:

```
/* get SPI0 CRC send value */
```

```
uint16_t value;
```

```
value = spi_crc_get(SPI0, SPI_CRC_TX);
```

### spi\_crc\_error\_clear

The description of spi\_crc\_error\_clear is shown as below:

**Table 3-459. Function spi\_crc\_error\_clear**

<b>Function name</b>	spi_crc_error_clear
----------------------	---------------------

<b>Function prototype</b>	void spi_crc_error_clear(uint32_t spi_periph);
<b>Function descriptions</b>	clear SPI CRC error flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx(x=0,1)</b>	SPI peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear SPI CRC error flag status */
```

```
spi_crc_error_clear(SPI0);
```

### spi\_ti\_mode\_enable

The description of spi\_ti\_mode\_enable is shown as below:

**Table 3-460. Function spi\_ti\_mode\_enable**

<b>Function name</b>	spi_ti_mode_enable
<b>Function prototype</b>	void spi_ti_mode_enable (uint32_t spi_periph);
<b>Function descriptions</b>	enable SPI TI mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx(x=0,1)</b>	SPI peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI TI mode */
```

```
spi_ti_mode_enable(SPI0);
```

### spi\_ti\_mode\_disable

The description of spi\_ti\_mode\_disable is shown as below:

Table 3-461. Function spi\_ti\_mode\_disable

Function name	spi_ti_mode_disable
Function prototype	void spi_ti_mode_disable (uint32_t spi_periph);
Function descriptions	disable SPI TI mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx(x=0,1)	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI TI mode */
spi_ti_mode_disable(SPI0);
```

### spi\_nssp\_mode\_enable

The description of spi\_nssp\_mode\_enable is shown as below:

Table 3-462. Function spi\_nssp\_mode\_enable

Function name	spi_nssp_mode_enable
Function prototype	void spi_nssp_mode_enable(uint32_t spi_periph);
Function descriptions	enable SPI NSS pulse mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx(x=0,1)	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI NSS pulse mode */
spi_nssp_mode_enable(SPI0);
```

### spi\_nssp\_mode\_disable

The description of spi\_nssp\_mode\_disable is shown as below:

Table 3-463. Function spi\_nssp\_mode\_disable

Function name	spi_nssp_mode_disable
Function prototype	void spi_nssp_mode_disable(uint32_t spi_periph);
Function descriptions	disable SPI NSS pulse mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx(x=0,1)	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI NSS pulse mode */
spi_nssp_mode_disable(SPI0);
```

### spi\_quad\_enable

The description of spi\_quad\_enable is shown as below:

Table 3-464. Function spi\_quad\_enable

Function name	spi_quad_enable
Function prototype	void spi_quad_enable(uint32_t spi_periph);
Function descriptions	enable quad wire SPI
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx(x=1)	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable quad wire SPI */
spi_quad_enable(SPI1);
```

### spi\_quad\_disable

The description of spi\_quad\_disable is shown as below:

Table 3-465. Function spi\_quad\_disable

Function name	spi_quad_disable
Function prototype	void spi_quad_disable(uint32_t spi_periph);
Function descriptions	disable quad wire SPI
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx(x=1)	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable quad wire SPI */
spi_quad_disable(SPI1);
```

### spi\_quad\_write\_enable

The description of spi\_quad\_write\_enable is shown as below:

Table 3-466. Function spi\_quad\_write\_enable

Function name	spi_quad_write_enable
Function prototype	void spi_quad_write_enable(uint32_t spi_periph);
Function descriptions	enable quad wire SPI write
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx(x=1)	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable quad wire SPI write */
spi_quad_write_enable(SPI1);
```

### spi\_quad\_read\_enable

The description of spi\_quad\_read\_enable is shown as below:



Table 3-467. Function spi\_quad\_read\_enable

<b>Function name</b>	spi_quad_read_enable
<b>Function prototype</b>	void spi_quad_read_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable quad wire SPI read
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=1)</i>	SPI peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable quad wire SPI read */
spi_quad_read_enable(SPI1);
```

### spi\_i2s\_format\_error\_clear

The description of spi\_i2s\_format\_error\_clear is shown as below:

Table 3-468. Function spi\_i2s\_format\_error\_clear

<b>Function name</b>	spi_i2s_format_error_clear
<b>Function prototype</b>	void spi_i2s_format_error_clear(uint32_t spi_periph, uint32_t flag);
<b>Function descriptions</b>	clear SPI/I2S format error flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<b>Input parameter{in}</b>	
<b>flag</b>	SPI/I2S frame format error flag
<i>SPI_FLAG_FERR</i>	SPI format error flag
<i>I2S_FLAG_FERR</i>	I2S format error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear SPI format error flag status */
spi_i2s_format_error_clear(SPI0, SPI_FLAG_FERR);
```

## spi\_i2s\_flag\_get

The description of spi\_i2s\_flag\_get is shown as below:

**Table 3-469. Function spi\_i2s\_flag\_get**

<b>Function name</b>	spi_i2s_flag_get
<b>Function prototype</b>	FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag);
<b>Function descriptions</b>	get SPI and I2S flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<b>Input parameter{in}</b>	
<b>flag</b>	SPI/I2S flag status
<i>SPI_FLAG_TBE</i>	transmit buffer empty flag
<i>SPI_FLAG_RBNE</i>	receive buffer not empty flag
<i>SPI_FLAG_TRANS</i>	transmit on-going flag
<i>SPI_FLAG_RXORERR</i>	receive overrun error flag
<i>SPI_FLAG_CONFERR</i>	mode config error flag
<i>SPI_FLAG_CRCERR</i>	CRC error flag
<i>SPI_FLAG_FERR</i>	SPI format error interrupt flag
<i>I2S_FLAG_TBE</i>	transmit buffer empty flag
<i>I2S_FLAG_RBNE</i>	receive buffer not empty flag
<i>I2S_FLAG_CH</i>	channel side flag
<i>I2S_FLAG_TXURERR</i>	underrun error flag
<i>I2S_FLAG_TRANS</i>	transmit on-going flag
<i>I2S_FLAG_RXORERR</i>	overrun error flag
<i>I2S_FLAG_FERR</i>	I2S format error interrupt flag
only for SPI0	
<i>SPI_FLAG_TXLVL_EMPTY</i>	SPI TXFIFO is empty
<i>SPI_FLAG_TXLVL_QUARTER_FULL</i>	SPI TXFIFO is a quarter of full
<i>SPI_FLAG_TXLVL_HALF_FULL</i>	SPI TXFIFO is a half of full
<i>SPI_FLAG_TXLVL_FULL</i>	SPI TXFIFO is full
<i>SPI_FLAG_RXLVL_EMPTY</i>	SPI RXFIFO is empty
<i>SPI_FLAG_RXLVL_QUARTER_FULL</i>	SPI RXFIFO is a quarter of full
<i>SPI_FLAG_RXLVL_HALF_FULL</i>	SPI RXFIFO is a half of full

<i>F_FULL</i>	
<i>SPI_FLAG_RXLVL_FULL</i>	SPI RXFIFO is full
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get SPI0 transmit buffer empty flag status */
```

```
FlagStatus Flag = RESET;
```

```
Flag = spi_i2s_flag_get(SPI0, SPI_FLAG_TBE);
```

### spi\_i2s\_interrupt\_enable

The description of spi\_i2s\_interrupt\_enable is shown as below:

**Table 3-470. Function spi\_i2s\_interrupt\_enable**

<b>Function name</b>	spi_i2s_interrupt_enable
<b>Function prototype</b>	void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	enable SPI and I2S interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<b>Input parameter{in}</b>	
<b>interrupt</b>	SPI/I2S interrupt
<i>SPI_I2S_INT_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_ERR</i>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 transmit buffer empty interrupt */
```

```
spi_i2s_interrupt_enable(SPI0, SPI_I2S_INT_TBE);
```

## spi\_i2s\_interrupt\_disable

The description of spi\_i2s\_interrupt\_disable is shown as below:

**Table 3-471. Function spi\_i2s\_interrupt\_disable**

<b>Function name</b>	spi_i2s_interrupt_disable
<b>Function prototype</b>	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	disable SPI and I2S interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<b>Input parameter{in}</b>	
<b>interrupt</b>	SPI/I2S interrupt
<i>SPI_I2S_INT_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_ERR</i>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 transmit buffer empty interrupt */
```

```
spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_TBE);
```

## spi\_i2s\_interrupt\_flag\_get

The description of spi\_i2s\_interrupt\_flag\_get is shown as below:

**Table 3-472. Function spi\_i2s\_interrupt\_flag\_get**

<b>Function name</b>	spi_i2s_interrupt_flag_get
<b>Function prototype</b>	FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	get SPI and I2S interrupt status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<b>Input parameter{in}</b>	
<b>interrupt</b>	SPI/I2S interrupt flag status
<i>SPI_I2S_INT_FLAG_T</i>	transmit buffer empty interrupt

<i>BE</i>	
<i>SPI_I2S_INT_FLAG_R</i> <i>BNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_FLAG_R</i> <i>XORERR</i>	overrun interrupt
<i>SPI_INT_FLAG_CONF</i> <i>ERR</i>	config error interrupt
<i>SPI_INT_FLAG_CRCE</i> <i>RR</i>	CRC error interrupt
<i>I2S_INT_FLAG_TXUR</i> <i>ERR</i>	underrun error interrupt
<i>SPI_I2S_INT_FLAG_F</i> <i>ERR</i>	format error interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get SPI0 transmit buffer empty interrupt status */
```

```
FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE);
```

## 3.17. SYSCFG

The SYSCFG registers are listed in chapter [3.17.1](#), the SYSCFG firmware functions are introduced in chapter [3.17.2](#).

### 3.17.1. Descriptions of Peripheral registers

SYSCFG registers are listed in the table shown as below:

**Table 3-473. 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
SYSCFG_CFG1	System configuration register 1
SYSCFG_STAT	SYSCFG STAT register
SYSCFG_CFG2	System configuration register 2

Registers	Descriptions
SYSCFG_CFG3	System configuration register 3
SYSCFG_CPU_IRQ_LAT	IRQ Latency register
SYSCFG_TIMER0CFG0	TIMER0 configuration register 0
SYSCFG_TIMER0CFG1	TIMER0 configuration register 1
SYSCFG_TIMER2CFG0	TIMER2 configuration register 0
SYSCFG_TIMER2CFG1	TIMER2 configuration register 1

### 3.17.2. Descriptions of Peripheral functions

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

**Table 3-474. SYSCFG firmware function**

Function name	Function description
syscfg_deinit	deinit syscfg module
syscfg_i2c_fast_mode_plus_enable	enable I2C Fm+ mode
syscfg_i2c_fast_mode_plus_disable	disable I2C Fm+ mode
syscfg_pin_remap_enable	enable remap pin function for small packages
syscfg_pin_remap_disable	disable remap pin function for small packages
syscfg_bootmode_get	get the boot mode
syscfg_exti_line_config	configure the GPIO pin as EXTI Line
syscfg_lockup_enable	enable module lockup
syscfg_lockup_disable	disable module lockup
syscfg_sram_ecc_single_correctable_bit_get	get SRAM ECC single correctable bit
syscfg_sram_ecc_error_address_get	get SRAM ECC error address
syscfg_irq_latency_set	set the IRQ_LATENCY value
syscfg_interrupt_enable	enable interrupt
syscfg_interrupt_disable	disable interrupt
syscfg_interrupt_flag_get	interrupt flag get
syscfg_interrupt_flag_clear	interrupt flag clear

#### syscfg\_deinit

The description of syscfg\_deinit is shown as below:

**Table 3-475. 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\_i2c\_fast\_mode\_plus\_enable

The description of syscfg\_i2c\_fast\_mode\_plus\_enable is shown as below:

**Table 3-476. Function syscfg\_i2c\_fast\_mode\_plus\_enable**

Function name	syscfg_i2c_fast_mode_plus_enable
Function prototype	void syscfg_i2c_fast_mode_plus_enable(uint32_t syscfg_gpio);
Function descriptions	enable I2C Fm+ mode
Precondition	-
The called functions	-
Input parameter{in}	
syscfg_gpio	gpio pin I2C Fm+ mode
SYSCFG_PB6_FMPEN	PB6 pin I2C Fm+ mode
SYSCFG_PB7_FMPEN	PB7 pin I2C Fm+ mode
SYSCFG_PB8_FMPEN	PB8 pin I2C Fm+ mode
SYSCFG_PB9_FMPEN	PB9 pin I2C Fm+ mode
SYSCFG_I2C0_FMPE N	I2C0 Fm+ mode
SYSCFG_PA9_FMPEN	PA9 pin I2C Fm+ mode
SYSCFG_PA10_FMPE N	PA10 pin I2C Fm+ mode
SYSCFG_PC14_FMPE N	PC14 pin I2C Fm+ mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable PB6 I2C Fm+ mode */
syscfg_i2c_fast_mode_plus_enable(SYSCFG_PB6_FMPEN);
```

## syscfg\_i2c\_fast\_mode\_plus\_disable

The description of syscfg\_i2c\_fast\_mode\_plus\_disable is shown as below:

**Table 3-477. Function syscfg\_i2c\_fast\_mode\_plus\_disable**

<b>Function name</b>	syscfg_i2c_fast_mode_plus_disable
<b>Function prototype</b>	void syscfg_i2c_fast_mode_plus_disable(uint32_t syscfg_gpio);
<b>Function descriptions</b>	disable I2C Fm+ mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_gpio</b>	gpio pin I2C Fm+ mode
SYSCFG_PB6_FMPEN	PB6 pin I2C Fm+ mode
SYSCFG_PB7_FMPEN	PB7 pin I2C Fm+ mode
SYSCFG_PB8_FMPEN	PB8 pin I2C Fm+ mode
SYSCFG_PB9_FMPEN	PB9 pin I2C Fm+ mode
SYSCFG_I2C0_FMPE N	I2C0 Fm+ mode
SYSCFG_PA9_FMPEN	PA9 pin I2C Fm+ mode
SYSCFG_PA10_FMPE N	PA10 pin I2C Fm+ mode
SYSCFG_PC14_FMPE N	PC14 pin I2C Fm+ mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable PB6 I2C Fm+ mode */
```

```
syscfg_i2c_fast_mode_plus_disable(SYSCFG_PB6_FMPEN);
```

## syscfg\_pin\_remap\_enable

The description of syscfg\_pin\_remap\_enable is shown as below:

**Table 3-478. Function syscfg\_pin\_remap\_enable**

<b>Function name</b>	syscfg_pin_remap_enable
<b>Function prototype</b>	Pin remap enable
<b>Function descriptions</b>	enable I2C Fm+ mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>remap_pin</b>	gpio pin remap



<code>SYSCFG_CFG0_PA11_RMP</code>	PA11 remap to PA9
<code>SYSCFG_CFG0_PA12_RMP</code>	PA12 remap to PA10
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PA11 remap to PA9 */
```

```
syscfg_pin_remap_enable(SYSCFG_CFG0_PA11_RMP);
```

### syscfg\_pin\_remap\_disable

The description of `syscfg_pin_remap_disable` is shown as below:

**Table 3-479. Function `syscfg_pin_remap_disable`**

<b>Function name</b>	<code>syscfg_pin_remap_disable</code>
<b>Function prototype</b>	<code>void syscfg_pin_remap_disable(uint32_t syscfg_gpio);</code>
<b>Function descriptions</b>	Pin remap disable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>remap_pin</b>	Pin remap
<code>SYSCFG_CFG0_PA11_RMP</code>	PA11 remap to PA9
<code>SYSCFG_CFG0_PA12_RMP</code>	PA12 remap to PA10
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable PA11 remap to PA9 */
```

```
syscfg_pin_remap_disable(SYSCFG_CFG0_PA11_RMP);
```

### syscfg\_bootmode\_get

The description of `syscfg_bootmode_get` is shown as below:

**Table 3-480. Function `syscfg_bootmode_get`**

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

<b>Function prototype</b>	void syscfg_bootmode_get(void);
<b>Function descriptions</b>	get the current boot mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* get the current boot mode */

uint8_t boot_mode;

boot_mode = syscfg_bootmode_get();

```

### syscfg\_exti\_line\_config

The description of syscfg\_exti\_line\_config is shown as below:

**Table 3-481. Function syscfg\_exti\_line\_config**

<b>Function name</b>	syscfg_exti_line_config
<b>Function prototype</b>	void syscfg_exti_line_config(uint8_t exti_port, uint8_t exti_pin);
<b>Function descriptions</b>	configure the GPIO pin as EXTI Line
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exti_port</b>	specify the GPIO port used in EXTI
<i>EXTI_SOURCE_GPIOx</i>	x = A,B,C,D,F
<b>exti_pin</b>	specify the EXTI line
<i>EXTI_SOURCE_PINx</i>	GPIOAx = 0..15, GPIOBx = 0..15, GPIOCx = 0..15, GPIODx = 0..6,8,9, GPIOFx = 0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure the GPIO pin as EXTI Line */

syscfg_exti_line_config(EXTI_SOURCE_GPIOA, EXTI_SOURCE_PIN0);

```

## syscfg\_lockup\_enable

The description of syscfg\_lockup\_enable is shown as below:

**Table 3-482. Function syscfg\_lockup\_enable**

<b>Function name</b>	syscfg_lockup_enable
<b>Function prototype</b>	void syscfg_lockup_enable(uint32_t lockup);
<b>Function descriptions</b>	enable module lockup
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lockup</b>	module lockup
SYSCFG_LOCKUP_LOCK	CPU lockup signal
SYSCFG_SRAM_LOCKUP	SRAM ECC check error lock signal
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* lock up CPU lockup signal */
```

```
syscfg_lockup_enable(SYSCFG_LOCKUP_LOCK);
```

## syscfg\_lockup\_disable

The description of syscfg\_lockup\_disable is shown as below:

**Table 3-483. Function syscfg\_lockup\_disable**

<b>Function name</b>	syscfg_lockup_disable
<b>Function prototype</b>	void syscfg_lockup_disable(uint32_t lockup);
<b>Function descriptions</b>	disable module lockup
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lockup</b>	module lockup
SYSCFG_LOCKUP_LOCK	CPU lockup signal
SYSCFG_SRAM_LOCKUP	SRAM ECC check error lock signal
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* disable lock up CPU lockup signal */
syscfg_lockup_disable(SYSCFG_LOCKUP_LOCK);
```

### syscfg\_sram\_ecc\_single\_correctable\_bit\_get

The description of syscfg\_sram\_ecc\_single\_correctable\_bit\_get is shown as below:

**Table 3-484. Function syscfg\_sram\_ecc\_single\_correctable\_bit\_get**

<b>Function name</b>	syscfg_sram_ecc_single_correctable_bit_get
<b>Function prototype</b>	uint32_t syscfg_sram_ecc_single_correctable_bit_get(void);
<b>Function descriptions</b>	SRAM ECC single correctable bit get
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>error_bits</b>	single correctable bit

Example:

```
/* SRAM ECC single correctable bit get */
Uint32_t error_bits;
error_bits = syscfg_sram_ecc_single_correctable_bit_get();
```

### syscfg\_sram\_ecc\_error\_address\_get

The description of syscfg\_sram\_ecc\_error\_address\_get is shown as below:

**Table 3-485. Function syscfg\_sram\_ecc\_error\_address\_get**

<b>Function name</b>	syscfg_sram_ecc_error_address_get
<b>Function prototype</b>	uint32_t syscfg_sram_ecc_error_address_get(void);
<b>Function descriptions</b>	SRAM ECC error address get
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

<b>addr</b>	SRAM ECC error address
-------------	------------------------

Example:

```
/* SRAM ECC single correctable bit get */

Uint32_t addr;

addr = syscfg_sram_ecc_error_address_get();
```

### syscfg\_irq\_latency\_set

The description of syscfg\_irq\_latency\_set is shown as below:

**Table 3-486. Function syscfg\_irq\_latency\_set**

<b>Function name</b>	syscfg_irq_latency_set
<b>Function prototype</b>	void syscfg_irq_latency_set(uint8_t irq_latency);
<b>Function descriptions</b>	set the IRQ_LATENCY value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>irq_latency</b>	IRQ_LATENCY value
0x00 - 0xFF	IRQ_LATENCY value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

syscfg_irq_latency_set(0xFF);
```

### syscfg\_pinmux\_config

The description of syscfg\_pinmux\_config is shown as below:

**Table 3-487. Function syscfg\_pinmux\_config**

<b>Function name</b>	syscfg_pinmux_config
<b>Function prototype</b>	void syscfg_pinmux_config(syscfg_pinmux_enum syscfg_pinmux_pin, syscfg_pin_enum syscfg_pin);
<b>Function descriptions</b>	configure the GPIO pinmux as specified
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_pinmux_pin</b>	specify the pinmux configuration
SYSCFG_PINMUX0_P	PB7 mapped to pin1

<i>B7</i>	
<i>SYSCFG_PINMUX0_P</i> <i>C14</i>	PC14 mapped to pin1
<i>SYSCFG_PINMUX1_P</i> <i>C2</i>	PC2 mapped to pin4
<i>SYSCFG_PINMUX1_P</i> <i>A0</i>	PA0 mapped to pin4
<i>SYSCFG_PINMUX1_P</i> <i>A1</i>	PA1 mapped to pin4
<i>SYSCFG_PINMUX1_P</i> <i>A2</i>	PA2 mapped to pin4
<i>SYSCFG_PINMUX2_P</i> <i>A8</i>	PA8 mapped to pin5
<i>SYSCFG_PINMUX2_P</i> <i>A11</i>	PA11 mapped to pin5
<i>SYSCFG_PINMUX3_P</i> <i>A14</i>	PA14 mapped to pin8
<i>SYSCFG_PINMUX3_P</i> <i>B6</i>	PB6 mapped to pin8
<i>SYSCFG_PINMUX3_P</i> <i>C15</i>	PC15 mapped to pin8
<b>syscfg_pin</b>	specify the SYSCFG pin to configure
<i>SYSCFG_PINx(x=1,4,5,8)</i>	Pin(1,4,5,8) configuration
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the pin 1 as PB7 */
```

```
syscfg_pinmux_config(SYSCFG_PINMUX0_PB7, SYSCFG_PIN1);
```

### syscfg\_interrupt\_enable

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

**Table 3-488. Function syscfg\_interrupt\_enable**

<b>Function name</b>	syscfg_interrupt_enable
<b>Function prototype</b>	void syscfg_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	interrupt enable
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
interrupt	interrupt type
<code>SYSCFG_CFG2_LXTALCSS_IE</code>	LXTAL css interrupt enable
<code>SYSCFG_CFG2_HXTALCSS_IE</code>	HXTAL css interrupt enable
<code>SYSCFG_CFG2_ECCMEIE</code>	Multi-bits (two bits) non-correction error NMI interrupt enable
<code>SYSCFG_CFG2_ECCSEIE</code>	Single bit correction error interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
syscfg_interrupt_enable(SYSCFG_CFG2_LXTALCSS_IE);
```

### syscfg\_interrupt\_disable

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

**Table 3-489. Function syscfg\_interrupt\_disable**

Function name	syscfg_interrupt_disable
Function prototype	void syscfg_interrupt_disable(uint32_t interrupt);
Function descriptions	interrupt disable
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	interrupt type
<code>SYSCFG_CFG2_LXTALCSS_IE</code>	LXTAL css interrupt disable
<code>SYSCFG_CFG2_HXTALCSS_IE</code>	HXTAL css interrupt disable
<code>SYSCFG_CFG2_ECCMEIE</code>	Multi-bits (two bits) non-correction error NMI interrupt disable
<code>SYSCFG_CFG2_ECCSEIE</code>	Single bit correction error interrupt disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
syscfg_interrupt_disable(SYSCFG_CFG2_LXTALCSS_IE);
```

## syscfg\_interrupt\_flag\_get

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

**Table 3-490. Function syscfg\_interrupt\_flag\_get**

<b>Function name</b>	syscfg_interrupt_flag_get
<b>Function prototype</b>	FlagStatus syscfg_interrupt_flag_get(uint32_t flag);
<b>Function descriptions</b>	Get interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt type
<i>SYSCFG_FLAG_ECCME</i>	SRAM two bits non-correction event flag
<i>SYSCFG_FLAG_ECCSE</i>	SRAM single bit correction event flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

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

```
FlagStatus Flag = RESET;
```

```
Flag = syscfg_interrupt_flag_get(SYSCFG_FLAG_ECCME);
```

## syscfg\_interrupt\_flag\_clear

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

**Table 3-491. Function syscfg\_interrupt\_flag\_clear**

<b>Function name</b>	syscfg_interrupt_flag_clear
<b>Function prototype</b>	FlagStatus syscfg_interrupt_flag_clear(uint32_t flag);
<b>Function descriptions</b>	Clear interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt type
<i>SYSCFG_FLAG_ECCME</i>	SRAM two bits non-correction event flag
<i>SYSCFG_FLAG_ECCSE</i>	SRAM single bit correction event flag
<b>Output parameter{out}</b>	
-	-



Return value	
-	-

Example:

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

```
syscfg_interrupt_flag_clear(SYSCFG_FLAG_ECCME);
```

## 3.18. 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 (TIMER2), general level2 timer (TIMER13), general level4 timer (TIMER15, TIMER16). The specific functions of different types of timer are different. The TIMER registers are listed in chapter [3.18.1](#), the TIMER firmware functions are introduced in chapter [3.18.2](#).

### 3.18.1. Descriptions of Peripheral registers

TIMERx registers are listed in the table shown as below:

**Table 3-492. TIMERx Registers**

Registers	Descriptions
TIMER_CTL0	Control register 0
TIMERx_CTL1	Control register 1
TIMERx_SMCFG	Slave mode configuration register
TIMERx_DMAINTEN	DMA and interrupt enable register
TIMERx_INTF	Interrupt flag register
TIMERx_SWEVG	Software event generation register
TIMERx_CHCTL0	Channel control register 0
TIMERx_CHCTL1	Channel control register 1
TIMERx_CHCTL2	Channel control register 2
TIMERx_CNT	Counter register
TIMERx_PSC	Prescaler register
TIMERx_CAR	Counter auto reload register
TIMERx_CREP	Counter repetition register
TIMERx_CH0CV	Channel 0 capture/compare value register
TIMERx_CH1CV	Channel 1 capture/compare value register
TIMERx_CH2CV	Channel 2 capture/compare value register
TIMERx_CH3CV	Channel 3 capture/compare value register
TIMERx_CCHP	TIMER complementary channel protection register
TIMERx_DMACFG	DMA configuration register

Registers	Descriptions
TIMERx_DMATB	DMA transfer buffer register
TIMER_CHCTL3	Channel control register 3
TIMERx_CH4CV	Channel 4 capture/compare value register
TIMERx_AFCTL0	Alternate function control register 0
TIMERx_AFCTL1	Alternate function control register 1
TIMERx_INSEL	input selection register
TIMERx_CFG(timerx, x=0, 1, 2, 8, 11, 14, 40)	Configuration register

### 3.18.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

**Table 3-493. TIMERx firmware function**

Function name	Function description
timer_deinit	deinit a timer
timer_struct_para_init	initialize the parameters of TIMER init parameter struct with the default values
timer_init	initialize TIMER counter
timer_enable	enable a timer
timer_disable	disable a timer
timer_auto_reload_shadow_enable	enable the auto reload shadow function
timer_auto_reload_shadow_disable	disable the auto reload shadow function
timer_update_event_enable	enable the update event
timer_update_event_disable	disable the update event
timer_counter_alignment	set TIMER counter alignment mode
timer_counter_up_direction	set TIMER counter up direction
timer_counter_down_direction	set TIMER counter down direction
timer_prescaler_config	configure TIMER prescaler
timer_repetition_value_config	configure TIMER repetition register value
timer_autoreload_value_config	configure TIMER autoreload register value
timer_counter_value_config	configure TIMER counter register value
timer_counter_read	read TIMER counter value
timer_prescaler_read	read TIMER prescaler value
timer_single_pulse_mode_config	configure TIMER single pulse mode
timer_update_source_config	configure TIMER update source
timer_dma_enable	enable the TIMER DMA
timer_dma_disable	disable the TIMER DMA
timer_channel_dma_request_source_select	channel DMA request source selection
timer_dma_transfer_config	configure the TIMER DMA transfer
timer_event_software_generate	software generate events
timer_break_struct_para_init	initialize the parameters of TIMER break parameter struct

Function name	Function description
	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_combined_3_phase_pwm_config	configure TIMER channel output combine mode
timer_channel_output_pulse_value_config	configure TIMER channel output pulse value
timer_channel_output_shadow_config	configure TIMER channel output shadow function
timer_channel_output_fast_config	configure TIMER channel output fast function
timer_channel_output_clear_config	configure TIMER channel output clear function
timer_channel_output_polarity_config	configure TIMER channel output polarity
timer_channel_complementary_output_polarity_config	configure TIMER channel complementary output polarity
timer_channel_output_state_config	configure TIMER channel enable state
timer_channel_complementary_output_state_config	configure TIMER channel complementary output enable state
timer_channel_input_struct_para_init	initialize the parameters of TIMER channel input parameter struct with the default values
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

Function name	Function description
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_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_input_selection_config	configure TIMER input selection
timer_write_chxval_register_config	configure TIMER write CHxVAL register selection
timer_output_value_selection_config	configure TIMER output value selection
timer_break_external_source_config	configure the TIMER break source
timer_break_external_polarity_config	configure TIMER break polarity
timer_flag_get	get TIMER flags
timer_flag_clear	clear TIMER flags
timer_interrupt_enable	enable the TIMER interrupt
timer_interrupt_disable	disable the TIMER interrupt
timer_interrupt_flag_get	get timer interrupt flag
timer_interrupt_flag_clear	clear TIMER interrupt flag

### Structure timer\_parameter\_struct

**Table 3-494. 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)
clockdivision	clock division value (TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4)
period	period value (0~65535)
repetitioncounter	the counter repetition value(0~255) only for GD32L235

### Structure timer\_break\_parameter\_struct

**Table 3-495. Structure timer\_break\_parameter\_struct**

Member name	Function description
runoffstate	run mode off-state(TIMER_ROS_STATE_ENABLE,

Member name	Function description
	TIMER_ROS_STATE_DISABLE)
idelloffstate	idle mode off-state(TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE)
deadtime	dead time(0~255)
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)
break0state	break enable(TIMER_BREAK0_ENABLE, TIMER_BREAK0_DISABLE)
break0filter	BREAK0 input filter(0-15)
break0polarity	Break0 polarity(TIMER_BREAK0_POLARITY_LOW, TIMER_BREAK0_POLARITY_HIGH)
break1state	break enable(TIMER_BREAK1_ENABLE, TIMER_BREAK1_DISABLE)
break1filter	BREAK1 input filter(0-15)
break1polarity	Break1 polarity(TIMER_BREAK1_POLARITY_LOW, TIMER_BREAK1_POLARITY_HIGH)

### Structure timer\_oc\_parameter\_struct

**Table 3-496. 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) only for GD32L235
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) only for GD32L235
ocidlestate	idle state of channel output(TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH) only for GD32L235
ocnidlestate	idle state of channel complementary output(TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH) only for GD32L235

### Structure timer\_ic\_parameter\_struct

**Table 3-497. 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)

Member name	Function description
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 3-498. Function timer\_deinit**

<b>Function name</b>	timer_deinit
<b>Function prototype</b>	void timer_deinit(uint32_t timer_periph);
<b>Function descriptions</b>	deinit a TIMER
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection (x=0,2,13,15,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset TIMER2 */
timer_deinit(TIMER2);
```

### timer\_struct\_para\_init

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

**Table 3-499. Function timer\_struct\_para\_init**

<b>Function name</b>	timer_struct_para_init
<b>Function prototype</b>	void timer_struct_para_init(timer_parameter_struct* initpara);
<b>Function descriptions</b>	initialize the parameters of TIMER init parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>initpara</b>	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-494. 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 3-500. 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
TIMEx	TIMER peripheral selection (x=0,2,13,15,16)
Input parameter{in}	
initpara	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-494. Structure timer_parameter_struct</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER2 */
```

```
timer_parameter_struct timer_initpara;
```

```
timer_initpara.prescaler = 47;
```

```
timer_initpara.alignedmode = TIMER_COUNTER_EDGE;
```

```
timer_initpara.counterdirection = TIMER_COUNTER_UP;
```

```
timer_initpara.period = 999;
```

```
timer_initpara.clockdivision = TIMER_CKDIV_DIV1;
```

```
timer_init(TIMER1,&timer_initpara);
```

## timer\_enable

The description of timer\_enable is shown as below:

**Table 3-501. 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</i>	TIMER peripheral selection (x=0,2,13,15,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER2 */
timer_enable(TIMER2);
```

## timer\_disable

The description of timer\_disable is shown as below:

**Table 3-502. 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>	TIMER peripheral selection (x=0,2,13,15,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:



```
/* disable TIMER2 */
```

```
timer_disable(TIMER2);
```

### timer\_auto\_reload\_shadow\_enable

The description of timer\_auto\_reload\_shadow\_enable is shown as below:

**Table 3-503. 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>	TIMER peripheral selection (x=0,2,13,15,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the TIMER2 auto reload shadow function */
```

```
timer_auto_reload_shadow_enable(TIMER2);
```

### timer\_auto\_reload\_shadow\_disable

The description of timer\_auto\_reload\_shadow\_disable is shown as below:

**Table 3-504. 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</i>	TIMER peripheral selection (x=0,2,13,15,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* disable the TIMER2 auto reload shadow function */
```

```
timer_auto_reload_shadow_disable(TIMER2);
```

### timer\_update\_event\_enable

The description of timer\_update\_event\_enable is shown as below:

**Table 3-505. 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</i>	TIMER peripheral selection (x=0,2,13,15,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER2 the update event */
```

```
timer_update_event_enable(TIMER2);
```

### timer\_update\_event\_disable

The description of timer\_update\_event\_disable is shown as below:

**Table 3-506. 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</i>	TIMER peripheral selection (x=0,2,13,15,16)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER2 the update event */
```

```
timer_update_event_disable(TIMER2);
```

### timer\_counter\_alignment

The description of timer\_counter\_alignment is shown as below:

**Table 3-507. Function timer\_counter\_alignment**

Function name	timer_counter_alignment
Function prototype	void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);
Function descriptions	set TIMER counter alignment mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	TIMER peripheral selection (x=0,2)
Input parameter{in}	
aligned	alignment mode
TIMER_COUNTER_EDGE	No center-aligned mode (edge-aligned mode). The direction of the counter is specified by the DIR bit.
TIMER_COUNTER_COUNTER_DOWN	Center-aligned and counting down assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0register). Only when the counter is counting down, compare interrupt flag of channels can be set.
TIMER_COUNTER_COUNTER_UP	Center-aligned and counting up assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0register). Only when the counter is counting up, compare interrupt flag of channels can be set.
TIMER_COUNTER_COUNTER_BOTH	Center-aligned and counting up/down assert mode. The counter counts under center-aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0 register). Both when the counter is counting up and counting down, compare interrupt flag of channels can be set.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set TIMER2 counter center-aligned and counting up assert mode */
timer_counter_alignment(TIMER2, TIMER_COUNTER_CENTER_UP);
```

### timer\_counter\_up\_direction

The description of timer\_counter\_up\_direction is shown as below:

**Table 3-508. 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</i>	TIMER peripheral selection (x=0,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set TIMER2 counter up direction */
timer_counter_up_direction(TIMER2);
```

### timer\_counter\_down\_direction

The description of timer\_counter\_down\_direction is shown as below:

**Table 3-509. 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</i>	TIMER peripheral selection (x=0,2)
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* set TIMER2 counter down direction */
```

```
timer_counter_down_direction(TIMER2);
```

### timer\_prescaler\_config

The description of timer\_prescaler\_config is shown as below:

**Table 3-510. Function timer\_prescaler\_config**

Function name	timer_prescaler_config
Function prototype	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint8_t pscreload);
Function descriptions	configure TIMER prescaler
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	TIMER peripheral selection (x=0,2,13,15,16)
Input parameter{in}	
prescaler	prescaler value (0~65535)
Input parameter{in}	
pscreload	prescaler reload mode
TIMER_PSC_RELOAD_NOW	the prescaler is loaded right now
TIMER_PSC_RELOAD_UPDATE	the prescaler is loaded at the next update event
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER2 prescaler */
```

```
timer_prescaler_config(TIMER2, 3000, TIMER_PSC_RELOAD_NOW);
```

### timer\_repetition\_value\_config

The description of timer\_repetition\_value\_config is shown as below:

Table 3-511. 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</i>	TIMER peripheral selection (x=0,15,16)
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 3-512. 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}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection (x=0,2,13,15,16)
Input parameter{in}	
autoreload	the counter auto-reload value (0-65535)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER2 autoreload register value */
```

```
timer_autoreload_value_config(TIMER2, 3000);
```

### timer\_counter\_value\_config

The description of timer\_counter\_value\_config is shown as below:

**Table 3-513. Function timer\_counter\_value\_config**

<b>Function name</b>	timer_counter_value_config
<b>Function prototype</b>	void timer_counter_value_config(uint32_t timer_periph, uint16_t counter);
<b>Function descriptions</b>	configure TIMER counter register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection (x=0,2,13,15,16)
<b>Input parameter{in}</b>	
<b>counter</b>	the counter value(0-65535)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER2 counter register value */
```

```
timer_counter_value_config(TIMER2, 3000);
```

### timer\_counter\_read

The description of timer\_counter\_read is shown as below:

**Table 3-514. Function timer\_counter\_read**

<b>Function name</b>	timer_counter_read
<b>Function prototype</b>	uint32_t timer_counter_read(uint32_t timer_periph);
<b>Function descriptions</b>	read TIMER counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection (x=0,2,13,15,16)

Output parameter{out}	
-	-
Return value	
uint32_t	counter value(0~65535)

Example:

```
/* read TIMER1 counter value */
uint32_t i = 0;
i = timer_counter_read(TIMER1);
```

### timer\_prescaler\_read

The description of timer\_prescaler\_read is shown as below:

**Table 3-515. 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	TIMER peripheral selection (x=0,2,13,15,16)
Output parameter{out}	
-	-
Return value	
uint16_t	prescaler register value (0~65535)

Example:

```
/* read TIMER2 prescaler value */
uint16_t i = 0;
i = timer_prescaler_read(TIMER2);
```

### timer\_single\_pulse\_mode\_config

The description of timer\_single\_pulse\_mode\_config is shown as below:

**Table 3-516. Function timer\_single\_pulse\_mode\_config**

Function name	timer_single_pulse_mode_config
Function prototype	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);
Function descriptions	configure TIMER single pulse mode



<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 (x=0,2,13,15,16)
<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 TIMER2 single pulse mode */
```

```
timer_single_pulse_mode_config(TIMER2, TIMER_SP_MODE_SINGLE);
```

### timer\_update\_source\_config

The description of timer\_update\_source\_config is shown as below:

**Table 3-517. 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>	TIMER peripheral selection (x=0,2,13,15,16)
<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.

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER2 update only by counter overflow/underflow */
timer_update_source_config(TIMER2, TIMER_UPDATE_SRC_REGULAR);
```

### timer\_dma\_enable

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

**Table 3-518. Function timer\_dma\_enable**

Function name	timer_dma_enable
Function prototype	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
Function descriptions	enable the TIMER DMA
Precondition	-
The called functions	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>dma</b>	timer DMA source enable
<i>TIMER_DMA_UPD</i>	update DMA enable TIMERx(x=0,2,13,15,16)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable TIMERx(x=0,2,13,15,16)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable TIMERx(x=0,2)
<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)
<i>TIMER_DMA_TRGD</i>	trigger DMA enable TIMERx(x=0,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the TIMER2 update DMA */
```

```
timer_dma_enable(TIMER2, TIMER_DMA_UPD);
```

### timer\_dma\_disable

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

**Table 3-519. Function timer\_dma\_disable**

<b>Function name</b>	timer_dma_disable
<b>Function prototype</b>	void timer_dma_disable(uint32_t timer_periph, uint16_t dma);
<b>Function descriptions</b>	disable the TIMER DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>dma</b>	timer DMA source disable
<i>TIMER_DMA_UPD</i>	update DMA enable TIMERx(x=0,2,13,15,16)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable TIMERx(x=0,2,13,15,16)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable TIMERx(x=0,2)
<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)
<i>TIMER_DMA_TRGD</i>	trigger DMA enable TIMERx(x=0,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the TIMER2 update DMA */
```

```
timer_dma_disable(TIMER2 TIMER_DMA_UPD);
```

## timer\_channel\_dma\_request\_source\_select

The description of timer\_channel\_dma\_request\_source\_select is shown as below:

**Table 3-520. Function timer\_channel\_dma\_request\_source\_select**

<b>Function name</b>	timer_channel_dma_request_source_select
<b>Function prototype</b>	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
<b>Function descriptions</b>	channel DMA request source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection TIMERx(x=0,2,15,16)
<b>Input parameter{in}</b>	
<b>dma_request</b>	channel DMA request source selection
<i>TIMER_DMAREQUEST_CHANNELEVENT</i>	DMA request of channel n is sent when channel y event occurs
<i>TIMER_DMAREQUEST_UPDATEEVENT</i>	DMA request of channel n is sent when update event occurs
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* TIMER2 channel DMA request of channel n is sent when channel y event occurs */
```

```
timer_channel_dma_request_source_select(TIMER2,  
TIMER_DMAREQUEST_CHANNELEVENT);
```

## timer\_dma\_transfer\_config

The description of timer\_dma\_transfer\_config is shown as below:

**Table 3-521. Function timer\_dma\_transfer\_config**

<b>Function name</b>	timer_dma_transfer_config
<b>Function prototype</b>	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
<b>Function descriptions</b>	configure the TIMER DMA transfer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral

<i>TIMERx</i>	TIMER peripheral selection (x=0,2,15,16)
<b>Input parameter{in}</b>	
<b>dma_baseaddr</b>	DMA transfer access start address
<i>TIMER_DMACFG_DMA TA_CTL0</i>	DMA transfer address is TIMER_CTL0
<i>TIMER_DMACFG_DMA TA_CTL1</i>	DMA transfer address is TIMER_CTL1
<i>TIMER_DMACFG_DMA TA_SMCFG</i>	DMA transfer address is TIMER_SMCFG
<i>TIMER_DMACFG_DMA TA_DMAINTEN</i>	DMA transfer address is TIMER_DMAINTEN
<i>TIMER_DMACFG_DMA TA_INTF</i>	DMA transfer address is TIMER_INTF
<i>TIMER_DMACFG_DMA TA_SWEVG</i>	DMA transfer address is TIMER_SWEVG
<i>TIMER_DMACFG_DMA TA_CHCTL0</i>	DMA transfer address is TIMER_CHCTL0
<i>TIMER_DMACFG_DMA TA_CHCTL1</i>	DMA transfer address is TIMER_CHCTL1
<i>TIMER_DMACFG_DMA TA_CHCTL2</i>	DMA transfer address is TIMER_CHCTL2
<i>TIMER_DMACFG_DMA TA_CNT</i>	DMA transfer address is TIMER_CNT
<i>TIMER_DMACFG_DMA TA_PSC</i>	DMA transfer address is TIMER_PSC
<i>TIMER_DMACFG_DMA TA_CAR</i>	MA transfer address is TIMER_CAR
<i>TIMER_DMACFG_DMA TA_CH0CV</i>	DMA transfer address is TIMER_CH0CV
<i>TIMER_DMACFG_DMA TA_CH1CV</i>	DMA transfer address is TIMER_CH1CV
<i>TIMER_DMACFG_DMA TA_CH2CV</i>	DMA transfer address is TIMER_CH2CV
<i>TIMER_DMACFG_DMA TA_CH3CV</i>	DMA transfer address is TIMER_CH3CV
<i>TIMER_DMACFG_DMA TA_CCHP</i>	DMA transfer address is TIMER_CCHP
<i>TIMER_DMACFG_DMA TA_DMACFG</i>	DMA transfer address is TIMER_DMACFG
<b>Input parameter{in}</b>	
<b>dma_lenth</b>	DMA transfer count
<i>TIMER_DMACFG_DMA</i>	x=1..17, DMA transfer x time

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

Example:

```
/* configure the TIMER2 DMA transfer */
```

```
timer_dma_transfer_config(TIMER2, TIMER_DMACFG_DMATA_CTL0,  
TIMER_DMACFG_DMATC_5TRANSFER);
```

### timer\_event\_software\_generate

The description of timer\_event\_software\_generate is shown as below:

**Table 3-522. 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 TIMERx(x=0,2,13,15,16)
<i>TIMER_EVENT_SRC_CH0G</i>	channel 0 capture or compare event generation TIMERx(x=0,2,13,15,16)
<i>TIMER_EVENT_SRC_CH1G</i>	channel 1 capture or compare event generation TIMERx(x=0,2)
<i>TIMER_EVENT_SRC_CH2G</i>	channel 2 capture or compare event generation TIMERx(x=0,2)
<i>TIMER_EVENT_SRC_CH3G</i>	channel 3 capture or compare event generation TIMERx(x=0,2)
<i>TIMER_EVENT_SRC_CMTG</i>	channel commutation event generation TIMERx(x=0,15,16)
<i>TIMER_EVENT_SRC_TRG</i>	trigger event generation TIMERx(x=0,15,16)
<i>TIMER_EVENT_SRC_BREAK0</i>	break0 event generation TIMERx(x=0,15,16)

<i>TIMER_EVENT_SRC_B</i> <i>RK1G</i>	break1 event generation TIMERx(x=0,15,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* software generate update event*/
```

```
timer_event_software_generate(TIMER2, TIMER_EVENT_SRC_UPG);
```

### timer\_break\_struct\_para\_init

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

**Table 3-523. Function timer\_break\_struct\_para\_init**

<b>Function name</b>	timer_break_struct_para_init
<b>Function prototype</b>	void timer_break_struct_para_init(timer_break_parameter_struct* breakpara);
<b>Function descriptions</b>	initialize the parameters of TIMER break parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>breakpara</b>	TIMER break parameter struct, the structure members can refer to <a href="#">Structure timer_break_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize TIMER break parameter struct with a default value */
```

```
timer_break_parameter_struct timer_breakpara;
```

```
timer_break_struct_para_init(timer_breakpara);
```

### timer\_break\_config

The description of timer\_break\_config is shown as below:

**Table 3-524. Function timer\_break\_config**

<b>Function name</b>	timer_break_config
<b>Function prototype</b>	void timer_break_config(uint32_t timer_periph,

	timer_break_parameter_struct* breakpara);
<b>Function descriptions</b>	configure TIMER break function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>breakpara</b>	TIMER break parameter struct, the structure members can refer to <a href="#">Structure timer break parameter struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure TIMER0 break function */

timer_break_parameter_struct timer_breakpara;

timer_breakpara.runoffstate      = TIMER_ROS_STATE_DISABLE;

timer_breakpara.ideloffstate     = TIMER_IOS_STATE_DISABLE ;

timer_breakpara.deadtime         = 255;

timer_breakpara.break0polarity   = TIMER_BREAK0_POLARITY_LOW;

timer_breakpara.outputautostate  = TIMER_OUTAUTO_ENABLE;

timer_breakpara.protectmode      = TIMER_CCHP_PROT_0;

timer_breakpara.break0state      = TIMER_BREAK0_ENABLE;

timer_break_config(TIMER0, &timer_breakpara);

```

### timer\_break\_enable

The description of timer\_break\_enable is shown as below:

**Table 3-525. 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, 15, 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 3-526. 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, 15, 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 3-527. Function timer\_automatic\_output\_enable**

<b>Function name</b>	timer_automatic_output_enable
<b>Function prototype</b>	void timer_automatic_output_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable TIMER output automatic function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in

	TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 15, 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 3-528. 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, 15, 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 3-529. 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, 15, 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 3-530. 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, 15, 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 3-531. 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, 15, 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 3-532. 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 3-496. 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 3-533. Function timer\_channel\_output\_config**

<b>Function name</b>	timer_channel_output_config
<b>Function prototype</b>	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara);
<b>Function descriptions</b>	configure TIMER channel output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 TIMERx(x=0,2,13,15,16)
<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)
<i>TIMER_CH_4</i>	TIMER channel 4 TIMERx(x=0)
<b>Input parameter{in}</b>	
<b>ocpara</b>	TIMER channel output parameter struct, the structure members can refer

	to <a href="#">Table 3-496. Structure timer_oc_parameter_struct.</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER2 channel 0 output function */

timer_oc_parameter_struct timer_ocinitpara;

timer_ocinitpara.outputstate = TIMER_CCX_ENABLE;

timer_ocinitpara.ocpolarity = TIMER_OC_POLARITY_HIGH;

timer_channel_output_config(TIMER2, TIMER_CH_0, &timer_ocinitpara);
```

### timer\_channel\_output\_mode\_config

The description of timer\_channel\_output\_mode\_config is shown as below:

**Table 3-534. 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=0,2,13,15,16)
<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>	IMER channel 3 TIMERx(x=0,2)
<i>TIMER_CH_4</i>	IMER channel 4 TIMERx(x=0)
<b>Input parameter{in}</b>	
<b>ocmode</b>	channel output compare mode
<i>TIMER_OC_MODE_TIM</i>	timing mode

ING	
TIMER_OC_MODE_ACTIV	set the channel output
TIMER_OC_MODE_INACTIVE	clear the channel output
TIMER_OC_MODE_TOGGLE	toggle on match
TIMER_OC_MODE_LOW	force low mode
TIMER_OC_MODE_HIGH	force high mode
TIMER_OC_MODE_PWM0	PWM mode 0
TIMER_OC_MODE_PWM1	PWM mode 1
TIMER_OC_MODE_DELAYABLE_SPM0	Delayable SPM mode 0
TIMER_OC_MODE_DELAYABLE_SPM1	Delayable SPM mode 1
TIMER_OC_MODE_COMBINED_PWM0	Combined PWM mode 0
TIMER_OC_MODE_COMBINED_PWM1	Combined PWM mode 1
TIMER_OC_MODE_ASYMMETRIC_PWM0	Asymmetric PWM mode 0
TIMER_OC_MODE_ASYMMETRIC_PWM1	Asymmetric PWM mode 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER2 channel PWM 0 mode */
```

```
timer_channel_output_mode_config(TIMER2, TIMER_CH_0, TIMER_OC_MODE_PWM0);
```

### timer\_channel\_combined\_3\_phase\_pwm\_config

The description of timer\_channel\_combined\_3\_phase\_pwm\_config is shown as below:

**Table 3-535. Function timer\_channel\_combined\_3\_phase\_pwm\_config**

Function name	timer_channel_combined_3_phase_pwm_config
Function prototype	void timer_channel_combined_3_phase_pwm_config(uint32_t

	timer_periph, uint16_t channel, uint32_t state);
<b>Function descriptions</b>	onfigure TIMER channel output combine 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)
<i>TIMER_CH_1</i>	TIMER channel 1 TIMERx(x=0)
<i>TIMER_CH_2</i>	TIMER channel 2 TIMERx(x=0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel output combine mode */
```

```
timer_channel_combined_3_phase_pwm_config (TIMER0, TIMER_CH_0);
```

### timer\_channel\_output\_pulse\_value\_config

The description of timer\_channel\_output\_pulse\_value\_config is shown as below:

**Table 3-536. 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=0,2,13,15,16)
<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)
<i>TIMER_CH_4</i>	TIMER channel 4 TIMERx(x=0)
<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 TIMER2 channel 0 output pulse value */
```

```
timer_channel_output_pulse_value_config(TIMER2, TIMER_CH_0, 399);
```

### timer\_channel\_output\_shadow\_config

The description of timer\_channel\_output\_shadow\_config is shown as below:

**Table 3-537. 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=0,2,13,15,16)
<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)
<i>TIMER_CH_4</i>	TIMER channel 4

	TIMERx(x=0)
<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 TIMER2 channel 0 output shadow function */
```

```
timer_channel_output_shadow_config(TIMER2, 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 3-538. 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=0,2,13,15,16)
<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)
<i>TIMER_CH_4</i>	TIMER channel 4 TIMERx(x=0)

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

Example:

```
/* configure TIMER2 channel 0 output fast function */
```

```
timer_channel_output_fast_config(TIMER2, 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 3-539. 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>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER periphera
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 TIMERx(x=0,2,13,15,16)
<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)
<i>TIMER_CH_4</i>	TIMER channel 4 TIMERx(x=0)
Input parameter{in}	
<b>occlear</b>	channel output clear function

<i>TIMER_OC_CLEAR_ENABLE</i>	channel output clear function enable
<i>TIMER_OC_CLEAR_DISABLE</i>	channel output clear function disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER2 channel 0 output clear function */
```

```
timer_channel_output_clear_config(TIMER2, 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 3-540. 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=0,2,13,15,16)
<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)
<i>TIMER_CH_4</i>	TIMER channel 4 TIMERx(x=0)
<b>Input parameter{in}</b>	
<b>ocpolarity</b>	channel output polarity
<i>TIMER_OC_POLARITY</i>	channel output polarity is high

<i>_HIGH</i>	
<i>TIMER_OC_POLARITY</i> <i>_LOW</i>	channel output polarity is low
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER2 channel 0 output polarity */
```

```
timer_channel_output_polarity_config(TIMER2, 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 3-541. 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 <i>TIMERx</i> (x=0, 14, 40)
<i>TIMER_CH_1</i>	TIMER channel 1 <i>TIMERx</i> (x=0)
<i>TIMER_CH_2</i>	TIMER channel 2 <i>TIMERx</i> (x=0)
<b>Input parameter{in}</b>	
<b>ocnpolarity</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 3-542. 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=0,2,13,15,16)
<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)
<i>TIMER_CH_4</i>	TIMER channel 4 TIMERx(x=0)
<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 TIMER2 channel 0 enable state */
```

```
timer_channel_output_state_config(TIMER2, 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 3-543. Function timer\_channel\_complementary\_output\_state\_config**

<b>Function name</b>	timer_channel_complementary_output_state_config
<b>Function prototype</b>	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
<b>Function descriptions</b>	configure TIMER channel complementary output enable state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14,.. 16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 <i>TIMERx(x=0,15,16)</i>
<i>TIMER_CH_1</i>	TIMER channel 1 <i>TIMERx(x=0)</i>
<i>TIMER_CH_2</i>	TIMER channel 2 <i>TIMERx(x=0)</i>
<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 3-544. 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

Precondition	-
The called functions	-
Input parameter{in}	
icpara	TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-497. Structure timer_ic_parameter_struct</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER channel input parameter struct with a default value */
```

```
timer_ic_parameter_struct timer_icinitpara;
```

```
timer_channel_input_struct_para_init(&timer_icinitpara);
```

### timer\_input\_capture\_config

The description of timer\_input\_capture\_config is shown as below:

**Table 3-545. Function timer\_input\_capture\_config**

Function name	timer_input_capture_config
Function prototype	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);
Function descriptions	configure TIMER input capture parameter
Precondition	-
The called functions	timer_channel_input_capture_prescaler_config
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0 TIMERx(x=0,2,13,15,16)
TIMER_CH_1	TIMER channel 1 TIMERx(x=0,2)
TIMER_CH_2	TIMER channel 2 TIMERx(x=0,2)
TIMER_CH_3	TIMER channel 3 TIMERx(x=0,2)
Input parameter{in}	
icpara	TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-497. Structure timer_ic_parameter_struct</a> .
Output parameter{out}	



-	-
Return value	
-	-

Example:

```

/* configure TIMER2 input capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_capture_config(TIMER2, 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 3-546. 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=0,2,13,15,16)
<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>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 TIMER2 channel 0 input capture prescaler value */
```

```
timer_channel_input_capture_prescaler_config(TIMER2, 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 3-547. 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=0,2,13,15,16)
<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>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	channel capture compare register value (0~65535)

Example:

```
/* read TIMER1 channel 0 capture compare register value */
```

```
uint32_t ch0_value = 0;
```

```
ch0_value = timer_channel_capture_value_register_read (TIMER1, TIMER_CH_0);
```

### timer\_input\_pwm\_capture\_config

The description of timer\_input\_pwm\_capture\_config is shown as below:

**Table 3-548. 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</i>	TIMER peripheral selection TIMERx(x=0,2)
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0
<i>TIMER_CH_1</i>	TIMER channel 1
<b>Input parameter{in}</b>	
<b>icpwm</b>	TIMER channel input pwm parameter struct, the structure members can refer to <a href="#">Table 3-497. Structure timer_ic_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER2 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(TIMER2, TIMER_CH_0, &timer_icinitpara);
```

## timer\_hall\_mode\_config

The description of timer\_hall\_mode\_config is shown as below:

**Table 3-549. Function timer\_hall\_mode\_config**

<b>Function name</b>	timer_hall_mode_config
<b>Function prototype</b>	void timer_hall_mode_config(uint32_t timer_periph, uint8_t hallmode);
<b>Function descriptions</b>	configure TIMER hall sensor mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection TIMERx(x=0,2)
<b>Input parameter{in}</b>	
<b>hallmode</b>	TIMER hall sensor mode state
<i>TIMER_HALLINTERFA CE_ENABLE</i>	TIMER hall sensor mode enable
<i>TIMER_HALLINTERFA CE_DISABLE</i>	TIMER hall sensor mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER2 hall sensor mode */
```

```
timer_hall_mode_config(TIMER2, TIMER_HALLINTERFACE_ENABLE);
```

## timer\_input\_trigger\_source\_select

The description of timer\_input\_trigger\_source\_select is shown as below:

**Table 3-550. Function timer\_input\_trigger\_source\_select**

<b>Function name</b>	timer_input_trigger_source_select
<b>Function prototype</b>	void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger);
<b>Function descriptions</b>	select TIMER input trigger source
<b>Precondition</b>	SMC[2:0] = 000
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	

<b>intrigger</b>	trigger selection
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI0</i>	Internal trigger input 0
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI2</i>	Internal trigger input 2
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI3</i>	Internal trigger input 3
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOF_ED</i>	CIO edge flag
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOFE0</i>	channel 0 input Filtered output
<i>TIMER_SMCFG_TRGS</i> <i>EL_CI1FE1</i>	channel 1 input Filtered output
<i>TIMER_SMCFG_TRGS</i> <i>EL_ETIFP</i>	External trigger input filter output
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select TIMER2 input trigger source */
```

```
timer_input_trigger_source_select(TIMER2, TIMER_SMCFG_TRGSEL_ITI0);
```

### timer\_master\_output\_trigger\_source\_select

The description of timer\_master\_output\_trigger\_source\_select is shown as below:

**Table 3-551. Function timer\_master\_output\_trigger\_source\_select**

<b>Function name</b>	timer_master_output_trigger_source_select
<b>Function prototype</b>	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
<b>Function descriptions</b>	select TIMER master mode output trigger source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>outrigger</b>	master mode control
<i>TIMER_TRI_OUT_SRC</i> <i>_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 TIMER2 master mode output trigger source */
```

```
timer_master_output_trigger_source_select(TIMER2, TIMER_TRI_OUT_SRC_RESET);
```

### timer\_slave\_mode\_select

The description of timer\_slave\_mode\_select is shown as below:

**Table 3-552. Function timer\_slave\_mode\_select**

<b>Function name</b>	timer_slave_mode_select
<b>Function prototype</b>	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
<b>Function descriptions</b>	select TIMER slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>slavemode</b>	slave mode

<i>TIMER_SLAVE_MODE_DISABLE</i>	slave mode disable
<i>TIMER_QUAD_DECODER_MODE0</i>	quadrature decoder mode 0
<i>TIMER_QUAD_DECODER_MODE1</i>	quadrature decoder mode 1
<i>TIMER_QUAD_DECODER_MODE2</i>	quadrature decoder mode 2
<i>TIMER_SLAVE_MODE_RESTART</i>	restart mode
<i>TIMER_SLAVE_MODE_PAUSE</i>	pause mode
<i>TIMER_SLAVE_MODE_EVENT</i>	event mode
<i>TIMER_SLAVE_MODE_EXTERNAL0</i>	external clock mode 0
<i>TIMER_SLAVE_MODE_RESTART_EVENT</i>	Restart + event mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select TIMER2 slave mode */
```

```
timer_slave_mode_select(TIMER2, TIMER_QUAD_DECODER_MODE0);
```

### timer\_master\_slave\_mode\_config

The description of timer\_master\_slave\_mode\_config is shown as below:

**Table 3-553. 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>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection
Input parameter{in}	
<b>masterslave</b>	master slave mode state

<i>TIMER_MASTER_SLAVE_MODE_ENABLE</i>	master slave mode enable
<i>TIMER_MASTER_SLAVE_MODE_DISABLE</i>	master slave mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER2 master slave mode */
```

```
timer_master_slave_mode_config(TIMER2, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

### timer\_external\_trigger\_config

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

**Table 3-554. Function timer\_external\_trigger\_config**

<b>Function name</b>	timer_external_trigger_config
<b>Function prototype</b>	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER external trigger input
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection TIMERx(x=0,2)
<b>Input parameter{in}</b>	
<b>extprescaler</b>	external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
<b>Input parameter{in}</b>	
<b>expolarity</b>	external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
<b>Input parameter{in}</b>	



<b>extfilter</b>	external trigger filter control(0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER2 external trigger input */
```

```
timer_external_trigger_config(TIMER2,                                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 3-555. Function timer\_quadrature\_decoder\_mode\_config**

<b>Function name</b>	timer_quadrature_decoder_mode_config
<b>Function prototype</b>	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity);
<b>Function descriptions</b>	configure TIMER quadrature decoder mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection TIMERx(x=0,2)
<b>Input parameter{in}</b>	
<b>decomode</b>	quadrature decoder mode
<i>TIMER_QUAD_DECODER_MODE0</i>	counter counts on CI0FE0 edge depending on CI1FE1 level
<i>TIMER_QUAD_DECODER_MODE1</i>	counter counts on CI1FE1 edge depending on CI0FE0 level
<i>TIMER_QUAD_DECODER_MODE2</i>	counter counts on both CI0FE0 and CI1FE1 edges depending on the level of the other input
<b>Input parameter{in}</b>	
<b>ic0polarity</b>	IC0 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	capture both edge
<b>Input parameter{in}</b>	

<b>ic1polarity</b>	IC1 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	capture both edge
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER2 quadrature decoder mode */
```

```
timer_quadrature_decoder_mode_config(TIMER2, 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 3-556. Function timer\_internal\_clock\_config**

<b>Function name</b>	timer_internal_clock_config
<b>Function prototype</b>	void timer_internal_clock_config(uint32_t timer_periph);
<b>Function descriptions</b>	configure TIMER internal clock mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection TIMERx(x=0,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 internal clock mode */
```

```
timer_internal_clock_config (TIMER0);
```

### timer\_external\_clock\_mode0\_config

The description of timer\_external\_clock\_mode0\_config is shown as below:

Table 3-557. 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</i>	TIMER peripheral selection TIMERx(x=0,2,13,15,16)
<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 TIMER2 the external clock mode0 */
```

```
timer_external_clock_mode0_config(TIMER2,                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 3-558. Function timer\_external\_clock\_mode1\_config

<b>Function name</b>	timer_external_clock_mode1_config
----------------------	-----------------------------------

Function prototype	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
Function descriptions	configure TIMER the external clock mode1
Precondition	-
The called functions	timer_external_trigger_config
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	TIMER peripheral selection TIMERx(x=0,2)
Input parameter{in}	
extprescaler	ETI external trigger prescaler
TIMER_EXT_TRI_PSC_OFF	no divided
TIMER_EXT_TRI_PSC_DIV2	divided by 2
TIMER_EXT_TRI_PSC_DIV4	divided by 4
TIMER_EXT_TRI_PSC_DIV8	divided by 8
Input parameter{in}	
expolarity	ETI external trigger polarity
TIMER_ETP_FALLING	active low or falling edge active
TIMER_ETP_RISING	active high or rising edge active
Input parameter{in}	
extfilter	ETI external trigger filter control(0~15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER2 the external clock mode1 */
```

```
timer_external_clock_mode1_config(TIMER2,          TIMER_EXT_TRI_PSC_DIV2,
timer_external_clock_mode1_config(TIMER2,          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 3-559. Function timer\_external\_clock\_mode1\_disable**

<b>Function name</b>	timer_external_clock_mode1_disable
<b>Function prototype</b>	void timer_external_clock_mode1_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER the external clock mode1

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection TIMERx(x=0,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER2 the external clock mode1 */
timer_external_clock_mode1_disable(TIMER2);
```

### timer\_input\_selection\_config

The description of timer\_input\_selection\_config is shown as below:

**Table 3-560. Function timer\_input\_selection\_config**

<b>Function name</b>	timer_input_selection_config
<b>Function prototype</b>	void timer_input_selection_config(uint32_t timer_periph, uint16_t channel, uint16_t insel);
<b>Function descriptions</b>	configure TIMER input selection
<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=0,2,13,15,16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>remap</b>	remap function selection
<i>TIMER_INSEL_CHx</i>	connect to CHx
<i>TIMER_INSEL_CMPx</i>	connect to CMPx
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 input selection */
timer_input_selection_config (TIMER0, TIMER_INSEL_CHx);
```

## timer\_write\_chxval\_register\_config

The description of timer\_write\_chxval\_register\_config is shown as below:

**Table 3-561. Function timer\_write\_chxval\_register\_config**

<b>Function name</b>	timer_write_chxval_register_config
<b>Function prototype</b>	void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);
<b>Function descriptions</b>	configure TIMER write CHxVAL register selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection TIMERx(x=0,2,13,15,16)
<b>Input parameter{in}</b>	
<b>ccsel</b>	write CHxVAL register selection
<i>TIMER_CHVSEL_DISABLE</i>	no effect
<i>TIMER_CHVSEL_ENABLE</i>	when write the CHxVAL register, if the write value is same as the CHxVAL value, the write access is ignored
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER1 write CHxVAL register selection */
timer_write_chxval_register_config(TIMER1, TIMER_CHVSEL_ENABLE);
```

## timer\_output\_value\_selection\_config

The description of timer\_output\_value\_selection\_config is shown as below:

**Table 3-562. Function timer\_output\_value\_selection\_config**

<b>Function name</b>	timer_output_value_selection_config
<b>Function prototype</b>	void timer_output_value_selection_config(uint32_t timer_periph, uint16_t outsel);
<b>Function descriptions</b>	configure TIMER output value selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection

	TIMERx(x=0,2,13,15,16)
<b>Input parameter{in}</b>	
<b>ccsel</b>	write CHxVAL register selection
<i>TIMER_OUTSEL_DISA BLE</i>	no effect
<i>TIMER_OUTSEL_ENAB LE</i>	if POEN and IOS is 0, the output disabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER output value selection */
```

```
timer_output_value_selection_config (TIMER0, TIMER_OUTSEL_ENABLE);
```

### timer\_break\_external\_source\_config

The description of timer\_break\_external\_source\_config is shown as below:

**Table 3-563. Function timer\_break\_external\_source\_config**

<b>Function name</b>	timer_break_external_source_config
<b>Function prototype</b>	void timer_break_external_source_config(uint32_t timer_periph, uint16_t break_num, ControlStatus newvalue);
<b>Function descriptions</b>	configure the TIMER break source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx (x=0, 15, 16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>break_num</b>	TIMER BREAKx
<i>TIMER_BREAK0</i>	BREAK0 input signal
<i>TIMER_BREAK1</i>	BREAK1 input signal
<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 break0 source */
```

```
timer_break_external_source_config(TIMER0, TIMER_BREAK0, ENABLE);
```

### timer\_break\_external\_polarity\_config

The description of timer\_break\_external\_polarity\_config is shown as below:

**Table 3-564. Function timer\_break\_external\_polarity\_config**

<b>Function name</b>	timer_break_external_source_config
<b>Function prototype</b>	void timer_break_external_polarity_config(uint32_t timer_periph, uint16_t break_num, uint16_t bkinpolarity);
<b>Function descriptions</b>	configure TIMER break polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx (x=0,15,16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>break_num</b>	TIMER BREAKx
<i>TIMER_BREAK0</i>	BREAK0 input signal
<i>TIMER_BREAK1</i>	BREAK1 input signal
<b>Input parameter{in}</b>	
<b>bkinpolarity</b>	break polarity
<i>TIMER_BRKIN_POLARITY_LOW</i>	break polarity is high
<i>TIMER_BRKIN_POLARITY_HIGH</i>	break polarity is low
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER break polarity */
```

```
timer_break_external_polarity_config(TIMER0, TIMER_BREAK0,
TIMER_BRKIN_POLARITY_HIGH);
```

### timer\_flag\_get

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

**Table 3-565. Function timer\_flag\_get**

<b>Function name</b>	timer_flag_get
<b>Function prototype</b>	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);



<b>Function descriptions</b>	get TIMER flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>flag</b>	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag
<i>TIMER_FLAG_CH0</i>	channel 0 flag
<i>TIMER_FLAG_CH1</i>	channel 1 flag
<i>TIMER_FLAG_CH2</i>	channel 2 flag
<i>TIMER_FLAG_CH3</i>	channel 3 flag
<i>TIMER_FLAG_CMT</i>	channel commutation flag
<i>TIMER_FLAG_TRG</i>	trigger flag
<i>TIMER_FLAG_BRK0</i>	Break0 flag
<i>TIMER_FLAG_BRK1</i>	Break1 flag
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TIMER2 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get(TIMER2, TIMER_FLAG_UP);
```

### timer\_flag\_clear

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

**Table 3-566. Function timer\_flag\_clear**

<b>Function name</b>	timer_flag_clear
<b>Function prototype</b>	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
<b>Function descriptions</b>	clear TIMER flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>flag</b>	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag
<i>TIMER_FLAG_CH0</i>	channel 0 flag
<i>TIMER_FLAG_CH1</i>	channel 1 flag
<i>TIMER_FLAG_CH2</i>	channel 2 flag
<i>TIMER_FLAG_CH3</i>	channel 3 flag
<i>TIMER_FLAG_CMT</i>	channel commutation flag
<i>TIMER_FLAG_TRG</i>	trigger flag
<i>TIMER_FLAG_BRK0</i>	Break0 flag
<i>TIMER_FLAG_BRK1</i>	Break1 flag
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TIMER2 update flags */
```

```
timer_flag_clear(TIMER2, TIMER_FLAG_UP);
```

### timer\_interrupt\_enable

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

**Table 3-567. Function timer\_interrupt\_enable**

<b>Function name</b>	timer_interrupt_enable
<b>Function prototype</b>	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable the TIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	timer interrupt enable source
<i>TIMER_INT_UP</i>	update interrupt enable
<i>TIMER_INT_CH0</i>	channel 0 interrupt enable

<i>TIMER_INT_CH1</i>	channel 1 interrupt enable
<i>TIMER_INT_CH2</i>	channel 2 interrupt enable
<i>TIMER_INT_CH3</i>	channel 3 interrupt enable
<i>TIMER_INT_CMT</i>	commutation interrupt enable
<i>TIMER_INT_TRG</i>	trigger interrupt enable
<i>TIMER_INT_BRK</i>	break interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the TIMER2 update interrupt */
```

```
timer_interrupt_enable(TIMER2, TIMER_INT_UP);
```

### timer\_interrupt\_disable

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

**Table 3-568. 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 enable
<i>TIMER_INT_CH0</i>	channel 0 interrupt enable
<i>TIMER_INT_CH1</i>	channel 1 interrupt enable
<i>TIMER_INT_CH2</i>	channel 2 interrupt enable
<i>TIMER_INT_CH3</i>	channel 3 interrupt enable
<i>TIMER_INT_CMT</i>	commutation interrupt enable
<i>TIMER_INT_TRG</i>	trigger interrupt enable
<i>TIMER_INT_BRK</i>	break interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the TIMER2 update interrupt */

timer_interrupt_disable(TIMER2, TIMER_INT_UP);
```

### timer\_interrupt\_flag\_get

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

**Table 3-569. Function timer\_interrupt\_flag\_get**

Function name	timer_interrupt_flag_get
Function prototype	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t interrupt);
Function descriptions	get timer interrupt flag
Precondition	-
The called functions	-
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>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag
<i>TIMER_INT_FLAG_CM</i> <i>T</i>	commutation interrupt flag
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag
<i>TIMER_INT_FLAG_BRK</i> <i>0</i>	Break0 interrupt flag
<i>TIMER_INT_FLAG_BRK</i> <i>1</i>	Break1 interrupt flag
<i>TIMER_INT_FLAG_SYS</i> <i>B</i>	trigger interrupt flag
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TIMER2 update interrupt flag */

FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = timer_interrupt_flag_get(TIMER2, TIMER_INT_FLAG_UP);
```

### timer\_interrupt\_flag\_clear

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

**Table 3-570. Function timer\_interrupt\_flag\_clear**

<b>Function name</b>	timer_interrupt_flag_clear
<b>Function prototype</b>	void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	clear TIMER interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag
<i>TIMER_INT_FLAG_CM</i> <i>T</i>	commutation interrupt flag
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag
<i>TIMER_INT_FLAG_BRK</i> <i>0</i>	Break0 interrupt flag
<i>TIMER_INT_FLAG_BRK</i> <i>1</i>	Break1 interrupt flag
<i>TIMER_INT_FLAG_SYS</i> <i>B</i>	trigger interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TIMER2 update interrupt flag */
```

```
timer_interrupt_flag_clear(TIMER2, TIMER_INT_FLAG_UP);
```

## 3.19. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible

serial data exchange interface. The USART registers are listed in chapter [3.19.1](#), the USART firmware functions are introduced in chapter [3.19.2](#).

### 3.19.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

**Table 3-571. USART Registers**

Registers	Descriptions
USART_CTL0	Control register 0
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
USART_CHC	Coherence control register
USART_RFCS	Receive FIFO control and status register

### 3.19.2. Descriptions of Peripheral functions

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

**Table 3-572. 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

Function name	Function description
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_command_enable	enable USART command
usart_autobaud_detection_enable	enable auto baud rate detection
usart_autobaud_detection_disable	disable auto baud rate detection
usart_autobaud_detection_mode_config	configure auto baud rate detection mode
usart_address_config	configure address of the USART
usart_address_detection_mode_config	configure address detection mode
usart_mute_mode_enable	enable mute mode
usart_mute_mode_disable	disable mute mode
usart_mute_mode_wakeup_config	configure wakeup method in mute mode
usart_lin_mode_enable	enable LIN mode
usart_lin_mode_disable	disable LIN mode
usart_lin_break_detection_length_config	LIN break detection length
usart_halfduplex_enable	enable half-duplex mode
usart_halfduplex_disable	disable half-duplex mode
usart_clock_enable	enable clock
usart_clock_disable	disable 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_mode_early_nack_enable	enable early NACK in smartcard mode
usart_smartcard_mode_early_nack_disable	disable early 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

Function name	Function description
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_hardware_flow_coherence_config	configure hardware flow control coherence mode
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_receive_fifo_enable	enable receive FIFO
usart_receive_fifo_disable	disable receive FIFO
usart_receive_fifo_counter_number	read receive FIFO counter number
usart_flag_get	get flag in STAT/RFCR register
usart_flag_clear	clear USART status
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 3-573. 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_ABDE	auto baudrate detection error flag
USART_FLAG_ABDF	auto baudrate detection flag
USART_FLAG_RT	receiver timeout flag



Member name	Function description
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
USART_FLAG_EPERR	early parity error flag
USART_FLAG_RFFINT	receive FIFO full interrupt flag
USART_FLAG_RFF	receive FIFO full flag
USART_FLAG_RFE	receive FIFO empty flag

### Enum usart\_interrupt\_flag\_enum

Table 3-574. Enum usart\_interrupt\_flag\_enum

Member name	Function description
USART_INT_FLAG_EB	end of block interrupt and flag
USART_INT_FLAG_RT	receiver timeout interrupt and flag
USART_INT_FLAG_AM	address match interrupt and flag
USART_INT_FLAG_PERR	parity error interrupt and flag
USART_INT_FLAG_TBE	transmitter buffer empty interrupt and flag
USART_INT_FLAG_TC	transmission complete interrupt and flag
USART_INT_FLAG_RBNE	read data buffer not empty interrupt and flag
USART_INT_FLAG_RBNE_ORE RR	read data buffer not empty interrupt and overrun error flag
USART_INT_FLAG_IDLE	IDLE line detected interrupt and flag
USART_INT_FLAG_LBD	LIN break detected interrupt and flag
USART_INT_FLAG_WU	wakeup from deep-sleep mode interrupt and flag
USART_INT_FLAG_CTS	CTS interrupt and flag
USART_INT_FLAG_ERR_NERR	error interrupt and noise error flag
USART_INT_FLAG_ERR_ORER R	error interrupt and overrun error
USART_INT_FLAG_ERR_FERR	error interrupt and frame error flag
USART_INT_FLAG_RFF	receive FIFO full interrupt and flag

## Enum `usart_interrupt_enum`

**Table 3-575. 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
USART_INT_RFF	receive FIFO full interrupt

## Enum `usart_invert_enum`

**Table 3-576. 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 3-577. Function `usart_deinit`**

Function name	<code>usart_deinit</code>
Function prototype	<code>void usart_deinit(uint32_t usart_periph);</code>
Function descriptions	reset USART
Precondition	-
The called functions	<code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code>
<b>Input parameter{in}</b>	
<code>usart_periph</code>	usart peripheral
<code>USARTx</code>	x=0,1,2

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset USART0 */
usart_deinit(USART0);
```

### usart\_baudrate\_set

The description of usart\_baudrate\_set is shown as below:

**Table 3-578. Function usart\_baudrate\_set**

Function name	usart_baudrate_set
Function prototype	void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);
Function descriptions	configure USART baud rate value
Precondition	-
The called functions	rcu_clock_freq_get
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2
Input parameter{in}	
baudval	baud rate value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 baud rate value */
usart_baudrate_set(USART0, 115200);
```

### usart\_parity\_config

The description of usart\_parity\_config is shown as below:

**Table 3-579. Function usart\_parity\_config**

Function name	usart_parity_config
Function prototype	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
Function descriptions	configure USART parity function
Precondition	-
The called functions	-

Input parameter{in}	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 parity */
```

```
usart_parity_config(USART0, USART_PM_EVEN);
```

### usart\_word\_length\_set

The description of usart\_word\_length\_set is shown as below:

**Table 3-580. Function usart\_word\_length\_set**

<b>Function name</b>	usart_word_length_set
<b>Function prototype</b>	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);
<b>Function descriptions</b>	configure USART word length
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
Input parameter{in}	
<b>wlen</b>	USART word length configure
<i>USART_WL_8BIT</i>	8 bits
<i>USART_WL_9BIT</i>	9 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 word length */
```

```
usart_word_length_set(USART0, USART_WL_9BIT);
```

## usart\_stop\_bit\_set

The description of usart\_stop\_bit\_set is shown as below:

**Table 3-581. Function usart\_stop\_bit\_set**

<b>Function name</b>	usart_stop_bit_set
<b>Function prototype</b>	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);
<b>Function descriptions</b>	configure USART stop bit length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
<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 3-582. Function usart\_enable**

<b>Function name</b>	usart_enable
<b>Function prototype</b>	void usart_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable USART
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
<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 3-583. Function usart\_disable**

<b>Function name</b>	usart_disable
<b>Function prototype</b>	void usart_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable USART
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<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 3-584. Function usart\_transmit\_config**

<b>Function name</b>	usart_transmit_config
<b>Function prototype</b>	void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);
<b>Function descriptions</b>	configure USART transmitter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>txconfig</b>	enable or disable USART transmitter

USART_TRANSMIT_ENABLE	enable USART transmission
USART_TRANSMIT_DISABLE	disable USART transmission
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 3-585. Function usart\_receive\_config**

<b>Function name</b>	usart_receive_config
<b>Function prototype</b>	void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);
<b>Function descriptions</b>	configure USART receiver
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
Input parameter{in}	
<b>rxconfig</b>	enable or disable USART receiver
USART_RECEIVE_ENABLE	enable USART reception
USART_RECEIVE_DISABLE	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 3-586. Function usart\_data\_first\_config**

<b>Function name</b>	usart_data_first_config
<b>Function prototype</b>	void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);
<b>Function descriptions</b>	data is transmitted/received with the LSB/MSB first
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
<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 3-587. Function usart\_invert\_config**

<b>Function name</b>	usart_invert_config
<b>Function prototype</b>	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);
<b>Function descriptions</b>	configure USART inverted
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
<b>Input parameter{in}</b>	
<b>invertpara</b>	refer to <a href="#">错误!未找到引用源。</a>
USART_DINV_ENABLe	data bit level inversion



<i>USART_DINV_DISAB LE</i>	data bit level not inversion
<i>USART_TXPIN_ENAB LE</i>	TX pin level inversion
<i>USART_TXPIN_DISAB LE</i>	TX pin level not inversion
<i>USART_RXPIN_ENAB LE</i>	RX pin level inversion
<i>USART_RXPIN_DISAB LE</i>	RX pin level not inversion
<i>USART_SWAP_ENAB LE</i>	swap TX/RX pins
<i>USART_SWAP_DISAB 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\_overnrun\_enable

The description of usart\_overnrun\_enable is shown as below:

**Table 3-588. Function usart\_overnrun\_enable**

<b>Function name</b>	usart_overnrun_enable
<b>Function prototype</b>	void usart_overnrun_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable the USART overrrun function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 overrrun */
```

```
usart_overrun_enable(USART0);
```

### usart\_overrun\_disable

The description of usart\_overrun\_disable is shown as below:

**Table 3-589. Function usart\_overrun\_disable**

<b>Function name</b>	usart_overrun_disable
<b>Function prototype</b>	void usart_overrun_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable the USART overrun function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 overrun */
```

```
usart_overrun_disable(USART0);
```

### usart\_oversample\_config

The description of usart\_oversample\_config is shown as below:

**Table 3-590. Function usart\_oversample\_config**

<b>Function name</b>	usart_oversample_config
<b>Function prototype</b>	void usart_oversample_config(uint32_t usart_periph, uint32_t oversamp);
<b>Function descriptions</b>	configure the USART oversample mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
<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:

```
/* config 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 3-591. 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>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
<b>Input parameter{in}</b>	
<b>osb</b>	sample bit
USART_OSB_1BIT	1 bit
USART_OSB_3BIT	3 bits
<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 3-592. Function usart\_receiver\_timeout\_enable**

<b>Function name</b>	usart_receiver_timeout_enable
<b>Function prototype</b>	void usart_receiver_timeout_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable receiver timeout
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-593. Function usart\_receiver\_timeout\_disable**

<b>Function name</b>	usart_receiver_timeout_disable
<b>Function prototype</b>	void usart_receiver_timeout_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable receiver timeout
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-594. Function usart\_receiver\_timeout\_threshold\_config**

<b>Function name</b>	usart_receiver_timeout_threshold_config
<b>Function prototype</b>	void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);
<b>Function descriptions</b>	configure receiver timeout threshold
<b>Precondition</b>	-

The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0
Input parameter{in}	
rtimeout	receiver timeout (0x00000000-0x00FFFFFF)
Output parameter{out}	
-	-
Return value	
-	-

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 3-595. Function usart\_data\_transmit**

Function name	usart_data_transmit
Function prototype	void usart_data_transmit(uint32_t usart_periph, uint16_t data);
Function descriptions	USART transmit data function
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2
Input parameter{in}	
data	data of transmission (0x00-0x1FF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 transmit data */
```

```
usart_data_transmit(USART0, 0xAA);
```

### usart\_data\_receive

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

Table 3-596. Function `usart_data_receive`

<b>Function name</b>	<code>usart_data_receive</code>
<b>Function prototype</b>	<code>uint16_t usart_data_receive(uint32_t usart_periph);</code>
<b>Function descriptions</b>	USART receive data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	data of received (0x00-0xFF)

Example:

```

/* USART0 receive data */

uint16_t temp;

temp = usart_data_receive(USART0);

```

### **`usart_command_enable`**

The description of `usart_command_enable` is shown as below:

Table 3-597. Function `usart_command_enable`

<b>Function name</b>	<code>usart_command_enable</code>
<b>Function prototype</b>	<code>void usart_command_enable(uint32_t usart_periph, uint32_t cmdtype);</code>
<b>Function descriptions</b>	enable USART command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>cmdtype</b>	command type
<i>USART_CMD_ABD</i> <i>CM</i>	auto baudrate detection command
<i>USART_CMD_S</i> <i>BKCM</i> <i>D</i>	send break command
<i>USART_CMD_M</i> <i>MCMD</i>	mute mode command
<i>USART_CMD_R</i> <i>XFCM</i> <i>D</i>	receive data flush command
<i>USART_CMD_T</i> <i>XFCM</i>	transmit data flush request

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

Example:

```
/* enable USART0 command */
```

```
usart_command_enable(USART0, USART_CMD_SBKCMD);
```

### usart\_autobaud\_detection\_enable

The description of usart\_autobaud\_detection\_enable is shown as below:

**Table 3-598. Function usart\_autobaud\_detection\_enable**

<b>Function name</b>	usart_autobaud_detection_enable
<b>Function prototype</b>	void usart_autobaud_detection_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable auto baud rate detection
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 auto baud rate detection */
```

```
usart_autobaud_detection_enable(USART0);
```

### usart\_autobaud\_detection\_disable

The description of usart\_autobaud\_detection\_disable is shown as below:

**Table 3-599. Function usart\_autobaud\_detection\_disable**

<b>Function name</b>	usart_autobaud_detection_disable
<b>Function prototype</b>	void usart_autobaud_detection_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable auto baud rate detection
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	

<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 auto baud rate detection */
```

```
usart_autobaud_detection_disable(USART0);
```

### usart\_autobaud\_detection\_mode\_config

The description of usart\_autobaud\_detection\_mode\_config is shown as below:

**Table 3-600. Function usart\_autobaud\_detection\_mode\_config**

<b>Function name</b>	usart_autobaud_detection_mode_config
<b>Function prototype</b>	void usart_autobaud_detection_mode_config(uint32_t usart_periph, uint32_t abdmmod);
<b>Function descriptions</b>	configure auto baud rate detection mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<b>Input parameter{in}</b>	
<b>abdmmod</b>	auto baud rate detection mode
<i>USART_ABDM_FTOR</i>	falling edge to rising edge measurement
<i>USART_ABDM_FTOF</i>	falling edge to falling edge measurement
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* Configure USART0 for falling-edge to rising-edge measurement mode. */
```

```
usart_autobaud_detection_mode_config(USART0, USART_ABDM_FTOR);
```

### usart\_address\_config

The description of usart\_address\_config is shown as below:



Table 3-601. Function `usart_address_config`

<b>Function name</b>	<code>usart_address_config</code>
<b>Function prototype</b>	<code>void usart_address_config(uint32_t usart_periph, uint8_t addr);</code>
<b>Function descriptions</b>	configure the address of the USART
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>addr</b>	address of USART (0x00-0xFF)
<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 3-602. 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>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>addmod</b>	address detection mode
<i>USART_ADDDM_4BIT</i>	4 bits
<i>USART_ADDDM_FULLBIT</i>	full bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*configure address detection mode */
usart_address_config(USART0, USART_ADDDM_4BIT);
```

### usart\_mute\_mode\_enable

The description of usart\_mute\_mode\_enable is shown as below:

**Table 3-603. Function usart\_mute\_mode\_enable**

<b>Function name</b>	usart_mute_mode_enable
<b>Function prototype</b>	void usart_mute_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable mute mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<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 3-604. Function usart\_mute\_mode\_disable**

<b>Function name</b>	usart_mute_mode_disable
<b>Function prototype</b>	void usart_mute_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable mute mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<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 3-605. Function usart\_mute\_mode\_wakeup\_config**

<b>Function name</b>	usart_mute_mode_wakeup_config
<b>Function prototype</b>	void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);
<b>Function descriptions</b>	configure wakeup method in mute mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
<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 3-606. Function usart\_lin\_mode\_enable**

<b>Function name</b>	usart_lin_mode_enable
<b>Function prototype</b>	void usart_lin_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable LIN mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral

USARTx	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 LIN mode enable */
usart_lin_mode_enable(USART0);
```

### usart\_lin\_mode\_disable

The description of usart\_lin\_mode\_disable is shown as below:

**Table 3-607. Function usart\_lin\_mode\_disable**

Function name	usart_lin_mode_disable
Function prototype	void usart_lin_mode_disable(uint32_t usart_periph);
Function descriptions	disable LIN mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 LIN mode disable */
usart_lin_mode_disable(USART0);
```

### usart\_lin\_break\_dection\_length\_config

The description of usart\_lin\_break\_dection\_length\_config is shown as below:

**Table 3-608. Function usart\_lin\_break\_dection\_length\_config**

Function name	usart_lin_break_dection_length_config
Function prototype	void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lblen);
Function descriptions	LIN break detection length
Precondition	-
The called functions	-

Input parameter{in}	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
Input parameter{in}	
<b>lblen</b>	two methods be used to enter or exit the mute mode
<i>USART_LBLEN_10B</i>	10 bits
<i>USART_LBLEN_11B</i>	11 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure LIN break frame length */
```

```
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

### usart\_halfduplex\_enable

The description of usart\_halfduplex\_enable is shown as below:

**Table 3-609. Function usart\_halfduplex\_enable**

<b>Function name</b>	usart_halfduplex_enable
<b>Function prototype</b>	void usart_halfduplex_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable half-duplex mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 half duplex mode*/
```

```
usart_halfduplex_enable(USART0);
```

### usart\_halfduplex\_disable

The description of usart\_halfduplex\_disable is shown as below:

Table 3-610. Function `usart_halfduplex_disable`

Function name	<code>usart_halfduplex_disable</code>
Function prototype	<code>void usart_halfduplex_disable(uint32_t usart_periph);</code>
Function descriptions	disable half-duplex mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	usart peripheral
<code>USARTx</code>	x=0,1,2
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 3-611. Function `usart_clock_enable`

Function name	<code>usart_clock_enable</code>
Function prototype	<code>void usart_clock_enable(uint32_t usart_periph);</code>
Function descriptions	enable clock
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	usart peripheral
<code>USARTx</code>	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable clock */
usart_clock_enable(USART0);
```

### `usart_clock_disable`

The description of `usart_clock_disable` is shown as below:

Table 3-612. Function `usart_clock_disable`

<b>Function name</b>	<code>usart_clock_disable</code>
<b>Function prototype</b>	<code>void usart_clock_disable(uint32_t usart_periph);</code>
<b>Function descriptions</b>	disable clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable clock */
```

```
usart_clock_disable(USART0);
```

### **`usart_synchronous_clock_config`**

The description of `usart_synchronous_clock_config` is shown as below:

Table 3-613. Function `usart_synchronous_clock_config`

<b>Function name</b>	<code>usart_synchronous_clock_config</code>
<b>Function prototype</b>	<code>void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);</code>
<b>Function descriptions</b>	configure USART synchronous mode parameters
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>clen</b>	last bit clock pulse
<i>USART_CLEN_NONE</i>	clock pulse of the last data bit (MSB) is not output to the CK pin
<i>USART_CLEN_EN</i>	clock pulse of the last data bit (MSB) is output to the CK pin
<b>Input parameter{in}</b>	
<b>cph</b>	clock phase
<i>USART_CPH_1CK</i>	first clock transition is the first data capture edge
<i>USART_CPH_2CK</i>	second clock transition is the first data capture edge
<b>Input parameter{in}</b>	
<b>cpl</b>	clock polarity
<i>USART_CPL_LOW</i>	steady low value on CK pin

<i>USART_CPL_HIGH</i>	steady high value on CK pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 synchronous mode parameters */
```

```
usart_synchronous_clock_config(USART0,          USART_CLEN_EN,USART_CPH_2CK,
USART_CPL_HIGH);
```

### usart\_guard\_time\_config

The description of usart\_guard\_time\_config is shown as below:

**Table 3-614. 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>The called functions</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 (0x00-0x000000FF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 guard time value in smartcard mode */
```

```
usart_guard_time_config(USART0, 0x0000 0055);
```

### usart\_smartcard\_mode\_enable

The description of usart\_smartcard\_mode\_enable is shown as below:

**Table 3-615. Function usart\_smartcard\_mode\_enable**

<b>Function name</b>	usart_smartcard_mode_enable
<b>Function prototype</b>	void usart_smartcard_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable smartcard mode



<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<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 3-616. Function usart\_smartcard\_mode\_disable**

<b>Function name</b>	usart_smartcard_mode_disable
<b>Function prototype</b>	void usart_smartcard_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<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 3-617. Function usart\_smartcard\_mode\_nack\_enable**

<b>Function name</b>	usart_smartcard_mode_nack_enable
<b>Function prototype</b>	void usart_smartcard_mode_nack_enable(uint32_t usart_periph);

<b>Function descriptions</b>	enable NACK in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	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 3-618. Function usart\_smartcard\_mode\_nack\_disable**

<b>Function name</b>	usart_smartcard_mode_nack_disable
<b>Function prototype</b>	void usart_smartcard_mode_nack_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable NACK in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	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\_mode\_early\_nack\_enable

The description of usart\_smartcard\_mode\_early\_nack\_enable is shown as below:

**Table 3-619. Function usart\_smartcard\_mode\_early\_nack\_enable**

<b>Function name</b>	usart_smartcard_mode_early_nack_enable
----------------------	--

<b>Function prototype</b>	void usart_smartcard_mode_early_nack_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable early NACK in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 early NACK in smartcard mode */
usart_smartcard_mode_early_nack_enable(USART0);
```

### usart\_smartcard\_mode\_early\_nack\_disable

The description of usart\_smartcard\_mode\_early\_nack\_disable is shown as below:

**Table 3-620. Function usart\_smartcard\_mode\_early\_nack\_disable**

<b>Function name</b>	usart_smartcard_mode_early_nack_disable
<b>Function prototype</b>	void usart_smartcard_mode_early_nack_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable early NACK in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 early NACK in smartcard mode */
usart_smartcard_mode_early_nack_disable(USART0);
```

### usart\_smartcard\_autoretry\_config

The description of usart\_smartcard\_autoretry\_config is shown as below:

Table 3-621. Function `usart_smartcard_autoretry_config`

<b>Function name</b>	<code>usart_smartcard_autoretry_config</code>
<b>Function prototype</b>	<code>void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrtnum);</code>
<b>Function descriptions</b>	configure smartcard auto-retry number
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<b>Input parameter{in}</b>	
<b>scrtnum</b>	smartcard auto-retry number (0x00-0x00000007)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure smartcard auto-retry number */
usart_smartcard_autoretry_config(USART0, 0x00000007);
```

### **usart\_block\_length\_config**

The description of `usart_block_length_config` is shown as below:

Table 3-622. Function `usart_block_length_config`

<b>Function name</b>	<code>usart_block_length_config</code>
<b>Function prototype</b>	<code>void usart_block_length_config(uint32_t usart_periph, uint32_t bl);</code>
<b>Function descriptions</b>	configure block length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<b>Input parameter{in}</b>	
<b>bl</b>	block length(0x00-0x000000FF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure block length in Smartcard T=1 reception */
```

```
usart_block_length_config(USART0, 0x000000FF);
```

## usart\_irda\_mode\_enable

The description of usart\_irda\_mode\_enable is shown as below:

**Table 3-623. Function usart\_irda\_mode\_enable**

<b>Function name</b>	usart_irda_mode_enable
<b>Function prototype</b>	void usart_irda_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable IrDA mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 IrDA mode */
```

```
usart_irda_mode_enable(USART0);
```

## usart\_irda\_mode\_disable

The description of usart\_irda\_mode\_disable is shown as below:

**Table 3-624. Function usart\_irda\_mode\_disable**

<b>Function name</b>	usart_irda_mode_disable
<b>Function prototype</b>	void usart_irda_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable IrDA mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 IrDA mode */
```

```
usart_irda_mode_disable(USART0);
```

### usart\_prescaler\_config

The description of usart\_prescaler\_config is shown as below:

**Table 3-625. 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
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0
<b>Input parameter{in}</b>	
<b>psc</b>	clock prescaler (0x00-0xFF)
<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 3-626. Function usart\_irda\_lowpower\_config**

<b>Function name</b>	usart_irda_lowpower_config
<b>Function prototype</b>	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
<b>Function descriptions</b>	configure IrDA low-power
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0
<b>Input parameter{in}</b>	
<b>irlp</b>	IrDA low-power or normal
USART_IRLP_LOW	low-power
USART_IRLP_NORMAL	normal

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 IrDA low-power */
```

```
usart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

### usart\_hardware\_flow\_rts\_config

The description of usart\_hardware\_flow\_rts\_config is shown as below:

**Table 3-627. Function usart\_hardware\_flow\_rts\_config**

Function name	usart_hardware_flow_rts_config
Function prototype	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
Function descriptions	configure hardware flow control RTS
Precondition	-
The called functions	-
Input parameter{in}	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
Input parameter{in}	
<b>rtsconfig</b>	enable or disable RTS
USART_RTS_ENABLE	enable RTS
USART_RTS_DISABLE	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 3-628. Function usart\_hardware\_flow\_cts\_config**

Function name	usart_hardware_flow_cts_config
---------------	--------------------------------

<b>Function prototype</b>	void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
<b>Function descriptions</b>	configure hardware flow control CTS
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
<b>Input parameter{in}</b>	
<b>ctsconfig</b>	enable or disable CTS
USART_CTS_ENABLE	enable CTS
USART_CTS_DISABLE	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\_hardware\_flow\_coherence\_config

The description of usart\_hardware\_flow\_coherence\_config is shown as below:

**Table 3-629. Function usart\_hardware\_flow\_coherence\_config**

<b>Function name</b>	usart_hardware_flow_coherence_config
<b>Function prototype</b>	void usart_hardware_flow_coherence_config(uint32_t usart_periph, uint32_t hcm);
<b>Function descriptions</b>	configure hardware flow control coherence mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
<b>Input parameter{in}</b>	
<b>hcm</b>	Hardware flow control coherence mode
USART_HCM_NONE	nRTS signal equals to the rxne status register
USART_HCM_EN	nRTS signal is set when the last data bit has been sampled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	



-	-
---	---

Example:

```
/* configure hardware flow control coherence mode */
```

```
usart_hardware_flow_coherence_config(USART0, USART_HCM_NONE);
```

### usart\_rs485\_driver\_enable

The description of usart\_rs485\_driver\_enable is shown as below:

**Table 3-630. Function usart\_rs485\_driver\_enable**

<b>Function name</b>	usart_rs485_driver_enable
<b>Function prototype</b>	void usart_rs485_driver_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable USART RS485 driver
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* 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 3-631. 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 USART RS485 driver
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* 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 3-632. Function usart\_driver\_assertime\_config**

Function name	usart_driver_assertime_config
Function prototype	void usart_driver_assertime_config(uint32_t usart_periph, uint32_t deatime);
Function descriptions	configure driver enable assertion time
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2
Input parameter{in}	
deatime	driver enable assertion time (0x00-0x0000001F)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set USART0 driver assertime */

usart_driver_assertime_config(USART0, 0x0000001F);
```

### usart\_driver\_deassertime\_config

The description of usart\_driver\_deassertime\_config is shown as below:

**Table 3-633. Function usart\_driver\_deassertime\_config**

Function name	usart_driver_deassertime_config
Function prototype	void usart_driver_deassertime_config(uint32_t usart_periph, uint32_t dedtime);
Function descriptions	configure driver enable de-assertion time
Precondition	-
The called functions	-

Input parameter{in}	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
Input parameter{in}	
<b>dedtime</b>	driver enable de-assertion time (0x00-0x0000001F)
Output parameter{out}	
-	-
Return value	
-	-

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 3-634. 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>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
Input parameter{in}	
<b>dep</b>	DE signal
<i>USART_DEP_HIGH</i>	DE signal is active high
<i>USART_DEP_LOW</i>	DE signal is active low
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure driver enable polarity mode */
```

```
usart_depolarity_config(USART0, USART_DEP_HIGH);
```

### usart\_dma\_receive\_config

The description of usart\_dma\_receive\_config is shown as below:

Table 3-635. Function `usart_dma_receive_config`

<b>Function name</b>	<code>usart_dma_receive_config</code>
<b>Function prototype</b>	<code>void usart_dma_receive_config(uint32_t usart_periph, uint32_t dmacmd);</code>
<b>Function descriptions</b>	configure USART DMA for reception
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>dmacmd</b>	enable or disable DMA for reception
<i>USART_RECEIVE_DMA_ENABLE</i>	DMA enable for reception
<i>USART_RECEIVE_DMA_DISABLE</i>	DMA disable for reception
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 DMA enable for reception */
```

```
usart_dma_receive_config(USART0, USART_RECEIVE_DMA_ENABLE);
```

### **usart\_dma\_transmit\_config**

The description of `usart_dma_transmit_config` is shown as below:

Table 3-636. Function `usart_dma_transmit_config`

<b>Function name</b>	<code>usart_dma_transmit_config</code>
<b>Function prototype</b>	<code>void usart_dma_transmit_config(uint32_t usart_periph, uint32_t dmacmd);</code>
<b>Function descriptions</b>	configure USART DMA for transmission
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<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

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 DMA enable for transmission */
```

```
usart_dma_transmit_config(USART0, USART_TRANSMIT_DMA_ENABLE);
```

### usart\_reception\_error\_dma\_disable

The description of usart\_reception\_error\_dma\_disable is shown as below:

**Table 3-637. Function usart\_reception\_error\_dma\_disable**

Function name	usart_reception_error_dma_disable
Function prototype	void usart_reception_error_dma_disable(uint32_t usart_periph);
Function descriptions	disable DMA on reception error
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

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 3-638. Function usart\_reception\_error\_dma\_enable**

Function name	usart_reception_error_dma_enable
Function prototype	void usart_reception_error_dma_enable(uint32_t usart_periph);
Function descriptions	enable DMA on reception error
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral

USARTx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

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 3-639. Function usart\_wakeup\_enable**

Function name	usart_wakeup_enable
Function prototype	void usart_wakeup_enable(uint32_t usart_periph);
Function descriptions	enable USART to wakeup the mcu from deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 wake up enable */
usart_wakeup_enable(USART0);
```

### usart\_wakeup\_disable

The description of usart\_wakeup\_disable is shown as below:

**Table 3-640. Function usart\_wakeup\_disable**

Function name	usart_wakeup_disable
Function prototype	void usart_wakeup_disable(uint32_t usart_periph);
Function descriptions	disable USART to wakeup the mcu from deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	

<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 3-641. 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>	configure the USART wakeup mode from deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0
<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\_receive\_fifo\_enable

The description of usart\_receive\_fifo\_enable is shown as below:

Table 3-642. Function `usart_receive_fifo_enable`

Function name	<code>usart_receive_fifo_enable</code>
Function prototype	<code>void usart_receive_fifo_enable(uint32_t usart_periph);</code>
Function descriptions	enable receive FIFO
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	usart peripheral
<code>USARTx</code>	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable receive FIFO */
usart_receive_fifo_enable(USART0);
```

### `usart_receive_fifo_disable`

The description of `usart_receive_fifo_disable` is shown as below:

Table 3-643. Function `usart_receive_fifo_disable`

Function name	<code>usart_receive_fifo_disable</code>
Function prototype	<code>void usart_receive_fifo_disable(uint32_t usart_periph);</code>
Function descriptions	disable receive FIFO
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	usart peripheral
<code>USARTx</code>	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable receive FIFO */
usart_receive_fifo_disable(USART0);
```

### `usart_receive_fifo_counter_number`

The description of `usart_receive_fifo_counter_number` is shown as below:



Table 3-644. Function usart\_receive\_fifo\_counter\_number

Function name	usart_receive_fifo_counter_number
Function prototype	uint8_t usart_receive_fifo_counter_number(uint32_t usart_periph);
Function descriptions	read receive FIFO counter number
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0
Output parameter{out}	
-	-
Return value	
uint8_t	receive FIFO counter number

Example:

```
/* read receive FIFO counter number */
uint8_t temp;
temp = usart_receive_fifo_counter_number(USART0);
```

### usart\_flag\_get

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

Table 3-645. Function usart\_flag\_get

Function name	usart_flag_get
Function prototype	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
Function descriptions	get flag in STAT/RFCR register
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2
Input parameter{in}	
flag	USART flags, refer to <a href="#">错误!未找到引用源。</a> only one among these parameters can be selected
USART_FLAG_PERR	parity error flag
USART_FLAG_FERR	frame error flag
USART_FLAG_NERR	noise error flag
USART_FLAG_ORER R	overrun error
USART_FLAG_IDLE	idle line detected flag
USART_FLAG_RBNE	read data buffer not empty

USART_FLAG_TC	transmission completed
USART_FLAG_TBE	transmit data register empty
USART_FLAG_LBD	LIN break detected flag (only support USART0)
USART_FLAG_CTSF	CTS change flag
USART_FLAG_CTS	CTS level
USART_FLAG_RT	receiver timeout flag (only support USART0)
USART_FLAG_EB	end of block flag (only support USART0)
USART_FLAG_ABDE	auto baudrate detection error flag(only support USART0)
USART_FLAG_ABDF	auto baudrate detection flag(only support USART0)
USART_FLAG_BSY	busy flag
USART_FLAG_AM	address match flag
USART_FLAG_SB	send break flag
USART_FLAG_RWU	receiver wakeup from mute mode
USART_FLAG_WU	wakeup from deep-sleep mode flag (only support USART0)
USART_FLAG_TEA	transmit enable acknowledge flag
USART_FLAG_REA	receive enable acknowledge flag
USART_FLAG_EPERR	early parity error flag
USART_FLAG_RFE	receive FIFO empty flag (only support USART0)
USART_FLAG_RFF	receive FIFO full flag (only support USART0)
USART_FLAG_RFFINT	receive FIFO full interrupt flag (only support USART0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get flag USART0 state */
```

```
FlagStatus status;
```

```
status = usart_flag_get(USART0, USART_FLAG_TBE);
```

### usart\_flag\_clear

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

**Table 3-646. Function usart\_flag\_clear**

<b>Function name</b>	usart_flag_clear
<b>Function prototype</b>	void usart_flag_clear(uint32_t usart_periph, usart_flag_enum flag);
<b>Function descriptions</b>	clear flag in STAT register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral

USARTx	x=0,1,2
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags, refer to <a href="#">错误!未找到引用源。</a> only one among these parameters can be selected
USART_FLAG_PERR	parity error flag
USART_FLAG_FERR	frame error flag
USART_FLAG_NERR	noise detected flag
USART_FLAG_ORER R	overrun error flag
USART_FLAG_IDLE	idle line detected flag
USART_FLAG_TC	transmission complete flag
USART_FLAG_LBD	LIN break detected flag (only support USART0)
USART_FLAG_CTSF	CTS change flag
USART_FLAG_RT	receiver timeout flag (only support USART0)
USART_FLAG_EB	end of block flag (only support USART0)
USART_FLAG_AM	address match flag
USART_FLAG_WU	wakeup from deep-sleep mode flag (only support USART0)
USART_FLAG_EPERR	early parity error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear USART0 flag */
```

```
usart_flag_clear(USART0, USART_FLAG_TC);
```

### usart\_interrupt\_enable

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

**Table 3-647. Function usart\_interrupt\_enable**

<b>Function name</b>	usart_interrupt_enable
<b>Function prototype</b>	void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);
<b>Function descriptions</b>	enable USART interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1,2
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt type, refer to <a href="#">错误!未找到引用源。</a>

	only one among these parameters can be selected
<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 (only support USART0)
<i>USART_INT_LBD</i>	LIN break detection interrupt (only support USART0)
<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 (only support USART0)
<i>USART_INT_RFF</i>	receive FIFO full interrupt enable (only support USART0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 TBE interrupt */
```

```
usart_interrupt_enable(USART0, USART_INT_TBE);
```

### usart\_interrupt\_disable

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

**Table 3-648. Function usart\_interrupt\_disable**

<b>Function name</b>	usart_interrupt_disable
<b>Function prototype</b>	void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);
<b>Function descriptions</b>	disable USART interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt type, refer to <a href="#">错误!未找到引用源。</a> only one among these parameters can be selected
<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 (only support USART0)
<i>USART_INT_LBD</i>	LIN break detection interrupt (only support USART0)
<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 (only support USART0)
<i>USART_INT_RFF</i>	receive FIFO full interrupt enable (only support USART0)
<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 3-649. Function usart\_interrupt\_flag\_get**

<b>Function name</b>	usart_interrupt_flag_get
<b>Function prototype</b>	FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	get USART interrupt and flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag, refer to <a href="#">错误!未找到引用源。</a> , only one among these parameters can be selected
<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</i>	parity error interrupt and flag

<i>RR</i>	
<i>USART_INT_FLAG_TB E</i>	transmitter buffer empty interrupt and flag
<i>USART_INT_FLAG_TC</i>	transmission complete interrupt and flag
<i>USART_INT_FLAG_RB NE</i>	read data buffer not empty interrupt and flag
<i>USART_INT_FLAG_RB NE_ORERR</i>	read data buffer not empty interrupt and overrun error flag
<i>USART_INT_FLAG_ID LE</i>	IDLE line detected interrupt and flag
<i>USART_INT_FLAG_LB D</i>	LIN break detected interrupt and flag (only support USART0)
<i>USART_INT_FLAG_W U</i>	wakeup from deep-sleep mode interrupt and flag (only support USART0)
<i>USART_INT_FLAG_CT S</i>	CTS interrupt and flag
<i>USART_INT_FLAG_ER R_NERR</i>	error interrupt and noise error flag
<i>USART_INT_FLAG_ER R_ORERR</i>	error interrupt and overrun error
<i>USART_INT_FLAG_ER R_FERR</i>	error interrupt and frame error flag
<i>USART_INT_FLAG_RF F</i>	receive FIFO full interrupt and flag (only support USART0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the USART0 interrupt flag status */
```

```
FlagStatus status;
```

```
status = usart_interrupt_flag_get(USART0, USART_INT_FLAG_RBNE);
```

### **usart\_interrupt\_flag\_clear**

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

**Table 3-650. Function usart\_interrupt\_flag\_clear**

<b>Function name</b>	usart_interrupt_flag_clear
<b>Function prototype</b>	void usart_interrupt_flag_clear(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	clear USART interrupt flag

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag, refer to <a href="#">错误!未找到引用源。</a> , only one among these parameters can be selected
<i>USART_INT_FLAG_PERRR</i>	parity error flag
<i>USART_INT_FLAG_ERRR_FERR</i>	frame error flag
<i>USART_INT_FLAG_ERRR_NERR</i>	noise detected flag
<i>USART_INT_FLAG_RBNE_ORERR</i>	read data buffer not empty interrupt and overrun error flag
<i>USART_INT_FLAG_ERRR_ORERR</i>	error interrupt and overrun error
<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_EB</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
<i>USART_INT_FLAG_RFF</i>	receive FIFO full interrupt and 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);
```

## 3.20. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.20.1](#), the WWDGT firmware functions are introduced in chapter [3.20.2](#).

### 3.20.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

**Table 3-651. WWDGT Registers**

Registers	Descriptions
WWDGT_CTL	WWDGT control register
WWDGT_CFG	WWDGT configuration register
WWDGT_STAT	WWDGT status register

### 3.20.2. Descriptions of Peripheral functions

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

**Table 3-652. WWDGT firmware function**

Function name	Function description
wwdgt_deinit	reset the window watchdog timer configuration
wwdgt_enable	start the window watchdog timer counter
wwdgt_prescaler_value_config	configure the window watchdog prescaler value
wwdgt_window_value_config	configure the window watchdog window value
wwdgt_counter_update	configure the window watchdog timer counter value
wwdgt_config	configure counter value, window value, and prescaler divider value
wwdgt_interrupt_enable	enable early wakeup interrupt of WWDGT
wwdgt_flag_get	check early wakeup interrupt state of WWDGT
wwdgt_flag_clear	clear early wakeup interrupt state of WWDGT

#### wwdgt\_deinit

The description of wwdgt\_deinit is shown as below:

**Table 3-653. Function wwdgt\_deinit**

Function name	wwdgt_deinit
Function prototype	void wwdgt_deinit(void);
Function descriptions	reset the window watchdog timer configuration
Precondition	-
The called functions	-
Input parameter{in}	



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

Example:

```
/* reset the window watchdog timer configuration */
```

```
wwdgt_deinit();
```

### wwdgt\_enable

The description of wwdgt\_enable is shown as below:

**Table 3-654. Function wwdgt\_enable**

Function name	wwdgt_enable
Function prototype	void wwdgt_enable(void);
Function descriptions	start the window watchdog timer counter
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* start the WWDGT counter */
```

```
wwdgt_enable();
```

### wwdgt\_prescaler\_value\_config

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

**表 3-3. Function wwdgt\_prescaler\_value\_config**

Function name	wwdgt_prescaler_value_config
Function prototype	void wwdgt_prescaler_value_config(uint16_t prescaler);
Function descriptions	configure the window watchdog prescaler value
Precondition	-
The called functions	-
Input parameter{in}	

prescaler	wwdgt prescaler value
WWDGT_CFG_PS C_DIV1	the time base of WWDGT counter = (PCLK1/4096)/1
WWDGT_CFG_PS C_DIV2	the time base of WWDGT counter = (PCLK1/4096)/2
WWDGT_CFG_PS C_DIV4	the time base of WWDGT counter = (PCLK1/4096)/4
WWDGT_CFG_PS C_DIV8	the time base of WWDGT counter = (PCLK1/4096)/8
WWDGT_CFG_PS C_DIV16	the time base of WWDGT counter = (PCLK1/4096)/16
WWDGT_CFG_PS C_DIV32	the time base of WWDGT counter = (PCLK1/4096)/32
WWDGT_CFG_PS C_DIV64	the time base of WWDGT counter = (PCLK1/4096)/64
WWDGT_CFG_PS C_DIV128	the time base of WWDGT counter = (PCLK1/4096)/128
WWDGT_CFG_PS C_DIV256	the time base of WWDGT counter = (PCLK1/4096)/256
WWDGT_CFG_PS C_DIV512	the time base of WWDGT counter = (PCLK1/4096)/512
WWDGT_CFG_PS C_DIV1024	the time base of WWDGT counter = (PCLK1/4096)/1024
WWDGT_CFG_PS C_DIV2048	the time base of WWDGT counter = (PCLK1/4096)/2048
WWDGT_CFG_PS C_DIV4096	the time base of WWDGT counter = (PCLK1/4096)/4096
WWDGT_CFG_PS C_DIV8192	the time base of WWDGT counter = (PCLK1/4096)/8192
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* update WWDGT prescaler value to 8 */
```

```
wwdgt_prescaler_value_config (WWDGT_CFG_PSC_DIV8);
```

### **wwdgt\_window\_value\_config**

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

表 3-4. Function wwdgt\_window\_value\_config

<b>Function name</b>	wwdgt_window_value_config
<b>Function prototype</b>	void wwdgt_window_value_config(uint16_t window);
<b>Function descriptions</b>	configure the window watchdog window value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>window</b>	window: 0x00000000 - 0x0000007F
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* update WWDGT window value to 0x50 */
```

```
wwdgt_window_value_config (80);
```

### wwdgt\_counter\_update

The description of wwdgt\_counter\_update is shown as below:

Table 3-655. Function wwdgt\_counter\_update

<b>Function name</b>	wwdgt_counter_update
<b>Function prototype</b>	void wwdgt_counter_update(uint16_t counter_value);
<b>Function descriptions</b>	configure the window watchdog timer counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<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 3-656. Function wwdgt\_config**

<b>Function name</b>	wwdgt_config
<b>Function prototype</b>	void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler);
<b>Function descriptions</b>	configure counter value, window value, and prescaler divider value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>counter</b>	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
WWDGT_CFG_PSC_D IV1	the time base of WWDGT counter = (PCLK1/4096)/1
WWDGT_CFG_PSC_D IV2	the time base of WWDGT counter = (PCLK1/4096)/2
WWDGT_CFG_PSC_D IV4	the time base of WWDGT counter = (PCLK1/4096)/4
WWDGT_CFG_PSC_D IV8	the time base of WWDGT counter = (PCLK1/4096)/8
WWDGT_CFG_PSC_D IV16	the time base of WWDGT counter = (PCLK1/4096)/16
WWDGT_CFG_PSC_D IV32	the time base of WWDGT counter = (PCLK1/4096)/32
WWDGT_CFG_PSC_D IV64	the time base of WWDGT counter = (PCLK1/4096)/64
WWDGT_CFG_PSC_D IV128	the time base of WWDGT counter = (PCLK1/4096)/128
WWDGT_CFG_PSC_D IV256	the time base of WWDGT counter = (PCLK1/4096)/256
WWDGT_CFG_PSC_D IV512	the time base of WWDGT counter = (PCLK1/4096)/512
WWDGT_CFG_PSC_D IV1024	the time base of WWDGT counter = (PCLK1/4096)/1024
WWDGT_CFG_PSC_D IV2048	the time base of WWDGT counter = (PCLK1/4096)/2048
WWDGT_CFG_PSC_D IV4096	the time base of WWDGT counter = (PCLK1/4096)/4096
WWDGT_CFG_PSC_D	the time base of WWDGT counter = (PCLK1/4096)/8192

IV8192	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
```

```
wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

### wwdgt\_interrupt\_enable

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

**Table 3-657. Function wwdgt\_interrupt\_enable**

Function name	wwdgt_interrupt_enable
Function prototype	void wwdgt_interrupt_enable(void);
Function descriptions	enable early wakeup interrupt of WWDGT
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable early wakeup interrupt of WWDGT */
```

```
wwdgt_interrupt_enable();
```

### wwdgt\_flag\_get

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

**Table 3-658. Function wwdgt\_flag\_get**

Function name	wwdgt_flag_get
Function prototype	FlagStatus wwdgt_flag_get(void);
Function descriptions	check early wakeup interrupt state of WWDGT
Precondition	-
The called functions	-
Input parameter{in}	

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

Example:

```
/* test if the counter value update has reached the 0x40 */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get();
```

### wwdgt\_flag\_clear

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

**Table 3-659. Function wwdgt\_flag\_clear**

Function name	wwdgt_flag_clear
Function prototype	void wwdgt_flag_clear(void);
Function descriptions	clear early wakeup interrupt state of WWDGT
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear early wakeup interrupt state of WWDGT */
```

```
wwdgt_flag_clear();
```

## 4. Revision history

**Table 4-1. Revision history**

Revision No.	Description	Date
1.0	Initial Release	May.30, 2025
1.1	1. Add function interfaces to the USART peripheral 2. Modify the I2C peripheral function interface	Aug.8, 2025
1.2	1. Delete the unsupported interfaces spi_quad_io23_output_enable()/spi_quad_io23_output_disable()	Feb.5, 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.