

**GigaDevice Semiconductor Inc.**

**GD32L23x**

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

**Firmware Library  
User Guide**

Revision 2.5

(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>27</b>
<b>1.1. Rules of User Manual and Firmware Library .....</b>	<b>27</b>
1.1.1. Peripherals.....	27
1.1.2. Naming rules.....	28
<b>2. Firmware Library Overview.....</b>	<b>30</b>
<b>2.1. File Structure of Firmware Library .....</b>	<b>30</b>
2.1.1. Examples Folder.....	31
2.1.2. Firmware Folder.....	31
2.1.3. Template Folder .....	31
2.1.4. Utilities Folder .....	34
<b>2.2. File descriptions of Firmware Library .....</b>	<b>34</b>
<b>3. Firmware Library of Standard Peripherals .....</b>	<b>36</b>
<b>3.1. Overview of Firmware Library of Standard Peripherals.....</b>	<b>36</b>
<b>3.2. ADC .....</b>	<b>36</b>
3.2.1. Descriptions of Peripheral registers.....	36
3.2.2. Descriptions of Peripheral functions.....	37
<b>3.3. CAN(only for GD32L235).....</b>	<b>62</b>
3.3.1. Descriptions of Peripheral registers.....	62
3.3.2. Descriptions of Peripheral functions .....	63
<b>3.4. CAU .....</b>	<b>80</b>
3.4.1. Descriptions of Peripheral registers.....	80
3.4.2. Descriptions of Peripheral functions .....	81
<b>3.5. CMP .....</b>	<b>108</b>
3.5.1. Descriptions of Peripheral registers.....	108
3.5.2. Descriptions of Peripheral functions .....	108
<b>3.6. CRC .....</b>	<b>118</b>
3.6.1. Descriptions of Peripheral registers.....	118
3.6.2. Descriptions of Peripheral functions .....	119
<b>3.7. CTC.....</b>	<b>126</b>
3.7.1. Descriptions of Peripheral registers.....	126

3.7.2.	Descriptions of Peripheral functions .....	127
<b>3.8.</b>	<b>DBG .....</b>	<b>140</b>
3.8.1.	Descriptions of Peripheral registers .....	140
3.8.2.	Descriptions of Peripheral functions .....	140
<b>3.9.</b>	<b>DAC .....</b>	<b>144</b>
3.9.1.	Peripheral register description .....	145
3.9.2.	Descriptions of Peripheral functions .....	145
<b>3.10.</b>	<b>DMA/DMAMUX.....</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>EXTI.....</b>	<b>201</b>
3.11.1.	Descriptions of Peripheral registers .....	201
3.11.2.	Descriptions of Peripheral functions .....	201
<b>3.12.</b>	<b>FMC .....</b>	<b>210</b>
3.12.1.	Descriptions of Peripheral registers .....	210
3.12.2.	Descriptions of Peripheral functions .....	210
<b>3.13.</b>	<b>FWDGT.....</b>	<b>236</b>
3.13.1.	Descriptions of Peripheral registers .....	236
3.13.2.	Descriptions of Peripheral functions .....	236
<b>3.14.</b>	<b>GPIO.....</b>	<b>241</b>
3.14.1.	Descriptions of Peripheral registers .....	242
3.14.2.	Descriptions of Peripheral functions .....	242
<b>3.15.</b>	<b>I2C .....</b>	<b>252</b>
3.15.1.	Descriptions of Peripheral registers .....	253
3.15.2.	Descriptions of Peripheral functions .....	253
<b>3.16.</b>	<b>LPTIMER .....</b>	<b>292</b>
3.16.1.	Descriptions of Peripheral registers .....	292
3.16.2.	Descriptions of Peripheral functions .....	292
<b>3.17.</b>	<b>LPUART .....</b>	<b>313</b>
3.17.1.	Descriptions of Peripheral registers .....	313
3.17.2.	Descriptions of Peripheral functions .....	314
<b>3.18.</b>	<b>MISC.....</b>	<b>343</b>
3.18.1.	Descriptions of Peripheral registers .....	343
3.18.2.	Descriptions of Peripheral functions .....	344
<b>3.19.</b>	<b>PMU.....</b>	<b>351</b>
3.19.1.	Descriptions of Peripheral registers .....	351
3.19.2.	Descriptions of Peripheral functions .....	352
<b>3.20.</b>	<b>RCU .....</b>	<b>372</b>
3.20.1.	Descriptions of Peripheral registers .....	372

3.20.2.	Descriptions of Peripheral functions .....	372
<b>3.21.</b>	<b>RTC .....</b>	<b>406</b>
3.21.1.	Descriptions of Peripheral registers .....	406
3.21.2.	Descriptions of Peripheral functions .....	407
<b>3.22.</b>	<b>SLCD .....</b>	<b>435</b>
3.22.1.	Descriptions of Peripheral registers .....	435
3.22.2.	Descriptions of Peripheral functions .....	436
<b>3.23.</b>	<b>SPI .....</b>	<b>454</b>
3.23.1.	Descriptions of Peripheral registers .....	454
3.23.2.	Descriptions of Peripheral functions .....	454
<b>3.24.</b>	<b>SYSCFG .....</b>	<b>483</b>
3.24.1.	Descriptions of Peripheral registers .....	483
3.24.2.	Descriptions of Peripheral functions .....	483
<b>3.25.</b>	<b>TIMER .....</b>	<b>491</b>
3.25.1.	Descriptions of Peripheral registers .....	492
3.25.2.	Descriptions of Peripheral functions .....	493
<b>3.26.</b>	<b>TRNG .....</b>	<b>552</b>
3.26.1.	Descriptions of Peripheral registers .....	552
3.26.2.	Descriptions of Peripheral functions .....	552
<b>3.27.</b>	<b>USART .....</b>	<b>557</b>
3.27.1.	Descriptions of Peripheral registers .....	558
3.27.2.	Descriptions of Peripheral functions .....	558
<b>3.28.</b>	<b>VREF .....</b>	<b>606</b>
3.28.1.	Descriptions of Peripheral registers .....	606
3.28.2.	Descriptions of Peripheral functions .....	607
<b>3.29.</b>	<b>WWDGT .....</b>	<b>612</b>
3.29.1.	Descriptions of Peripheral registers .....	612
3.29.2.	Descriptions of Peripheral functions .....	612
<b>4.</b>	<b>Revision history .....</b>	<b>617</b>

## List of Figures

Figure 2-1. File structure of firmware library of GD32L23x .....	30
Figure 2-2. Select peripheral example files .....	32
Figure 2-3. Copy the peripheral example files .....	32
Figure 2-4. Open the project file .....	33
Figure 2-5. Configure project files .....	33
Figure 2-6. Compile-debug-download .....	34

## List of Tables

Table 1-1. Peripherals .....	27
Table 2-1. Function descriptions of Firmware Library .....	34
Table 3-1. Peripheral function format of Firmware Library .....	36
Table 3-2. ADC Registers .....	36
Table 3-3. ADC firmware function .....	37
Table 3-4. Function adc_deinit .....	38
Table 3-5. Function adc_enable .....	38
Table 3-6. Function adc_disable .....	39
Table 3-7. Function adc_calibration_number .....	39
Table 3-8. Function adc_calibration_enable .....	40
Table 3-9. Function adc_dma_mode_enable .....	41
Table 3-10. Function adc_dma_mode_disable .....	41
Table 3-11. Function adc_discontinuous_mode_config .....	41
Table 3-12. Function adc_special_function_config .....	42
Table 3-13. Function adc_channel_16_to_19 .....	43
Table 3-14. Function adc_data_alignment_config .....	44
Table 3-15. Function adc_channel_length_config .....	44
Table 3-16. Function adc_routine_channel_config .....	45
Table 3-17. Function adc_inserted_channel_config .....	46
Table 3-18. Function adc_inserted_channel_offset_config .....	47
Table 3-19. Function adc_channel_differential_mode_config .....	48
Table 3-20. Function adc_external_trigger_config .....	48
Table 3-21. Function adc_external_trigger_source_config .....	49
Table 3-22. Function adc_software_trigger_enable .....	50
Table 3-23. Function adc_routine_data_read .....	51
Table 3-24. Function adc_inserted_data_read .....	51
Table 3-25. Function adc_watchdog_single_channel_enable .....	52
Table 3-26. Function adc_watchdog_sequence_channel_enable .....	53
Table 3-27. Function adc_watchdog_disable .....	53
Table 3-28. Function adc_watchdog_threshold_config .....	54
Table 3-29. Function adc_resolution_config .....	54
Table 3-30. Function adc_oversample_mode_config .....	55
Table 3-31. Function adc_oversample_mode_enable .....	56
Table 3-32. Function adc_oversample_mode_disable .....	57
Table 3-33. Function adc_charge_pulse_width_counter .....	57
Table 3-34. Function adc_charge_flag_get .....	58
Table 3-35. Function adc_flag_get .....	58
Table 3-36. Function adc_flag_clear .....	59
Table 3-37. Function adc_interrupt_enable .....	60
Table 3-38. Function adc_interrupt_disable .....	60

Table 3-39. Function <code>adc_interrupt_flag_get</code> .....	61
Table 3-40. Function <code>adc_interrupt_flag_clear</code> .....	61
Table 3-41. CAN Registers .....	62
Table 3-42. CAN firmware function .....	63
Table 3-43. Structure <code>can_parameter_struct</code> .....	64
Table 3-44. Structure <code>can_transmit_message_struct</code> .....	64
Table 3-45. Structure <code>can_receive_message_struct</code> .....	64
Table 3-46. Structure <code>can_filter_parameter_struct</code> .....	65
Table 3-47. Function <code>can_deinit</code> .....	65
Table 3-48. Function <code>can_struct_para_init</code> .....	65
Table 3-49. Function <code>can_init</code> .....	66
Table 3-50. Function <code>can_filter_init</code> .....	67
Table 3-51. Function <code>can_debug_freeze_enable</code> .....	67
Table 3-52. Function <code>can_debug_freeze_disable</code> .....	68
Table 3-53. Function <code>can_time_trigger_mode_enable</code> .....	68
Table 3-54. Function <code>can_time_trigger_mode_disable</code> .....	69
Table 3-55. Function <code>can_message_transmit</code> .....	69
Table 3-56. Function <code>can_transmit_states</code> .....	70
Table 3-57. Function <code>can_transmission_stop</code> .....	70
Table 3-58. Function <code>can_message_receive</code> .....	71
Table 3-59. Function <code>can_fifo_release</code> .....	71
Table 3-60. Function <code>can_receive_message_length_get</code> .....	72
Table 3-61. Function <code>can_working_mode_set</code> .....	72
Table 3-62. Function <code>can_wakeup</code> .....	73
Table 3-63. Function <code>can_error_get</code> .....	73
Table 3-64. Function <code>can_receive_error_number_get</code> .....	74
Table 3-65. Function <code>can_transmit_error_number_get</code> .....	74
Table 3-66. Function <code>can_flag_get</code> .....	75
Table 3-67. Function <code>can_flag_clear</code> .....	76
Table 3-68. Function <code>can_interrupt_enable</code> .....	77
Table 3-69. Function <code>can_interrupt_disable</code> .....	77
Table 3-70. Function <code>can_interrupt_flag_get</code> .....	78
Table 3-71. Function <code>can_interrupt_flag_clear</code> .....	79
Table 3-72. CAU Registers .....	80
Table 3-73. CAU firmware function .....	81
Table 3-74. Structure <code>cau_key_parameter_struct</code> .....	82
Table 3-75. Structure <code>cau_iv_parameter_struct</code> .....	82
Table 3-76. Structure <code>cau_context_parameter_struct</code> .....	82
Table 3-77. Structure <code>cau_parameter_struct</code> .....	83
Table 3-78. Function <code>cau_deinit</code> .....	83
Table 3-79. Function <code>cau_struct_para_init</code> .....	83
Table 3-80. Function <code>cau_key_struct_para_init</code> .....	84
Table 3-81. Function <code>cau_iv_struct_para_init</code> .....	85
Table 3-82. Function <code>cau_context_struct_para_init</code> .....	85

Table 3-83. Function cau_enable .....	86
Table 3-84. Function cau_disable .....	86
Table 3-85. Function cau_dma_enable .....	87
Table 3-86. Function cau_dma_disable .....	87
Table 3-87. Function cau_init .....	88
Table 3-88. Function cau_aes_keysize_config .....	89
Table 3-89. Function cau_key_init .....	90
Table 3-90. Function cau_iv_init .....	90
Table 3-91. Function cau_phase_config .....	91
Table 3-92. Function cau_fifo_flush .....	92
Table 3-93. Function cau_enable_state_get .....	92
Table 3-94. Function cau_data_write .....	93
Table 3-95. Function cau_data_read .....	93
Table 3-96. Function cau_context_save .....	94
Table 3-97. Function cau_context_restore .....	95
Table 3-98. Function cau_aes_ecb .....	95
Table 3-99. Function cau_aes_cbc .....	96
Table 3-100. Function cau_aes_ctr .....	97
Table 3-101. Function cau_aes_cfb .....	98
Table 3-102. Function cau_aes_ofb .....	99
Table 3-103. Function cau_aes_gcm .....	100
Table 3-104. Function cau_aes_ccm .....	101
Table 3-105. Function cau_tdes_ecb .....	102
Table 3-106. Function cau_tdes_cbc .....	103
Table 3-107. Function cau_des_ecb .....	104
Table 3-108. Function cau_des_cbc .....	104
Table 3-109. Function cau_flag_get .....	105
Table 3-110. Function cau_interrupt_enable .....	106
Table 3-111. Function cau_interrupt_disable .....	107
Table 3-112. Function cau_interrupt_flag_get .....	107
Table 3-113. CMP Registers .....	108
Table 3-114. CMP firmware function .....	108
Table 3-115. Enum cmp_enum .....	109
Table 3-116. Function cmp_deinit .....	109
Table 3-117. Function cmp_mode_init .....	109
Table 3-118. Function cmp_noninverting_input_select .....	111
Table 3-119. Function cmp_output_init .....	111
Table 3-120. Function cmp_blanking_init .....	112
Table 3-121. Function cmp_enable .....	113
Table 3-122. Function cmp_disable .....	114
Table 3-123. Function cmp_window_enable .....	114
Table 3-124. Function cmp_window_disable .....	115
Table 3-125. Function cmp_lock_enable .....	115
Table 3-126. Function cmp_voltage_scaler_enable .....	116



Table 3-127. Function cmp_voltage_scaler_disable .....	116
Table 3-128. Function cmp_scaler_bridge_enable .....	117
Table 3-129. Function cmp_scaler_bridge_disable .....	117
Table 3-130. Function cmp_output_level_get .....	118
Table 3-131. CRC Registers .....	118
Table 3-132. CRC firmware function .....	119
Table 3-133. Function crc_deinit .....	119
Table 3-134. Function crc_reverse_output_data_enable .....	120
Table 3-135. Function crc_reverse_output_data_disable .....	120
Table 3-136. Function crc_data_register_reset .....	121
Table 3-137. Function crc_data_register_read .....	121
Table 3-138. Function crc_free_data_register_read .....	122
Table 3-139. Function crc_free_data_register_write .....	122
Table 3-140. Function crc_init_data_register_write .....	123
Table 3-141. Function crc_input_data_reverse_config .....	123
Table 3-142. Function crc_polynomial_size_set .....	124
Table 3-143. Function crc_polynomial_set .....	124
Table 3-144. Function crc_single_data_calculate .....	125
Table 3-145. Function crc_block_data_calculate .....	125
Table 3-146. CTC Registers .....	127
Table 3-147. CTC firmware function .....	127
Table 3-148. Function ctc_deinit .....	127
Table 3-149. Function ctc_counter_enable .....	128
Table 3-150. Function ctc_counter_disable .....	128
Table 3-151. Function ctc_irc48m_trim_value_config .....	129
Table 3-152. Function ctc_software_refsource_pulse_generate .....	129
Table 3-153. Function ctc_hardware_trim_mode_config .....	130
Table 3-154. Function ctc_refsource_polarity_config .....	131
Table 3-155. Function ctc_refsource_signal_select .....	131
Table 3-156. Function ctc_refsource_prescaler_config .....	132
Table 3-157. Function ctc_clock_limit_value_config .....	133
Table 3-158. Function ctc_counter_reload_value_config .....	133
Table 3-159. Function ctc_counter_capture_value_read .....	134
Table 3-160. Function ctc_counter_direction_read .....	134
Table 3-161. Function ctc_counter_reload_value_read .....	135
Table 3-162. Function ctc_irc48m_trim_value_read .....	135
Table 3-163. Function ctc_flag_get .....	136
Table 3-164. Function ctc_flag_clear .....	136
Table 3-165. Function ctc_interrupt_enable .....	137
Table 3-166. Function ctc_interrupt_disable .....	138
Table 3-167. Function ctc_interrupt_flag_get .....	138
Table 3-168. Function ctc_interrupt_flag_clear .....	139
Table 3-169. DBG Registers .....	140
Table 3-170. DBG firmware function .....	140

Table 3-171. Enum dbg_periph_enum .....	140
Table 3-172. Function dbg_deinit .....	141
Table 3-173. Function dbg_id_get .....	141
Table 3-174. Function dbg_low_power_enable.....	142
Table 3-175. Function dbg_low_power_disable.....	143
Table 3-176. Function dbg_periph_enable .....	143
Table 3-177. Function dbg_periph_disable .....	144
Table 3-178. DAC Registers .....	145
Table 3-179. DAC firmware functions .....	145
Table 3-180. Function dac_deinit.....	146
Table 3-181. Function dac_enable .....	146
Table 3-182. Function dac_disable.....	147
Table 3-183. Function dac_dma_enable.....	147
Table 3-184. Function dac_dma_disable .....	148
Table 3-185. Function dac_gpio_connect_config.....	148
Table 3-186. Function dac_output_buffer_enable .....	149
Table 3-187. Function dac_output_buffer_disable .....	150
Table 3-188. Function dac_output_value_get .....	150
Table 3-189. Function dac_data_set .....	151
Table 3-190. Function dac_trigger_enable.....	152
Table 3-191. Function dac_trigger_disable.....	152
Table 3-192. Function dac_trigger_source_config.....	153
Table 3-193. Function dac_software_trigger_enable .....	154
Table 3-194. Function dac_wave_mode_config.....	154
Table 3-195. Function dac_lfsr_noise_config .....	155
Table 3-196. Function dac_triangle_noise_config.....	156
Table 3-197. Function dac_flag_get .....	156
Table 3-198. Function dac_flag_clear .....	157
Table 3-199. Function dac_interrupt_enable .....	158
Table 3-200. Function dac_interrupt_disable .....	158
Table 3-201. Function dac_interrupt_flag_get.....	159
Table 3-202. Function dac_interrupt_flag_clear .....	159
Table 3-203. DMA Registers.....	160
Table 3-204. DMAMUX Registers .....	161
Table 3-205. DMA firmware function .....	161
Table 3-206. DMAMUX firmware function.....	162
Table 3-207. Structure dma_parameter_struct.....	163
Table 3-208. Structure dmamux_sync_parameter_struct .....	163
Table 3-209. Structure dmamux_gen_parameter_struct .....	163
Table 3-210. Enum dmamux_interrupt_enum .....	163
Table 3-211. Enum dmamux_flag_enum .....	164
Table 3-212. Enum dmamux_interrupt_flag_enum.....	165
Table 3-213. Enum dma_channel_enum .....	165
Table 3-214. Enum dmamux_multiplexer_channel_enum .....	166

Table 3-215. Enum dmamux_generator_channel_enum .....	166
Table 3-216. Function dma_deinit .....	166
Table 3-217. Function dma_struct_para_init .....	167
Table 3-218. Function dma_init .....	167
Table 3-219. Function dma_circulation_enable .....	168
Table 3-220. Function dma_circulation_disable .....	169
Table 3-221. Function dma_memory_to_memory_enable .....	169
Table 3-222. Function dma_memory_to_memory_disable .....	170
Table 3-223. Function dma_channel_enable .....	170
Table 3-224. Function dma_channel_disable .....	171
Table 3-225. Function dma_periph_address_config .....	171
Table 3-226. Function dma_memory_address_config .....	172
Table 3-227. Function dma_transfer_number_config .....	172
Table 3-228. Function dma_transfer_number_get .....	173
Table 3-229. Function dma_priority_config .....	173
Table 3-230. Function dma_memory_width_config .....	174
Table 3-231. Function dma_periph_width_config .....	175
Table 3-232. Function dma_memory_increase_enable .....	176
Table 3-233. Function dma_memory_increase_disable .....	176
Table 3-234. Function dma_periph_increase_enable .....	177
Table 3-235. Function dma_periph_increase_disable .....	177
Table 3-236. Function dma_transfer_direction_config .....	178
Table 3-237. Function dma_flag_get .....	178
Table 3-238. Function dma_flag_clear .....	179
Table 3-239. Function dma_interrupt_enable .....	180
Table 3-240. Function dma_interrupt_disable .....	180
Table 3-241. Function dma_interrupt_flag_get .....	181
Table 3-242. Function dma_interrupt_flag_clear .....	181
Table 3-243. Function dmamux_sync_struct_para_init .....	182
Table 3-244. Function dmamux_synchronization_init .....	183
Table 3-245. Function dmamux_synchronization_enable .....	184
Table 3-246. Function dmamux_synchronization_disable .....	184
Table 3-247. Function dmamux_event_generation_enable .....	185
Table 3-248. Function dmamux_event_generation_disable .....	185
Table 3-249. Function dmamux_gen_struct_para_init .....	186
Table 3-250. Function dmamux_request_generator_init .....	186
Table 3-251. Function dmamux_request_generator_channel_enable .....	187
Table 3-252. Function dmamux_request_generator_channel_disable .....	188
Table 3-253. Function dmamux_synchronization_polarity_config .....	188
Table 3-254. Function dmamux_request_forward_number_config .....	189
Table 3-255. Function dmamux_sync_id_config .....	190
Table 3-256. Function dmamux_request_id_config .....	191
Table 3-257. Function dmamux_trigger_polarity_config .....	195
Table 3-258. Function dmamux_request_generate_number_config .....	195

Table 3-259. Function <code>dmamux_trigger_id_config</code> .....	196
Table 3-260. Function <code>dmamux_flag_get</code> .....	198
Table 3-261. Function <code>dmamux_flag_clear</code> .....	198
Table 3-262. Function <code>dmamux_interrupt_enable</code> .....	199
Table 3-263. Function <code>dmamux_interrupt_disable</code> .....	199
Table 3-264. Function <code>dmamux_interrupt_flag_get</code> .....	200
Table 3-265. Function <code>dmamux_interrupt_flag_clear</code> .....	200
Table 3-266. EXTI Registers.....	201
Table 3-267. EXTI firmware function .....	201
Table 3-268. <code>exti_line_enum</code> for GD32L233xx devices .....	202
Table 3-269. <code>exti_line_enum</code> for GD32L235xx devices .....	202
Table 3-270. <code>exti_mode_enum</code> .....	203
Table 3-271. <code>exti_trig_type_enum</code> .....	203
Table 3-272. Function <code>exti_deinit</code> .....	204
Table 3-273. Function <code>exti_init</code> .....	204
Table 3-274. Function <code>exti_interrupt_enable</code> .....	205
Table 3-275. Function <code>exti_interrupt_disable</code> .....	205
Table 3-276. Function <code>exti_event_enable</code> .....	206
Table 3-277. Function <code>exti_event_disable</code> .....	206
Table 3-278. Function <code>exti_software_interrupt_enable</code> .....	207
Table 3-279. Function <code>exti_software_interrupt_disable</code> .....	207
Table 3-280. Function <code>exti_flag_get</code> .....	208
Table 3-281. Function <code>exti_flag_clear</code> .....	208
Table 3-282. Function <code>exti_interrupt_flag_get</code> .....	209
Table 3-283. Function <code>exti_interrupt_flag_clear</code> .....	209
Table 3-284. FMC Registers .....	210
Table 3-285. FMC firmware function .....	211
Table 3-286. <code>fmc_state_enum</code> .....	212
Table 3-287. <code>fmc_flag_enum</code> .....	212
Table 3-288. <code>fmc_interrupt_flag_enum</code> .....	213
Table 3-289. <code>fmc_interrupt_enum</code> .....	214
Table 3-290. Function <code>fmc_unlock</code> .....	214
Table 3-291. Function <code>fmc_lock</code> .....	214
Table 3-292. Function <code>fmc_wscnt_set</code> .....	215
Table 3-293. Function <code>fmc_prefetch_enable</code> .....	215
Table 3-294. Function <code>fmc_prefetch_disable</code> .....	216
Table 3-295. Function <code>fmc_low_power_enable</code> .....	216
Table 3-296. Function <code>fmc_low_power_disable</code> .....	217
Table 3-297. Function <code>fmc_page_erase</code> .....	217
Table 3-298. Function <code>fmc_mass_erase</code> .....	218
Table 3-299. Function <code>fmc_word_program</code> .....	218
Table 3-300. Function <code>fmc_fast_program</code> .....	219
Table 3-301. Function <code>fmc_doubleword_program</code> .....	220
Table 3-302. Function <code>ob_unlock</code> .....	221

Table 3-303. Function ob_lock .....	221
Table 3-304. Function ob_erase .....	222
Table 3-305. Function ob_write_protection_enable .....	223
Table 3-306. Function ob_security_protection_config .....	223
Table 3-307. Function ob_user_write .....	224
Table 3-308. Function ob_data_program .....	225
Table 3-309. Function ob_user_get .....	226
Table 3-310. Function ob_data_get .....	226
Table 3-311. Function ob_write_protection_get .....	227
Table 3-312. Function ob_security_protection_flag_get .....	227
Table 3-313. Function fmc_ecc_address_get .....	228
Table 3-314. Function fmc_slp_unlock .....	228
Table 3-315. Function fmc_sleep_slp_enable .....	229
Table 3-316. Function fmc_sleep_slp_disable .....	229
Table 3-317. Function fmc_run_slp_enable .....	230
Table 3-318. Function fmc_run_slp_disable .....	230
Table 3-319. Function fmc_sleep_mode_enter .....	231
Table 3-320. Function fmc_pd_mode_enter .....	231
Table 3-321. Function fmc_slp_mode_exit .....	232
Table 3-322. Function fmc_flag_get .....	232
Table 3-323. Function fmc_flag_clear .....	233
Table 3-324. Function fmc_interrupt_enable .....	234
Table 3-325. Function fmc_interrupt_disable .....	234
Table 3-326. Function fmc_interrupt_flag_get .....	235
Table 3-327. Function fmc_interrupt_flag_clear .....	235
Table 3-328. FWDGT Registers .....	236
Table 3-329. FWDGT firmware function .....	236
Table 3-330. Function fwdgt_write_enable .....	237
Table 3-331. Function fwdgt_write_disable .....	237
Table 3-332. Function fwdgt_enable .....	238
Table 3-333. Function fwdgt_prescaler_value_config .....	238
Table 3-334. Function fwdgt_reload_value_config .....	239
Table 3-335. Function fwdgt_window_value_config .....	239
Table 3-336. Function fwdgt_counter_reload .....	240
Table 3-337. Function fwdgt_config .....	240
Table 3-338. Function fwdgt_flag_get .....	241
Table 3-339. GPIO Registers .....	242
Table 3-340. GPIO firmware function .....	242
Table 3-341. Function gpio_deinit .....	243
Table 3-342. Function gpio_mode_set .....	243
Table 3-343. Function gpio_output_options_set .....	244
Table 3-344. Function gpio_bit_set .....	245
Table 3-345. Function gpio_bit_reset .....	246
Table 3-346. Function gpio_bit_write .....	246

Table 3-347. Function gpio_port_write .....	247
Table 3-348. Function gpio_input_bit_get.....	247
Table 3-349. Function gpio_input_port_get.....	248
Table 3-350. Function gpio_output_bit_get .....	249
Table 3-351. Function gpio_output_port_get .....	249
Table 3-352. Function gpio_af_set .....	250
Table 3-353. Function gpio_pin_lock .....	251
Table 3-354. Function gpio_bit_toggle .....	251
Table 3-355. Function gpio_port_toggle .....	252
Table 3-356. I2C Registers .....	253
Table 3-357. I2C firmware function.....	253
Table 3-358. i2c_interrupt_flag_enum .....	255
Table 3-359. Function i2c_deinit .....	255
Table 3-360. Function i2c_timing_config .....	256
Table 3-361. Function i2c_digital_noise_filter_config .....	257
Table 3-362. Function i2c_analog_noise_filter_enable .....	257
Table 3-363. Function i2c_analog_noise_filter_disable .....	258
Table 3-364. Function i2c_master_clock_config .....	259
Table 3-365. Function i2c_master_addressing .....	259
Table 3-366. Function i2c_address10_header_enable.....	260
Table 3-367. Function i2c_address10_header_disable.....	260
Table 3-368. Function i2c_address10_enable .....	261
Table 3-369. Function i2c_address10_disable .....	261
Table 3-370. Function i2c_automatic_end_enable .....	262
Table 3-371. Function i2c_automatic_end_disable .....	262
Table 3-372. Function i2c_slave_response_to_gcall_enable .....	263
Table 3-373. Function i2c_slave_response_to_gcall_disable .....	263
Table 3-374. Function i2c_stretch_scl_low_enable .....	264
Table 3-375. Function i2c_stretch_scl_low_disable .....	264
Table 3-376. Function i2c_address_config .....	265
Table 3-377. Function i2c_address_bit_compare_config .....	266
Table 3-378. Function i2c_address_disable .....	267
Table 3-379. Function i2c_second_address_config.....	267
Table 3-380. Function i2c_second_address_disable .....	268
Table 3-381. Function i2c_receved_address_get .....	269
Table 3-382. Function i2c_slave_byte_control_enable.....	269
Table 3-383. Function i2c_slave_byte_control_disable.....	270
Table 3-384. Function i2c_nack_enable .....	270
Table 3-386. Function i2c_wakeup_from_deepsleep_enable.....	271
Table 3-387. Function i2c_wakeup_from_deepsleep_disable.....	271
Table 3-388. Function i2c_enable .....	272
Table 3-389. Function i2c_disable .....	272
Table 3-390. Function i2c_start_on_bus .....	273
Table 3-391. Function i2c_stop_on_bus.....	273

Table 3-392. Function i2c_data_transmit .....	274
Table 3-393. Function i2c_data_receive .....	274
Table 3-394. Function i2c_reload_enable.....	275
Table 3-395. Function i2c_reload_disable.....	275
Table 3-396. Function i2c_transfer_byte_number_config.....	276
Table 3-397. Function i2c_dma_enable .....	276
Table 3-398. Function i2c_dma_disable .....	277
Table 3-399. Function i2c_pec_transfer .....	278
Table 3-400. Function i2c_pec_enable.....	278
Table 3-401. Function i2c_pec_disable.....	279
Table 3-402. Function i2c_pec_value_get .....	279
Table 3-403. Function i2c_smbus_alert_enable.....	280
Table 3-404. Function i2c_smbus_alert_disable.....	280
Table 3-405. Function i2c_smbus_default_addr_enable .....	281
Table 3-406. Function i2c_smbus_default_addr_disable .....	281
Table 3-407. Function i2c_smbus_host_addr_enable .....	282
Table 3-408. Function i2c_smbus_host_addr_disable .....	282
Table 3-409. Function i2c_extented_clock_timeout_enable.....	283
Table 3-410. Function i2c_extented_clock_timeout_disable.....	283
Table 3-411. Function i2c_clock_timeout_enable.....	284
Table 3-412. Function i2c_clock_timeout_disable.....	284
Table 3-413. Function i2c_bus_timeout_b_config.....	285
Table 3-414. Function i2c_bus_timeout_a_config .....	285
Table 3-415. Function i2c_idle_clock_timeout_config.....	286
Table 3-416. Function i2c_flag_get.....	286
Table 3-417. Function i2c_flag_clear.....	287
Table 3-418. Function i2c_interrupt_enable.....	288
Table 3-419. Function i2c_interrupt_disable .....	289
Table 3-420. Function i2c_interrupt_flag_get.....	290
Table 3-421. Function i2c_interrupt_flag_clear.....	291
Table 3-422. LPTIMER Registers .....	292
Table 3-423. LPTIMER firmware function .....	292
Table 3-424. Structure lptimer_parameter_struct.....	293
Table 3-425. Function lptimer_deinit.....	294
Table 3-426. Function lptimer_struct_para_init .....	294
Table 3-427. Function lptimer_init .....	295
Table 3-428. Function lptimer_inputremap .....	296
Table 3-429. Function lptimer_register_shadow_enable .....	297
Table 3-430. Function lptimer_register_shadow_disable .....	297
Table 3-431. Function lptimer_timeout_enable.....	298
Table 3-432. Function lptimer_timeout_disable.....	298
Table 3-433. Function lptimer_countinue_start .....	299
Table 3-434. Function lptimer_single_start .....	300
Table 3-435. Function lptimer_stop.....	300



Table 3-436. Function <code>lptimer_counter_read</code> .....	301
Table 3-437. Function <code>lptimer_autoreload_read</code> .....	301
Table 3-438. Function <code>lptimer_compare_read</code> .....	302
Table 3-439. Function <code>lptimer_autoreload_value_config</code> .....	303
Table 3-440. Function <code>lptimer_compare_value_config</code> .....	303
Table 3-441. Function <code>lptimer_decodemode0_enable</code> .....	304
Table 3-442. Function <code>lptimer_decodemode1_enable</code> .....	304
Table 3-443. Function <code>lptimer_decodemode_disable</code> .....	305
Table 3-444. Function <code>lptimer_highlevelcounter_enable</code> .....	305
Table 3-445. Function <code>lptimer_highlevelcounter_disable</code> .....	306
Table 3-446. Function <code>lptimer_flag_get</code> .....	307
Table 3-447. Function <code>lptimer_flag_clear</code> .....	308
Table 3-448. Function <code>lptimer_interrupt_enable</code> .....	309
Table 3-449. Function <code>lptimer_interrupt_disable</code> .....	310
Table 3-450. Function <code>lptimer_interrupt_flag_get</code> .....	311
Table 3-451. Function <code>lptimer_interrupt_flag_clear</code> .....	312
Table 3-452. LPUART Registers .....	313
Table 3-453. LPUART firmware function .....	314
Table 3-454. Enum <code>lpuart_flag_enum</code> .....	315
Table 3-455. Enum <code>lpuart_interrupt_flag_enum</code> .....	315
Table 3-456. Enum <code>lpuart_interrupt_enum</code> .....	316
Table 3-457. Enum <code>lpuart_invert_enum</code> .....	316
Table 3-458. Function <code>lpuart_deinit</code> .....	317
Table 3-459. Function <code>lpuart_baudrate_set</code> .....	317
Table 3-460. Function <code>lpuart_parity_config</code> .....	318
Table 3-461. Function <code>lpuart_word_length_set</code> .....	318
Table 3-462. Function <code>lpuart_stop_bit_set</code> .....	319
Table 3-463. Function <code>lpuart_enable</code> .....	319
Table 3-464. Function <code>lpuart_disable</code> .....	320
Table 3-465. Function <code>lpuart_transmit_config</code> .....	320
Table 3-466. Function <code>lpuart_receive_config</code> .....	321
Table 3-467. Function <code>lpuart_data_first_config</code> .....	322
Table 3-468. Function <code>lpuart_invert_config</code> .....	322
Table 3-469. Function <code>lpuart_overshoot_enable</code> .....	323
Table 3-470. Function <code>lpuart_overshoot_disable</code> .....	323
Table 3-471. Function <code>lpuart_data_transmit</code> .....	324
Table 3-472. Function <code>lpuart_data_receive</code> .....	325
Table 3-473. Function <code>lpuart_command_enable</code> .....	325
Table 3-474. Function <code>lpuart_address_config</code> .....	326
Table 3-475. Function <code>lpuart_address_detection_mode_config</code> .....	326
Table 3-476. Function <code>lpuart_mute_mode_enable</code> .....	327
Table 3-477. Function <code>lpuart_mute_mode_disable</code> .....	327
Table 3-478. Function <code>lpuart_mute_mode_wakeup_config</code> .....	328
Table 3-479. Function <code>lpuart_halfduplex_enable</code> .....	329



Table 3-480. Function lpuart_halfduplex_disable.....	329
Table 3-481. Function lpuart_hardware_flow_rts_config.....	330
Table 3-482. Function lpuart_hardware_flow_cts_config.....	330
Table 3-483. Function lpuart_hardware_flow_coherence_config.....	331
Table 3-484. Function lpuart_rs485_driver_enable.....	332
Table 3-485. Function lpuart_rs485_driver_disable.....	332
Table 3-486. Function lpuart_driver_asserttime_config.....	333
Table 3-487. Function lpuart_driver_deasserttime_config.....	333
Table 3-488. Function lpuart_depolarity_config.....	334
Table 3-489. Function lpuart_dma_receive_config.....	334
Table 3-490. Function lpuart_dma_transmit_config.....	335
Table 3-491. Function lpuart_reception_error_dma_disable.....	336
Table 3-492. Function lpuart_reception_error_dma_enable.....	336
Table 3-493. Function lpuart_wakeup_enable.....	337
Table 3-494. Function lpuart_wakeup_disable.....	337
Table 3-495. Function lpuart_wakeup_mode_config.....	338
Table 3-496. Function lpuart_flag_get.....	338
Table 3-497. Function lpuart_flag_clear.....	339
Table 3-498. Function lpuart_interrupt_enable.....	340
Table 3-499. Function lpuart_interrupt_disable.....	340
Table 3-500. Function lpuart_interrupt_flag_get.....	341
Table 3-501. Function lpuart_interrupt_flag_clear.....	342
Table 3-502. NVIC Registers.....	343
Table 3-503. SysTick Registers.....	343
Table 3-504. MISC firmware function.....	344
Table 3-505. IRQn_Type for GD32L233xx devices.....	344
Table 3-506. IRQn_Type for GD32L235xx devices.....	345
Table 3-507. Function nvic_irq_enable.....	347
Table 3-508. Function nvic_irq_disable.....	348
Table 3-509. Function nvic_system_reset.....	348
Table 3-510. Function nvic_vector_table_set.....	349
Table 3-511. Function system_lowpower_set.....	349
Table 3-512. Function system_lowpower_reset.....	350
Table 3-513. Function systick_clksource_set.....	351
Table 3-514. PMU Registers.....	351
Table 3-515. PMU firmware function.....	352
Table 3-516. Function pmu_deinit.....	353
Table 3-517. Function pmu_lvd_select.....	353
Table 3-518. Function pmu_lvd_disable.....	354
Table 3-519. Function pmu_ido_output_select.....	354
Table 3-520. Function pmu_vc_enable.....	355
Table 3-521. Function pmu_vc_disable.....	356
Table 3-522. Function pmu_vcr_select.....	356
Table 3-523. Function pmu_low_power_enable.....	357

Table 3-524. Function pmu_low_power_disable.....	357
Table 3-525. Function pmu_to_sleepmode.....	358
Table 3-526. Function pmu_to_deepsleepmode.....	358
Table 3-527. Function pmu_to_standbymode.....	359
Table 3-528. Function pmu_wakeup_pin_enable.....	360
Table 3-529. Function pmu_wakeup_pin_disable.....	360
Table 3-530. Function pmu_backup_write_enable.....	361
Table 3-531. Function pmu_backup_write_disable.....	361
Table 3-532. Function pmu_sram_power_config.....	362
Table 3-533. Function pmu_core1_power_config.....	362
Table 3-534. Function pmu_deepsleep2_retention_enable.....	363
Table 3-535. Function pmu_deepsleep2_retention_disable.....	363
Table 3-536. Function pmu_eflash_sleep_power_config.....	364
Table 3-537. Function pmu_eflash_deepsleep_power_config.....	364
Table 3-538. Function pmu_deepsleep2_sram_power_config.....	365
Table 3-539. Function pmu_deepsleep_wait_time_config.....	366
Table 3-540. Function pmu_wakeuptime_core1_software_enable.....	366
Table 3-541. Function pmu_wakeuptime_core1_software_disable.....	367
Table 3-542. Function pmu_wakeuptime_eflash_config.....	367
Table 3-543. Function pmu_wakeuptime_sram_config.....	368
Table 3-544. Function pmu_wakeuptime_sram_software_enable.....	368
Table 3-545. Function pmu_wakeuptime_sram_software_disable.....	369
Table 3-546. Function pmu_wakeuptime_deepsleep2_software_enable.....	369
Table 3-547. Function pmu_wakeuptime_deepsleep2_software_disable.....	370
Table 3-548. Function pmu_flag_get.....	370
Table 3-549. Function pmu_flag_clear.....	371
Table 3-550. RCU Registers.....	372
Table 3-551. RCU firmware function.....	372
Table 3-552. Enum rcu_periph_enum.....	374
Table 3-553. Enum rcu_periph_sleep_enum.....	375
Table 3-554. Enum rcu_periph_reset_enum.....	375
Table 3-555. Enum rcu_flag_enum.....	376
Table 3-556. Enum rcu_int_flag_enum.....	377
Table 3-557. Enum rcu_int_flag_clear_enum.....	377
Table 3-558. Enum rcu_int_enum.....	378
Table 3-559. Enum rcu_osci_type_enum.....	378
Table 3-560. Enum rcu_clock_freq_enum.....	378
Table 3-561. Enum usart_idx_enum.....	379
Table 3-562. Enum lptimer_idx_enum.....	379
Table 3-563. Enum lpuart_idx_enum.....	379
Table 3-564. Enum i2c_idx_enum.....	379
Table 3-565. Function rcu_deinit.....	379
Table 3-566. Function rcu_periph_clock_enable.....	380
Table 3-567. Function rcu_periph_clock_disable.....	380

Table 3-568. Function rcu_periph_clock_sleep_enable .....	381
Table 3-569. Function rcu_periph_clock_sleep_disable .....	381
Table 3-570. Function rcu_periph_reset_enable.....	382
Table 3-571. Function rcu_periph_reset_disable .....	382
Table 3-572. Function rcu_bkp_reset_enable .....	383
Table 3-573. Function rcu_bkp_reset_disable .....	383
Table 3-574. Function rcu_system_clock_source_config.....	384
Table 3-575. Function rcu_system_clock_source_get .....	384
Table 3-576. Function rcu_ahb_clock_config .....	385
Table 3-577. Function rcu_apb1_clock_config .....	385
Table 3-578. Function rcu_apb2_clock_config .....	386
Table 3-579. Function rcu_adc_clock_config.....	387
Table 3-580. Function rcu_ckout_config.....	388
Table 3-581. Function rcu_pll_config .....	389
Table 3-582. Function rcu_usart_clock_config.....	389
Table 3-583. Function rcu_i2c_clock_config .....	390
Table 3-584. Function rcu_lptimer_clock_config .....	391
Table 3-585. Function rcu_lptimer_clock_config .....	391
Table 3-586. Function rcu_lpuart_clock_config.....	392
Table 3-587. Function rcu_lpuart_clock_config.....	393
Table 3-588. Function rcu_irc16mdiv_clock_config .....	394
Table 3-589. Function rcu_usbd_clock_config .....	394
Table 3-590. Function rcu_rtc_clock_config .....	395
Table 3-591. Function rcu_pll_source_ck_prediv_config .....	395
Table 3-592. Function rcu_lxtal_drive_capability_config.....	396
Table 3-593. Function rcu_lp_bandgap_config.....	396
Table 3-594. Function rcu_osci_stab_wait .....	397
Table 3-595. Function rcu_osci_on .....	398
Table 3-596. Function rcu_osci_off.....	398
Table 3-597. Function rcu_osci_bypass_mode_enable .....	399
Table 3-598. Function rcu_osci_bypass_mode_disable .....	399
Table 3-599. Function rcu_irc16m_adjust_value_set.....	400
Table 3-600. Function rcu_hxtal_clock_monitor_enable .....	400
Table 3-601. Function rcu_hxtal_clock_monitor_disable .....	401
Table 3-602. Function rcu_lxtal_clock_monitor_enable.....	401
Table 3-603. Function rcu_lxtal_clock_monitor_disable.....	402
Table 3-604. Function rcu_voltage_key_unlock .....	402
Table 3-605. Function rcu_clock_freq_get.....	403
Table 3-606. Function rcu_flag_get.....	403
Table 3-607. Function rcu_all_reset_flag_clear .....	404
Table 3-608. Function rcu_interrupt_flag_get .....	404
Table 3-609. Function rcu_interrupt_flag_clear .....	405
Table 3-610. Function rcu_interrupt_enable.....	405
Table 3-611. Function rcu_interrupt_disable .....	406

Table 3-612. RTC Registers .....	406
Table 3-613. RTC firmware function.....	407
Table 3-614. Structure rtc_parameter_struct.....	408
Table 3-615. Structure rtc_alarm_struct.....	409
Table 3-616. Structure rtc_timestamp_struct.....	409
Table 3-617. Structure rtc_tamper_struct .....	409
Table 3-618. Function rtc_deinit .....	410
Table 3-619. Function rtc_init.....	410
Table 3-620. Function rtc_init_mode_enter .....	411
Table 3-621. Function rtc_init_mode_exit.....	411
Table 3-622. Function rtc_register_sync_wait .....	412
Table 3-623. Function rtc_current_time_get.....	413
Table 3-624. Function rtc_subsecond_get.....	413
Table 3-625. Function rtc_alarm_config.....	414
Table 3-626. Function rtc_alarm_subsecond_config.....	414
Table 3-627. Function rtc_alarm_enable .....	416
Table 3-628. Function rtc_alarm_disable .....	416
Table 3-629. Function rtc_alarm_get.....	417
Table 3-630. Function rtc_alarm_subsecond_get .....	417
Table 3-631. Function rtc_timestamp_enable .....	418
Table 3-632. Function rtc_timestamp_disable .....	418
Table 3-633. Function rtc_timestamp_internalevent_config .....	419
Table 3-634. Function rtc_timestamp_get.....	419
Table 3-635. Function rtc_timestamp_subsecond_get.....	420
Table 3-636. Function rtc_timestamp_enable .....	420
Table 3-637. Function rtc_tamper_disable.....	421
Table 3-638. Function rtc_tamper_mask .....	421
Table 3-639. Function rtc_tamper_without_bkp_reset .....	422
Table 3-640. Function rtc_output_pin_select.....	423
Table 3-641. Function rtc_alarm_output_config .....	423
Table 3-642. Function rtc_calibration_output_config .....	424
Table 3-643. Function rtc_hour_adjust.....	425
Table 3-644. Function rtc_second_adjust.....	425
Table 3-645. Function rtc_bypass_shadow_enable .....	426
Table 3-646. Function rtc_bypass_shadow_disable .....	426
Table 3-647. Function rtc_refclock_detection_enable .....	427
Table 3-648. Function rtc_refclock_detection_disable.....	427
Table 3-649. Function rtc_wakeup_enable .....	428
Table 3-650. Function rtc_wakeup_disable .....	428
Table 3-651. Function rtc_wakeup_clock_set .....	429
Table 3-652. Function rtc_wakeup_timer_set.....	429
Table 3-653. Function rtc_wakeup_timer_get .....	430
Table 3-654. Function rtc_smooth_calibration_config .....	430
Table 3-655. Function rtc_interrupt_enable .....	431

Table 3-656. Function rtc_interrupt_disable.....	432
Table 3-657. Function rtc_flag_get.....	433
Table 3-658. Function rtc_flag_clear.....	434
Table 3-645. Function rtc_lxtal_stab_reset_enable.....	434
Table 3-646. Function rtc_lxtal_stab_reset_disable.....	435
Table 3-659. SLCD Registers .....	435
Table 3-660. SLCD firmware function .....	436
Table 3-661. Enum slcd_data_register_enum .....	437
Table 3-662. Function slcd_deinit.....	437
Table 3-663. Function slcd_enable.....	437
Table 3-664. Function slcd_disable.....	438
Table 3-665. Function slcd_init .....	438
Table 3-666. Function slcd_enhance_mode_enable .....	440
Table 3-667. Function slcd_enhance_mode_disable .....	441
Table 3-668. Function slcd_weak_driving_resistance_select .....	441
Table 3-669. Function slcd_bias_voltage_select .....	442
Table 3-670. Function slcd_duty_select.....	442
Table 3-671. Function slcd_clock_config.....	443
Table 3-672. Function slcd_blink_mode_config .....	444
Table 3-673. Function slcd_contrast_ratio_config .....	445
Table 3-674. Function slcd_dead_time_config .....	446
Table 3-675. Function slcd_pulse_on_duration_config .....	447
Table 3-676. Function slcd_com_seg_remap.....	448
Table 3-677. Function slcd_voltage_source_select .....	448
Table 3-678. Function slcd_high_drive_config .....	449
Table 3-679. Function slcd_data_register_write .....	450
Table 3-680. Function slcd_data_update_request.....	450
Table 3-681. Function slcd_flag_get .....	451
Table 3-682. Function slcd_flag_clear .....	451
Table 3-683. Function slcd_interrupt_enable .....	452
Table 3-684. Function slcd_interrupt_disable .....	452
Table 3-685. Function slcd_interrupt_flag_get .....	453
Table 3-686. Function slcd_interrupt_flag_clear .....	453
Table 3-687. SPI/I2S Registers .....	454
Table 3-688. SPI/I2S firmware function.....	454
Table 3-689. spi_parameter_struct.....	455
Table 3-690. Function spi_i2s_deinit .....	456
Table 3-691. Function spi_struct_para_init .....	457
Table 3-692. Function spi_init .....	457
Table 3-693. Function spi_enable.....	458
Table 3-694. Function spi_disable.....	459
Table 3-695. Function i2s_init .....	459
Table 3-696. Function i2s_psc_config .....	460
Table 3-697. Function i2s_enable .....	461

Table 3-698. Function i2s_disable .....	462
Table 3-699. Function spi_nss_output_enable .....	463
Table 3-700. Function spi_nss_output_disable .....	463
Table 3-701. Function spi_nss_internal_high .....	464
Table 3-702. Function spi_nss_internal_low .....	464
Table 3-703. Function spi_dma_enable .....	465
Table 3-704. Function spi_dma_disable .....	465
Table 3-705. Function spi_transmit_odd_config .....	466
Table 3-706. Function spi_receive_odd_config .....	466
Table 3-707. Function spi_i2s_data_frame_format_config .....	467
Table 3-708. Function spi_fifo_access_size_config .....	468
Table 3-709. Function spi_bidirectional_transfer_config .....	468
Table 3-710. Function spi_i2s_data_transmit .....	469
Table 3-711. Function spi_i2s_data_receive .....	469
Table 3-712. Function spi_crc_polynomial_set .....	470
Table 3-713. Function spi_crc_polynomial_get .....	471
Table 3-714. Function spi_crc_length_set .....	471
Table 3-715. Function spi_crc_on .....	472
Table 3-716. Function spi_crc_off .....	472
Table 3-717. Function spi_crc_next .....	473
Table 3-718. Function spi_crc_get .....	473
Table 3-719. Function spi_crc_error_clear .....	474
Table 3-720. Function spi_ti_mode_enable .....	474
Table 3-721. Function spi_ti_mode_disable .....	475
Table 3-722. Function spi_nssp_mode_enable .....	475
Table 3-723. Function spi_nssp_mode_disable .....	476
Table 3-724. Function spi_quad_enable .....	476
Table 3-725. Function spi_quad_disable .....	477
Table 3-726. Function spi_quad_write_enable .....	478
Table 3-727. Function spi_quad_read_enable .....	478
Table 3-730. Function spi_i2s_format_error_clear .....	479
Table 3-731. Function spi_i2s_flag_get .....	479
Table 3-732. Function spi_i2s_interrupt_enable .....	480
Table 3-733. Function spi_i2s_interrupt_disable .....	481
Table 3-734. Function spi_i2s_interrupt_flag_get .....	482
Table 3-735. SYSCFG Registers .....	483
Table 3-736. SYSCFG firmware function .....	483
Table 3-737. Function syscfg_deinit .....	484
Table 3-738. Function syscfg_exti_line_config .....	484
Table 3-739. Function syscfg_pin_remap_enable .....	485
Table 3-740. Function syscfg_pin_remap_disable .....	485
Table 3-741. Function syscfg_high_current_enable .....	486
Table 3-742. Function syscfg_high_current_disable .....	487
Table 3-743. Function irq_latency_set .....	487

Table 3-744. Function syscfg_bootmode_get .....	488
Table 3-745. Function syscfg_sram_waitstate_insert .....	489
Table 3-746. Function syscfg_sram_waitstate_cancel .....	489
Table 3-747. Function syscfg_lock_config .....	490
Table 3-748. Function syscfg_flag_get.....	490
Table 3-749. Function syscfg_flag_clear.....	491
Table 3-750. TIMERx Registers .....	492
Table 3-751. TIMERx firmware function.....	493
Table 3-752. Structure timer_parameter_struct .....	495
Table 3-753. Structure timer_break_parameter_struct.....	495
Table 3-754. Structure timer_oc_parameter_struct.....	496
Table 3-755. Structure timer_ic_parameter_struct.....	496
Table 3-756. Function timer_deinit.....	496
Table 3-757. Function timer_struct_para_init.....	497
Table 3-758. Function timer_init .....	497
Table 3-759. Function timer_enable .....	498
Table 3-760. Function timer_disable .....	499
Table 3-761. Function timer_auto_reload_shadow_enable .....	499
Table 3-762. Function timer_auto_reload_shadow_disable .....	500
Table 3-763. Function timer_update_event_enable .....	500
Table 3-764. Function timer_update_event_disable .....	501
Table 3-765. Function timer_counter_alignment .....	501
Table 3-766. Function timer_counter_up_direction .....	502
Table 3-767. timer_counter_down_direction .....	503
Table 3-768. Function timer_prescaler_config.....	503
Table 3-769. Function timer_repetition_value_config .....	504
Table 3-770. Function timer_autoreload_value_config .....	505
Table 3-771. Function timer_counter_value_config.....	505
Table 3-772. Function timer_counter_read .....	506
Table 3-773. Function timer_prescaler_read .....	507
Table 3-774. Function timer_single_pulse_mode_config .....	507
Table 3-775. Function timer_update_source_config.....	508
Table 3-776. Function timer_dma_enable .....	508
Table 3-777. Function timer_dma_disable .....	509
Table 3-778. Function timer_channel_dma_request_source_select.....	510
Table 3-779. Function timer_dma_transfer_config.....	511
Table 3-780. Function timer_event_software_generate.....	513
Table 3-781. Function timer_break_struct_para_init .....	514
Table 3-782. Function timer_break_config .....	514
Table 3-783. Function timer_break_enable.....	515
Table 3-784. Function timer_break_disable .....	516
Table 3-785. Function timer_automatic_output_enable .....	516
Table 3-786. Function timer_automatic_output_disable .....	517
Table 3-787. Function timer_primary_output_config.....	517



Table 3-788. Function timer_channel_control_shadow_config .....	518
Table 3-789. Function timer_channel_control_shadow_update_config.....	518
Table 3-790. Function timer_channel_output_struct_para_init .....	519
Table 3-791. Function timer_channel_output_config .....	520
Table 3-792. Function timer_channel_output_mode_config .....	521
Table 3-793. Function timer_channel_output_pulse_value_config.....	522
Table 3-794. Function timer_channel_output_shadow_config .....	523
Table 3-795. Function timer_channel_output_fast_config.....	524
Table 3-796. Function timer_channel_output_clear_config .....	524
Table 3-797. Function timer_channel_output_polarity_config.....	525
Table 3-798. Function timer_channel_complementary_output_polarity_config .....	526
Table 3-799. Function timer_channel_output_state_config .....	527
Table 3-800. Function timer_channel_complementary_output_state_config .....	528
Table 3-801. Function timer_channel_input_struct_para_init.....	529
Table 3-802. Function timer_input_capture_config.....	529
Table 3-803. Function timer_channel_input_capture_prescaler_config .....	530
Table 3-804. Function timer_channel_capture_value_register_read .....	531
Table 3-805. Function timer_input_pwm_capture_config .....	532
Table 3-806. Function timer_hall_mode_config.....	533
Table 3-807. Function timer_input_trigger_source_select .....	534
Table 3-808. Function timer_master_output_trigger_source_select .....	535
Table 3-809. Function timer_slave_mode_select .....	536
Table 3-810. Function timer_master_slave_mode_config .....	537
Table 3-811. Function timer_external_trigger_config .....	537
Table 3-812. Function timer_quadrature_decoder_mode_config.....	538
Table 3-813. Function timer_internal_clock_config .....	539
Table 3-814. Function timer_internal_trigger_as_external_clock_config .....	540
Table 3-815. Function timer_external_trigger_as_external_clock_config .....	541
Table 3-816. Function timer_external_clock_mode0_config .....	542
Table 3-817. Function timer_external_clock_mode1_config .....	543
Table 3-818. Function timer_external_clock_mode1_disable .....	544
Table 3-819. Function timer_channel_remap_config.....	544
Table 3-820. Function timer_write_chxval_register_config.....	545
Table 3-821. Function timer_flag_get .....	546
Table 3-822. Function timer_flag_clear .....	547
Table 3-823. Function timer_interrupt_enable .....	548
Table 3-824. Function timer_interrupt_disable .....	549
Table 3-825. Function timer_interrupt_flag_get.....	550
Table 3-826. Function timer_interrupt_flag_clear.....	551
Table 3-827. TRNG Registers .....	552
Table 3-828. TRNG firmware function .....	552
Table 3-829. Enum trng_flag_enum .....	553
Table 3-830. Enum trng_int_flag_enum.....	553
Table 3-831. Function trng_deinit.....	553



Table 3-832. Function <code>trng_enable</code> .....	553
Table 3-833 Function <code>trng_disable</code> .....	554
Table 3-834 Function <code>trng_get_true_random_data</code> .....	554
Table 3-835 <code>trng_flag_get</code> .....	555
Table 3-836 <code>trng_interrupt_enable</code> .....	555
Table 3-837 <code>trng_interrupt_disable</code> .....	556
Table 3-838 <code>trng_interrupt_flag_get</code> .....	556
Table 3-839 <code>trng_interrupt_flag_clear</code> .....	557
Table 3-840. USART Registers .....	558
Table 3-841. USART firmware function .....	558
Table 3-842. Enum <code>usart_flag_enum</code> .....	560
Table 3-843. Enum <code>usart_interrupt_flag_enum</code> .....	561
Table 3-844. Enum <code>usart_interrupt_enum</code> .....	561
Table 3-845. Enum <code>usart_invert_enum</code> .....	562
Table 3-846. Function <code>usart_deinit</code> .....	562
Table 3-847. Function <code>usart_baudrate_set</code> .....	563
Table 3-848. Function <code>usart_parity_config</code> .....	563
Table 3-849. Function <code>usart_word_length_set</code> .....	564
Table 3-850. Function <code>usart_stop_bit_set</code> .....	564
Table 3-851. Function <code>usart_enable</code> .....	565
Table 3-852. Function <code>usart_disable</code> .....	566
Table 3-853. Function <code>usart_transmit_config</code> .....	566
Table 3-854. Function <code>usart_receive_config</code> .....	567
Table 3-855. Function <code>usart_data_first_config</code> .....	568
Table 3-856. Function <code>usart_invert_config</code> .....	568
Table 3-857. Function <code>usart_overrun_enable</code> .....	569
Table 3-858. Function <code>usart_overrun_disable</code> .....	570
Table 3-859. Function <code>usart_oversample_config</code> .....	570
Table 3-860. Function <code>usart_sample_bit_config</code> .....	571
Table 3-861. Function <code>usart_receiver_timeout_enable</code> .....	571
Table 3-862. Function <code>usart_receiver_timeout_disable</code> .....	572
Table 3-863. Function <code>usart_receiver_timeout_threshold_config</code> .....	572
Table 3-864. Function <code>usart_data_transmit</code> .....	573
Table 3-865. Function <code>usart_data_receive</code> .....	574
Table 3-866. Function <code>usart_command_enable</code> .....	574
Table 3-867. Function <code>usart_address_config</code> .....	575
Table 3-868. Function <code>usart_address_detection_mode_config</code> .....	576
Table 3-869. Function <code>usart_mute_mode_enable</code> .....	576
Table 3-870. Function <code>usart_mute_mode_disable</code> .....	577
Table 3-871. Function <code>usart_mute_mode_wakeup_config</code> .....	577
Table 3-872. Function <code>usart_lin_mode_enable</code> .....	578
Table 3-873. Function <code>usart_lin_mode_disable</code> .....	578
Table 3-874. Function <code>usart_lin_break_dection_length_config</code> .....	579
Table 3-875. Function <code>usart_halfduplex_enable</code> .....	580

Table 3-876. Function <code>usart_halfduplex_disable</code> .....	580
Table 3-877. Function <code>usart_clock_enable</code> .....	581
Table 3-878. Function <code>usart_clock_disable</code> .....	581
Table 3-879. Function <code>usart_synchronous_clock_config</code> .....	582
Table 3-880. Function <code>usart_guard_time_config</code> .....	582
Table 3-881. Function <code>usart_smartcard_mode_enable</code> .....	583
Table 3-882. Function <code>usart_smartcard_mode_disable</code> .....	584
Table 3-883. Function <code>usart_smartcard_mode_nack_enable</code> .....	584
Table 3-884. Function <code>usart_smartcard_mode_nack_disable</code> .....	585
Table 3-885. Function <code>usart_smartcard_mode_early_nack_enable</code> .....	585
Table 3-886. Function <code>usart_smartcard_mode_early_nack_disable</code> .....	586
Table 3-887. Function <code>usart_smartcard_autoretry_config</code> .....	586
Table 3-888. Function <code>usart_block_length_config</code> .....	587
Table 3-889. Function <code>usart_irda_mode_enable</code> .....	587
Table 3-890. Function <code>usart_irda_mode_disable</code> .....	588
Table 3-891. Function <code>usart_prescaler_config</code> .....	588
Table 3-892. Function <code>usart_irda_lowpower_config</code> .....	589
Table 3-893. Function <code>usart_hardware_flow_rts_config</code> .....	589
Table 3-894. Function <code>usart_hardware_flow_cts_config</code> .....	590
Table 3-895. Function <code>usart_hardware_flow_coherence_config</code> .....	591
Table 3-896. Function <code>usart_rs485_driver_enable</code> .....	591
Table 3-897. Function <code>usart_rs485_driver_disable</code> .....	592
Table 3-898. Function <code>usart_driver_asserttime_config</code> .....	592
Table 3-899. Function <code>usart_driver_deasserttime_config</code> .....	593
Table 3-900. Function <code>usart_depolarity_config</code> .....	593
Table 3-901. Function <code>usart_dma_receive_config</code> .....	594
Table 3-902. Function <code>usart_dma_transmit_config</code> .....	595
Table 3-903. Function <code>usart_reception_error_dma_disable</code> .....	595
Table 3-904. Function <code>usart_reception_error_dma_enable</code> .....	596
Table 3-905. Function <code>usart_wakeup_enable</code> .....	596
Table 3-906. Function <code>usart_wakeup_disable</code> .....	597
Table 3-907. Function <code>usart_wakeup_mode_config</code> .....	597
Table 3-908. Function <code>usart_receive_fifo_enable</code> .....	598
Table 3-909. Function <code>usart_receive_fifo_disable</code> .....	599
Table 3-910. Function <code>usart_receive_fifo_counter_number</code> .....	599
Table 3-911. Function <code>usart_flag_get</code> .....	600
Table 3-912. Function <code>usart_flag_clear</code> .....	601
Table 3-913. Function <code>usart_interrupt_enable</code> .....	602
Table 3-914. Function <code>usart_interrupt_disable</code> .....	603
Table 3-915. Function <code>usart_interrupt_flag_get</code> .....	604
Table 3-916. Function <code>usart_interrupt_flag_clear</code> .....	605
Table 3-917. WWDGT Registers .....	607
Table 3-918. VREF firmware function .....	607
Table 3-919. Function <code>vref_deinit</code> .....	607

Table 3-920. Function <code>vref_enable</code> .....	608
Table 3-921. Function <code>vref_disable</code> .....	608
Table 3-922. Function <code>vref_high_impedance_mode_enable</code> .....	609
Table 3-923. Function <code>vref_high_impedance_mode_disable</code> .....	609
Table 3-924. Function <code>vref_voltage_select</code> .....	610
Table 3-925. Function <code>vref_status_get</code> .....	610
Table 3-926. Function <code>vref_calib_value_set</code> .....	611
Table 3-927. Function <code>vref_calib_value_get</code> .....	611
Table 3-928. WWDGT Registers .....	612
Table 3-929. WWDGT firmware function .....	612
Table 3-930. Function <code>wwdgt_deinit</code> .....	612
Table 3-931. Function <code>wwdgt_enable</code> .....	613
Table 3-932. Function <code>wwdgt_counter_update</code> .....	613
Table 3-933. Function <code>wwdgt_config</code> .....	614
Table 3-934. Function <code>wwdgt_flag_get</code> .....	615
Table 3-935. Function <code>wwdgt_flag_clear</code> .....	616
Table 4-1. Revision history .....	617
Function descriptionS and Input parameter descriptionS Add SLCD in <i>Table 4-2</i> .	
<i>Function</i> <code>rcu_rtc_clock_config</code> .....	617

## 1. Introduction

This manual introduces firmware library of GD32L23x 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 GD32L23x devices are involved in the package. The peripheral driving code and firmware examples on evaluation board are also included in firmware library. Users need not learn each peripherals in details and it's easy to apply a peripheral by using the firmware library. Using firmware library can greatly reduce programming time, thereby reducing development costs.

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

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

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

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

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

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

#### 1.1.1. Peripherals

**Table 1-1. Peripherals**

Peripherals	Descriptions
ADC	Analog-to-digital converter
CAN	Controller area network
CAU	Cryptographic Acceleration Unit
CMP	Comparator
CRC	CRC calculation unit

Peripherals	Descriptions
CTC	Clock trim controller
DBG	Debug
DAC	Digital-to-analog converter
DMA	Direct memory access controller
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
LPTIMER	Low power timer
LPUART	Low-power universal asynchronous receiver /transmitter
MISC	Nested Vectored Interrupt Controller
PMU	Power management unit
RCU	Reset and clock unit
RTC	Real-time Clock
SLCD	Segment LCD controller
SPI/I2S	Serial peripheral interface/Inter-IC sound
SYSCFG	System configuration
TIMER	TIMER
TRNG	True random number generator
USART	Universal synchronous/asynchronous receiver /transmitter
VREF	VREF
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 “gd32l23x\_”, such as: gd32l23x\_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

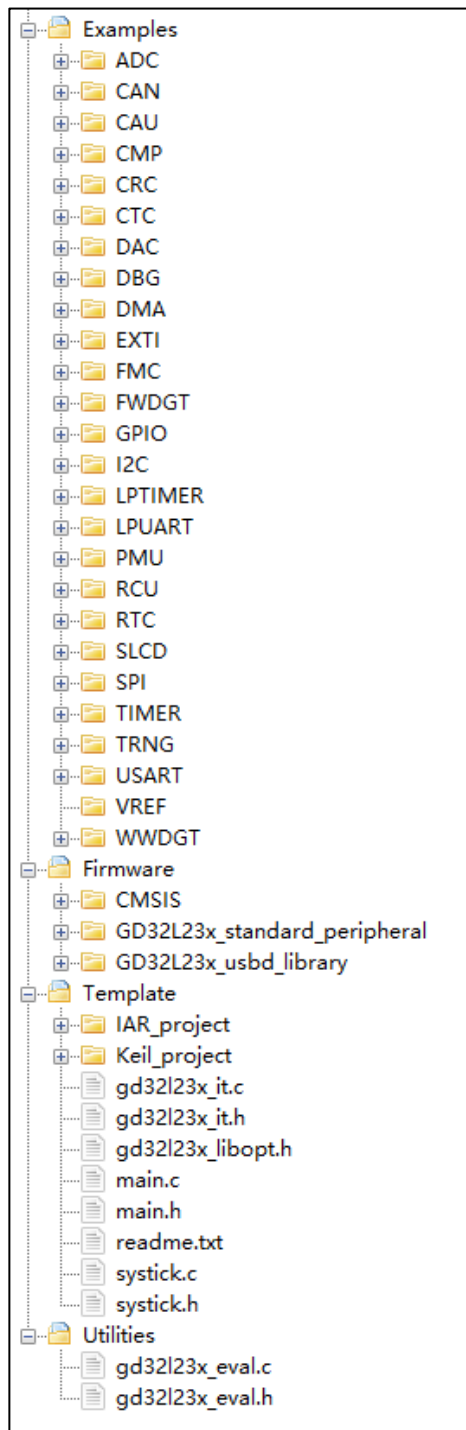
should be adapted among words, and all the peripheral functions are written in lowercase.

## 2. Firmware Library Overview

### 2.1. File Structure of Firmware Library

GD32L23x\_Firmware\_Library, the file structure is shown as below:

**Figure 2-1. File structure of firmware library of GD32L23x**



### 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;
- gd32l23x\_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);
- gd32l23x\_it.c: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- gd32l23x\_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 GD32L23x and system configuration file;
- GD32L23x\_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

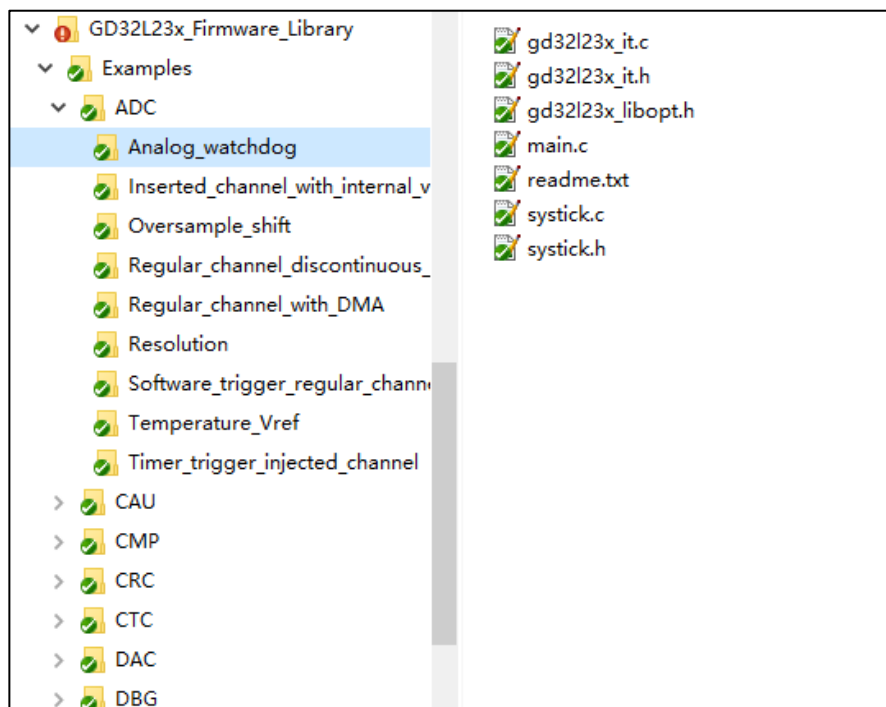
Template folder includes a simple demo of how to use LED, how to print by USART and use key to control, (IAR\_project is run in IAR, and Keil\_project is run in Keil5). User can use the project template to compile the formware examples, the steps are shown as below:

#### 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:



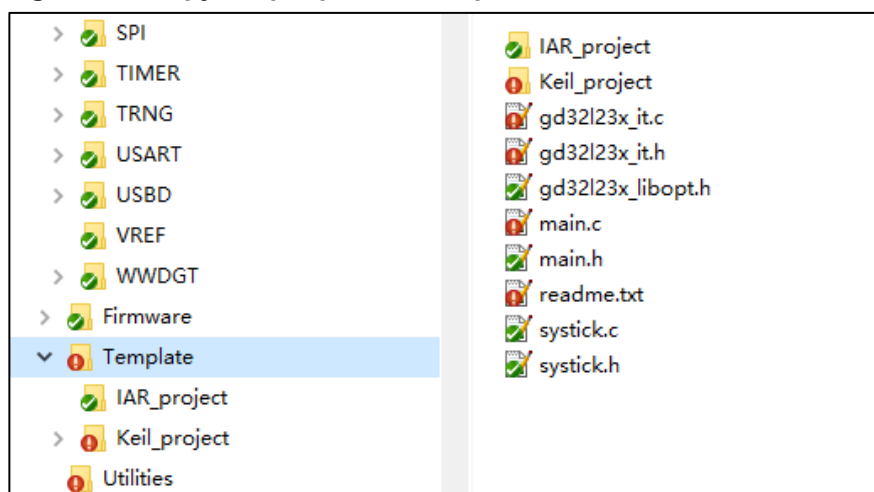
**Figure 2-2. Select peripheral example files**



## 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:

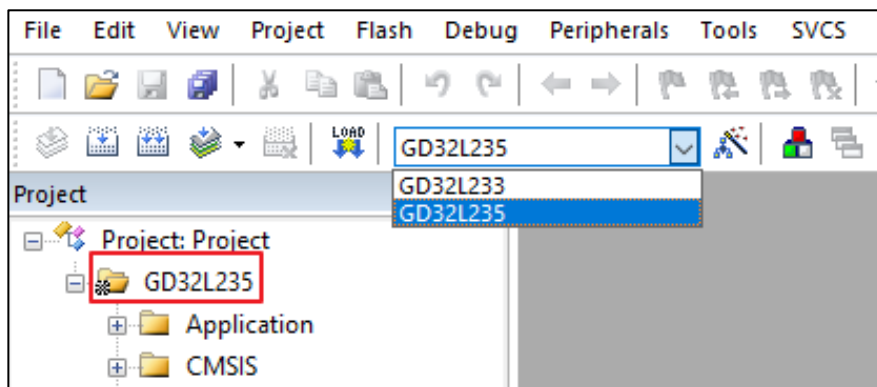
**Figure 2-3. Copy the peripheral example files**



## Open a project

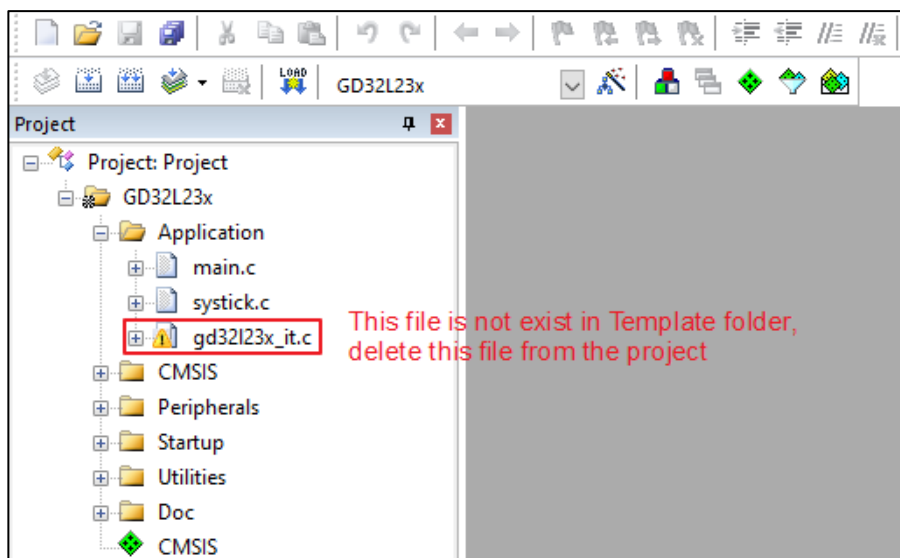
GD provides project in Keil and IAR, users can open project in different IDEs according to their need, such as "Keil\_project", open \Template\Keil\_project\Project.uvprojx, you can select L233 or L235 project, shown as below:

**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:

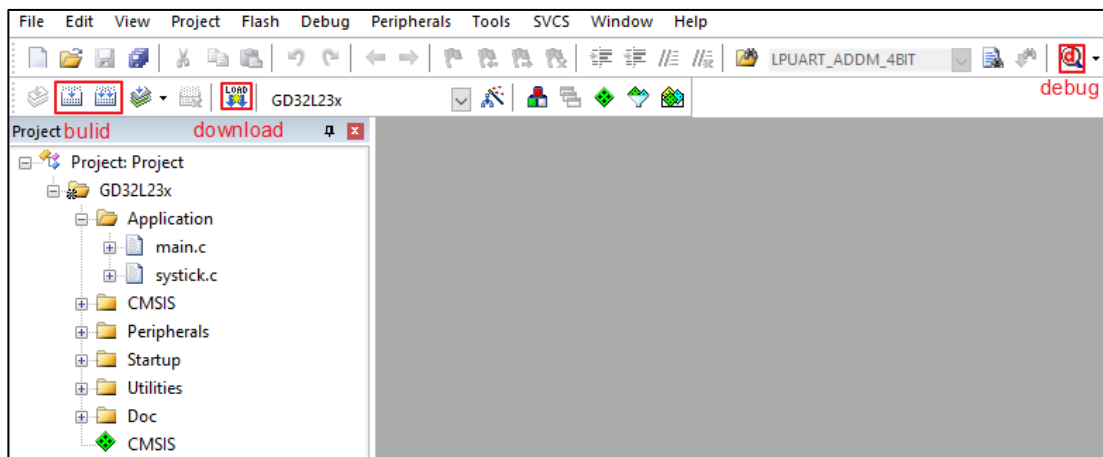
**Figure 2-5. Configure project files**



## 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:

- `gd32e133r_eval.h` is related header file of the evaluation board about running the firmware examples;
- `gd32e233r_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
<code>gd32l23x_libopt.h</code>	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.
<code>main.c</code>	Example of main function.
<code>gd32l23x_it.h</code>	Header file, including all the prototypes of interrupt service routines.
<code>gd32l23x_it.c</code>	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.
<code>gd32l23x_xxx.h</code>	The header file of peripheral xxx, including functions about peripheral xxx, and the variables used for functions.

Files	Descriptions
gd32l23x_xxx.c	The C source file for driving peripheral xxx.
systick.h	The header file of systick.c, including prototypes of systick configuration function and delay function.
systick.c	The source file about systick configuration function and delay function.
readme.txt	Description document about how to configure and how to use the firmware example.

## 3. Firmware Library of Standard Peripherals

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

The description format of firmware functions are shown as below:

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

<b>Function name</b>	Name of peripheral function
<b>Function prototype</b>	Declaration prototype
<b>Function descriptions</b>	Explain the function how to work
<b>Precondition</b>	Requirements should meet before calling this function
<b>The called functions</b>	Other firmware functions called in this function
<b>Input parameter{in}</b>	
<b>Input parameter name</b>	Description
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_WDHT	Watchdog high threshold register

Registers	Descriptions
ADC_WDLT	Watchdog low threshold register
ADC_RSQ0	Routine sequence register 0
ADC_RSQ1	Routine sequence register 1
ADC_RSQ2	Routine sequence register 2
ADC_ISQ	Inserted sequence register
ADC_IDATAx	Inserted data register x(x=0..3)
ADC_RDATA	Routine data register
ADC_OVSAMPCTL	Oversampling control register
ADC_CCTL	Charge control register
ADC_DIFCTL	Differential mode control register

### 3.2.2. Descriptions of Peripheral functions

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

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

Function name	Function description
adc_deinit	reset ADC
adc_enable	enable ADC
adc_disable	disable ADC
adc_calibration_number	configure ADC calibration number
adc_calibration_enable	ADC calibration and reset calibration
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_channel_16_to_19	configure temperature sensor, internal reference voltage channel, VBAT channel or VSLCD channel
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_channel_differential_mode_config	configure differential mode for ADC channel
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_watchdog_single_channel	enable ADC analog watchdog single channel

Function name	Function description
_enable	
adc_watchdog_sequence_channel_enable	enable ADC analog watchdog sequence channel
adc_watchdog_disable	disable ADC analog watchdog
adc_watchdog_threshold_config	configure ADC analog watchdog 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_charge_pulse_width_counter	configure ADC charge pulse width counter
adc_charge_flag_get	get ADC charge flag
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**

Function name	adc_enable
---------------	------------

Function prototype	void adc_enable(void);
Function descriptions	enable ADC
Precondition	-
The called functions	adc_charge_pulse_width_counter
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### adc\_disable

The description of adc\_disable is shown as below:

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

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

Example:

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

### adc\_calibration\_number (only for GD32L235)

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

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

Function name	adc_calibration_number
Function prototype	void adc_calibration_number(uint32_t clb_num);
Function descriptions	configure ADC calibration number
Precondition	-
The called functions	-
Input parameter{in}	



clb_num	calibration number
ADC_CALIBRATION_NUM1	calibrate once
ADC_CALIBRATION_NUM2	calibrate twice
ADC_CALIBRATION_NUM4	calibrate 4 times
ADC_CALIBRATION_NUM8	calibrate 8 times
ADC_CALIBRATION_NUM16	calibrate 16 times
ADC_CALIBRATION_NUM32	calibrate 32 times
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
adc_calibration_number(ADC_CALIBRATION_NUM1);
```

### adc\_calibration\_enable

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

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

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

Example:

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

```
adc_calibration_enable();
```

## adc\_dma\_mode\_enable

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

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

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

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 3-10. Function adc\_dma\_mode\_disable**

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

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 3-11. Function adc\_discontinuous\_mode\_config**

<b>Function name</b>	adc_discontinuous_mode_config
----------------------	-------------------------------

<b>Function prototype</b>	void adc_discontinuous_mode_config(uint8_t adc_sequence, uint8_t length);
<b>Function descriptions</b>	configure ADC discontinuous mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	select the sequence
ADC_ROUTINE_CHANNEL	routine sequence
ADC_INSERTED_CHANNEL	inserted sequence
ADC_CHANNEL_DISCON_DISABLE	disable discontinuous mode of routine and inserted sequence
<b>Input parameter{in}</b>	
<b>length</b>	number of conversions in discontinuous mode,the number can be 1..16 for routine sequence, the number has no effect for inserted sequence
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-12. 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>	-
<b>Input parameter{in}</b>	
<b>function</b>	the function to configure
ADC_SCAN_MODE	scan mode select
ADC_INSERTED_CHANNEL_AUTO	inserted sequence convert automatically
ADC_CONTINUOUS_MODE	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\_channel\_16\_to\_19

The description of adc\_channel\_16\_to\_19 is shown as below:

**Table 3-13. Function adc\_channel\_16\_to\_19**

<b>Function name</b>	adc_channel_16_to_19
<b>Function prototype</b>	void adc_channel_16_to_19(uint32_t function, ControlStatus newvalue);
<b>Function descriptions</b>	configure temperature sensor, internal reference voltage channel, VBAT channel or VSLCD channel
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>function</b>	temperature sensor or internal reference voltage channel or VBAT channel or VSLCD channel function
<i>ADC_TEMP_CHANNEL_SWITCH</i>	channel 16 (temperature sensor) switch of ADC
<i>ADC_INTERNAL_CHANNEL_SWITCH</i>	channel 17 (internal reference voltage) switch of ADC
<i>ADC_VBAT_CHANNEL_SWITCH</i>	channel 18 (1/3 voltage of external battery) switch of ADC
<i>ADC_VSLCD_CHANNEL_SWITCH</i>	channel 19 (1/3 voltage of VSLCD) switch of ADC
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 temperature sensor*/
```

```
adc_channel_16_to_19(ADC_TEMP_CHANNEL_SWITCH, ENABLE);
```

### adc\_data\_alignment\_config

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

**Table 3-14. 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 3-15. 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

<code>ADC_INSERTED_CHANNEL</code>	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 channel 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 3-16. Function `adc_routine_channel_config`**

<b>Function name</b>	<code>adc_routine_channel_config</code>
<b>Function prototype</b>	<code>void adc_routine_channel_config(uint8_t rank, uint8_t adc_channel, uint32_t sample_time);</code>
<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
<code>ADC_CHANNEL_x</code>	ADC Channelx (x=0..19)
<b>Input parameter{in}</b>	
<b>sample_time</b>	the sample time value
<code>ADC_SAMPLETIME_2POINT5</code>	2.5 cycles
<code>ADC_SAMPLETIME_7POINT5</code>	7.5 cycles
<code>ADC_SAMPLETIME_13POINT5</code>	13.5 cycles
<code>ADC_SAMPLETIME_28POINT5</code>	28.5 cycles
<code>ADC_SAMPLETIME_41POINT5</code>	41.5 cycles
<code>ADC_SAMPLETIME_55POINT5</code>	55.5 cycles

55POINT5	
ADC_SAMPLETIME_71POINT5	71.5 cycles
ADC_SAMPLETIME_239POINT5	239.5 cycles
Output parameter{out}	
-	-
Return value	
-	-

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 3-17. 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>	-
Input parameter{in}	
<b>rank</b>	the inserted sequencer rank, this parameter must be between 0 to 3
Input parameter{in}	
<b>adc_channel</b>	the selected ADC channel
ADC_CHANNEL_x	ADC Channelx (x=0..19)
Input parameter{in}	
<b>sample_time</b>	the sample time value
ADC_SAMPLETIME_2POINT5	2.5 cycles
ADC_SAMPLETIME_7POINT5	7.5 cycles
ADC_SAMPLETIME_13POINT5	13.5 cycles
ADC_SAMPLETIME_28POINT5	28.5 cycles
ADC_SAMPLETIME_41POINT5	41.5 cycles
ADC_SAMPLETIME_55POINT5	55.5 cycles

55POINT5	
ADC_SAMPLETIME_71POINT5	71.5 cycles
ADC_SAMPLETIME_239POINT5	239.5 cycles
Output parameter{out}	
-	-
Return value	
-	-

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 3-18. Function adc\_inserted\_channel\_offset\_config**

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

Example:

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

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

### adc\_channel\_differential\_mode\_config (only for GD32L235)

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



Table 3-19. Function `adc_channel_differential_mode_config`

<b>Function name</b>	<code>adc_channel_differential_mode_config</code>
<b>Function prototype</b>	<code>void adc_channel_differential_mode_config(uint32_t adc_channel, ControlStatus newvalue);</code>
<b>Function descriptions</b>	configure differential mode for ADC channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_channel</b>	the channel use differential mode
<code>ADC_DIFFERENTIAL_MODE_CHANNEL_x</code>	ADC Channelx (x=0..14) for differential mode
<code>ADC_DIFFERENTIAL_MODE_CHANNEL_ALL</code>	ADC Channel0~14 for differential mode
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<code>ENABLE</code>	enable function
<code>DISABLE</code>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC channel0 differential mode */
```

```
adc_channel_differential_mode_config(ADC_DIFFERENTIAL_MODE_CHANNEL_0,
ENABLE);
```

### `adc_external_trigger_config`

The description of `adc_external_trigger_config` is shown as below:

Table 3-20. Function `adc_external_trigger_config`

<b>Function name</b>	<code>adc_external_trigger_config</code>
<b>Function prototype</b>	<code>void adc_external_trigger_config(uint8_t adc_sequence, ControlStatus newvalue);</code>
<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
<code>ADC_ROUTINE_CHANNEL</code>	routine sequence
<code>ADC_INSERTED_</code>	inserted sequence

<i>CHANNEL</i>	
<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 3-21. 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_CHANNEL</i>	routine sequence
<i>ADC_INSERTED_CHANNEL</i>	inserted sequence
<b>Input parameter{in}</b>	
<b>external_trigger_source</b>	routine or inserted sequence trigger source
<i>ADC_EXTTRIG_ROUTINE_T8_CH0</i>	external trigger TIMER8 CH0 event select for routine sequence
<i>ADC_EXTTRIG_ROUTINE_T8_CH1</i>	external trigger TIMER8 CH1 event select for routine sequence
<i>ADC_EXTTRIG_ROUTINE_T0_CH2</i>	external trigger TIMER0 CH2 event select for routine sequence (for GD32L235xx devices)
<i>ADC_EXTTRIG_ROUTINE_T1_CH1</i>	external trigger TIMER1 CH1 event select for routine sequence
<i>ADC_EXTTRIG_ROUTINE_T2_TRGO</i>	external trigger TIMER2 TRGO event select for routine sequence

NE_T2_TRGO	
ADC_EXTTRIG_ROUTINE_NE_T11_CH0	external trigger TIMER11 CH0 event select for routine sequence
ADC_EXTTRIG_ROUTINE_NE_EXTI_11	external trigger interrupt line 11 select for routine sequence
ADC_EXTTRIG_ROUTINE_NE_NONE	external trigger software event select for routine sequence
ADC_EXTTRIG_INSERTED_TED_T0_TRGO	TIMER0 TRGO event select for inserted sequence (for GD32L235xx devices)
ADC_EXTTRIG_INSERTED_TED_T0_CH3	TIMER0 CH3 event select for inserted sequence (for GD32L235xx devices)
ADC_EXTTRIG_INSERTED_TED_T14_TRGO	TIMER14 TRGO event select for inserted sequence (for GD32L235xx devices)
ADC_EXTTRIG_INSERTED_TED_T1_TRGO	TIMER1 TRGO event select for inserted sequence
ADC_EXTTRIG_INSERTED_TED_T1_CH0	TIMER1 CH0 event select for inserted sequence
ADC_EXTTRIG_INSERTED_TED_T2_CH3	TIMER2 CH3 event select for inserted sequence
ADC_EXTTRIG_INSERTED_TED_EXTI_15	external interrupt line 15 event select for inserted sequence
ADC_EXTTRIG_INSERTED_TED_NONE	external trigger software event select for inserted sequence
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
adc_external_trigger_source_config(ADC_ROUTINE_CHANNEL,  
ADC_EXTTRIG_ROUTINE_T8_CH0);
```

### adc\_software\_trigger\_enable

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

**Table 3-22. Function adc\_software\_trigger\_enable**

Function name	adc_software_trigger_enable
Function prototype	void adc_software_trigger_enable(uint8_t adc_sequence);
Function descriptions	enable ADC software trigger
Precondition	-
The called functions	-

Input parameter{in}	
<b>adc_sequence</b>	select the sequence
<i>ADC_ROUTINE_CHAN NEL</i>	routine sequence
<i>ADC_INSERTED_CHA NNEL</i>	inserted sequence
Output parameter{out}	
-	-
Return value	
-	-

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 3-23. 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>	-
Output parameter{out}	
-	-
Return value	
<b>uint16_t</b>	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 3-24. Function adc\_inserted\_data\_read**

<b>Function name</b>	adc_inserted_data_read
<b>Function prototype</b>	uint16_t adc_inserted_data_read(uint8_t inserted_channel);
<b>Function descriptions</b>	read ADC inserted sequence data register

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>inserted_channel</b>	insert channel select
<i>ADC_INSERTED_CHANNEL_x</i>	inserted Channelx, x=0,1,2,3
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	the conversion value: 0~0xFFFF

Example:

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

```
uint16_t adc_value = 0;
```

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

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

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

**Table 3-25. Function adc\_watchdog\_single\_channel\_enable**

<b>Function name</b>	adc_watchdog_single_channel_enable
<b>Function prototype</b>	void adc_watchdog_single_channel_enable(uint8_t adc_channel);
<b>Function descriptions</b>	enable ADC analog watchdog single channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_channel</b>	the selected ADC channel
<i>ADC_CHANNEL_x</i>	ADC Channelx(x=0..19)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
adc_watchdog_single_channel_enable(ADC_CHANNEL_1);
```

### adc\_watchdog\_sequence\_channel\_enable

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

Table 3-26. Function `adc_watchdog_sequence_channel_enable`

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

Example:

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

```
adc_watchdog_sequence_channel_enable(ADC_ROUTINE_CHANNEL);
```

### **adc\_watchdog\_disable**

The description of `adc_watchdog_disable` is shown as below:

Table 3-27. Function `adc_watchdog_disable`

<b>Function name</b>	<code>adc_watchdog_disable</code>
<b>Function prototype</b>	<code>void adc_watchdog_disable(void);</code>
<b>Function descriptions</b>	disable ADC analog watchdog
<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 */
```

```
adc_watchdog_disable();
```

## adc\_watchdog\_threshold\_config

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

**Table 3-28. Function adc\_watchdog\_threshold\_config**

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

Example:

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

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

## adc\_resolution\_config

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

**Table 3-29. 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

6B	
Output parameter{out}	
-	-
Return value	
-	-

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 3-30. 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_RATIO_MUL64	oversampling ratio multiple 64
ADC_OVERSAMPLING_RATIO_MUL128	oversampling ratio multiple 128
ADC_OVERSAMPLING_RATIO_MUL256	oversampling ratio multiple 256
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure 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 3-31. Function adc\_oversample\_mode\_enable**

<b>Function name</b>	adc_oversample_mode_enable
<b>Function prototype</b>	void adc_oversample_mode_enable(void);
<b>Function descriptions</b>	enable ADC oversample mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

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

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 3-32. Function adc\_oversample\_mode\_disable**

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

Example:

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

### adc\_charge\_pulse\_width\_counter

The description of adc\_charge\_pulse\_width\_counter is shown as below:

**Table 3-33. Function adc\_charge\_pulse\_width\_counter**

<b>Function name</b>	adc_charge_pulse_width_counter
<b>Function prototype</b>	void adc_charge_pulse_width_counter(uint32_t value);
<b>Function descriptions</b>	configure ADC charge pulse width counter
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>value</b>	ADC charge pulse width counter value

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC charge pulse width counter */
adc_charge_pulse_width_counter(100);
```

### adc\_charge\_flag\_get

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

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

Function name	adc_charge_flag_get
Function prototype	FlagStatus adc_charge_flag_get(uint32_t flag);
Function descriptions	get the ADC charge flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	the ADC charge flag
ADC_FLAG_CHARGE	ADC charge flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the ADC charge flag */
FlagStatus flag_value;
flag_value = adc_charge_flag_get(ADC_FLAG_CHARGE);
```

### adc\_flag\_get

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

**Table 3-35. Function adc\_flag\_get**

Function name	adc_flag_get
Function prototype	FlagStatus adc_flag_get(uint32_t flag);
Function descriptions	get ADC flag
Precondition	-
The called functions	-
Input parameter{in}	

flag	ADC flag
<i>ADC_FLAG_WDE</i>	analog watchdog event flag
<i>ADC_FLAG_EOC</i>	end of sequence conversion flag
<i>ADC_FLAG_EOIC</i>	end of inserted sequence conversion flag
<i>ADC_FLAG_STIC</i>	start flag of inserted sequence
<i>ADC_FLAG_STRC</i>	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 flag*/

FlagStatus flag_value;

flag_value = adc_flag_get(ADC_FLAG_WDE);
```

### adc\_flag\_clear

The description of `adc_flag_clear` is shown as below:

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

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

Example:

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

adc_flag_clear(ADC_FLAG_WDE);
```

## adc\_interrupt\_enable

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

**Table 3-37. 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
<i>ADC_INT_WDE</i>	analog watchdog 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:

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

## adc\_interrupt\_disable

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

**Table 3-38. Function adc\_interrupt\_disable**

<b>Function name</b>	adc_interrupt_disable
<b>Function prototype</b>	void adc_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable ADC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	ADC interrupt
<i>ADC_INT_WDE</i>	analog watchdog 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 interrupt */
```

```
adc_interrupt_disable(ADC_INT_WDE);
```

### adc\_interrupt\_flag\_get

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

**Table 3-39. Function adc\_interrupt\_flag\_get**

<b>Function name</b>	adc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus adc_interrupt_flag_get(uint32_t int_flag);
<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
ADC_INT_FLAG_WDE	analog watchdog interrupt flag
ADC_INT_FLAG_EOC	end of sequence conversion interrupt flag
ADC_INT_FLAG_EOIC	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 interrupt flag */
```

```
FlagStatus flag_value;
```

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

### adc\_interrupt\_flag\_clear

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

**Table 3-40. 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
ADC_INT_FLAG_WDE	analog watchdog interrupt flag

ADC_INT_FLAG_EOC	end of sequence conversion interrupt flag
ADC_INT_FLAG_EOIC	end of inserted sequence conversion interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
adc_interrupt_flag_clear(ADC_INT_FLAG_WDE);
```

### 3.3. CAN(only for GD32L235)

CAN bus (for Controller Area Network) is a bus standard designed to allow microcontrollers and devices to communicate with each other without a host computer. The CAN registers are listed in chapter [3.3.1](#), the CAN firmware functions are introduced in chapter [3.3.2](#).

#### 3.3.1. Descriptions of Peripheral registers

CAN registers are listed in the table shown as below:

**Table 3-41. CAN Registers**

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

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

### 3.3.2. Descriptions of Peripheral functions

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

**Table 3-42. CAN firmware function**

Function name	Function description
can_deinit	deinitialize CAN
can_struct_para_init	initialize CAN parameter struct with a default value
can_init	initialize CAN
can_filter_init	initialize CAN filter
can_debug_freeze_enable	CAN debug freeze enable
can_debug_freeze_disable	CAN debug freeze disable
can_time_trigger_mode_enable	CAN time trigger mode enable
can_time_trigger_mode_disable	CAN time trigger mode disable
can_message_transmit	transmit CAN message
can_transmit_states	get CAN transmit state
can_transmission_stop	stop CAN transmission
can_message_receive	CAN receive message
can_fifo_release	CAN release fifo
can_receive_message_length_get	CAN receive message length
can_working_mode_set	CAN working mode
can_wakeup	CAN wakeup from sleep mode
can_error_get	CAN get error
can_receive_error_number_get	get CAN receive error number
can_transmit_error_number_get	get CAN transmit error number
can_flag_get	CAN get flag state
can_flag_clear	CAN clear flag state
can_interrupt_enable	CAN interrupt enable
can_interrupt_disable	CAN interrupt disable
can_interrupt_flag_get	CAN get interrupt flag state
can_interrupt_flag_clear	CAN clear interrupt flag state



## Structure can\_parameter\_struct

**Table 3-43. Structure can\_parameter\_struct**

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

## Structure can\_transmit\_message\_struct

**Table 3-44. Structure can\_transmit\_message\_struct**

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

## Structure can\_receive\_message\_struct

**Table 3-45. Structure can\_receive\_message\_struct**

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

## Structure can\_filter\_parameter\_struct

**Table 3-46. Structure can\_filter\_parameter\_struct**

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

## can\_deinit

The description of can\_deinit is shown as below:

**Table 3-47. Function can\_deinit**

Function name	can_deinit
Function prototype	void can_deinit(void);
Function descriptions	deinitialize CAN
Precondition	-
The called functions	rcu_periph_reset_enable/ rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

## can\_struct\_para\_init

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

**Table 3-48. Function can\_struct\_para\_init**

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

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>type</b>	Sturct type
<i>CAN_INIT_STRUCT</i>	CAN initilize parameters struct
<i>CAN_FILTER_STRUCT</i>	CAN filter parameters struct
<i>CAN_TX_MESSAGE_STRUCT</i>	CAN transmit message struct
<i>CAN_RX_MESSAGE_STRUCT</i>	CAN receive message struct
<b>Output parameter{out}</b>	
<b>p_struct</b>	the struct pointer that needs initialize
<b>Return value</b>	
-	-

Example:

```
can_parameter_struct can_init;
```

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

## can\_init

The description of can\_init is shown as below:

**Table 3-49. Function can\_init**

<b>Function name</b>	can_init
<b>Function prototype</b>	ErrStatus can_init(can_parameter_struct* can_parameter_init);
<b>Function descriptions</b>	initialize CAN
<b>Precondition</b>	can_struct_para_init()
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_parameter_init</b>	CAN parameter initialization stuct, the structure members can refer to members of the structure <a href="#">Table 3-43. Structure can_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS / ERROR

Example:

```
can_parameter_struct can_parameter;
```

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

```
can_init (&can_parameter);
```

## can\_filter\_init

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

**Table 3-50. Function can\_filter\_init**

<b>Function name</b>	can_filter_init
<b>Function prototype</b>	void can_filter_init(can_filter_parameter_struct* can_filter_parameter_init);
<b>Function descriptions</b>	initialize CAN filter
<b>Precondition</b>	can_struct_para_init()
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_filter_parameter_i nit</b>	CAN filter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-46. Structure can_filter_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize CAN filter */
can_filter_init(&can_filter);
```

## can\_debug\_freeze\_enable

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

**Table 3-51. Function can\_debug\_freeze\_enable**

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

Example:

```
/* enable CAN debug freeze */
can_debug_freeze_enable ();
```

## can\_debug\_freeze\_disable

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

**Table 3-52. Function can\_debug\_freeze\_disable**

<b>Function name</b>	can_debug_freeze_disable
<b>Function prototype</b>	void can_debug_freeze_disable(void);
<b>Function descriptions</b>	disable CAN debug freeze
<b>Precondition</b>	-
<b>The called functions</b>	dbg_periph_disable
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## can\_time\_trigger\_mode\_enable

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

**Table 3-53. Function can\_time\_trigger\_mode\_enable**

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

Example:

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

## can\_time\_trigger\_mode\_disable

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

**Table 3-54. Function can\_time\_trigger\_mode\_disable**

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

Example:

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

## can\_message\_transmit

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

**Table 3-55. Function can\_message\_transmit**

<b>Function name</b>	can_message_transmit
<b>Function prototype</b>	uint8_t can_message_transmit(can_transmit_message_struct* transmit_message);
<b>Function descriptions</b>	transmit CAN message
<b>Precondition</b>	can_struct_para_init()
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
transmit_message	CAN transmit message struct, the structure members can refer to members of the structure <a href="#">Table 3-44. Structure can transmit message struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint8_t	0x00-0x03

Example:

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

```
uint8_t transmit_mailbox = 0;

transmit_mailbox = can_message_transmit(&transmit_message);
```

## can\_transmit\_states

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

**Table 3-56. Function can\_transmit\_states**

<b>Function name</b>	can_transmit_states
<b>Function prototype</b>	can_transmit_state_enum can_transmit_states(uint8_t mailbox_number);
<b>Function descriptions</b>	get CAN transmit state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mailbox_number</b>	Mailbox number
<b>CAN_MAILBOXx</b>	CAN_MAILBOXx(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>can_transmit_state_enum</b>	0..4

Example:

```
/* CAN mailbox0 transmit state */

uint8_t transmit_state = 0;

transmit_state = can_transmit_states (CAN_MAILBOX0);
```

## can\_transmission\_stop

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

**Table 3-57. Function can\_transmission\_stop**

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

-	-
---	---

Example:

```
/* stop CAN mailbox0 transmission */
can_transmission_stop (CAN_MAILBOX0);
```

## can\_message\_receive

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

**Table 3-58. Function can\_message\_receive**

<b>Function name</b>	can_message_receive
<b>Function prototype</b>	void can_message_receive(uint8_t fifo_number, can_receive_message_struct* receive_message);
<b>Function descriptions</b>	CAN receive message
<b>Precondition</b>	can_struct_para_init()
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>fifo_number</b>	Fifo number
<i>CAN_FIFOx</i>	CAN_FIFOx(x=0,1)
<b>Input parameter{in}</b>	
<b>receive_message</b>	CAN message receive struct, the structure members can refer to members of the structure <a href="#">Table 3-45. Structure can_receive_message_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## can\_fifo\_release

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

**Table 3-59. Function can\_fifo\_release**

<b>Function name</b>	can_fifo_release
<b>Function prototype</b>	void can_fifo_release(uint8_t fifo_number);
<b>Function descriptions</b>	release FIFO0
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	



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

Example:

```
/* CAN release FIFO0*/
```

```
can_fifo_release (CAN_FIFO0);
```

### can\_receive\_message\_length\_get

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

**Table 3-60. Function can\_receive\_message\_length\_get**

<b>Function name</b>	can_receive_message_length_get
<b>Function prototype</b>	uint8_t can_receive_message_length_get(uint8_t fifo_number);
<b>Function descriptions</b>	CAN receive message length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>fifo_number</b>	Fifo number
<i>CAN_FIFOx</i>	CAN_FIFOx(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	0..3

Example:

```
/* CAN FIFO0 receive message length */
```

```
uint8_t frame_number = 0;
```

```
frame_number = can_receive_message_length_get (CAN_FIFO0);
```

### can\_working\_mode\_set

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

**Table 3-61. Function can\_working\_mode\_set**

<b>Function name</b>	can_working_mode_set
<b>Function prototype</b>	ErrStatus can_working_mode_set(uint8_t working_mode);
<b>Function descriptions</b>	set CAN working mode
<b>Precondition</b>	-

The called functions	-
Input parameter{in}	
can_working_mode	Mode select
CAN_MODE_INITIALIZE	Initialize mode
CAN_MODE_NORMAL	Normal mode
CAN_MODE_SLEEP	Sleep mode
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS / ERROR

Example:

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

### can\_wakeup

The description of can\_wakeup is shown as below:

**Table 3-62. Function can\_wakeup**

Function name	can_wakeup
Function prototype	ErrStatus can_wakeup(void);
Function descriptions	wake up CAN
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS / ERROR

Example:

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

### can\_error\_get

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

**Table 3-63. Function can\_error\_get**

Function name	can_error_get
---------------	---------------

<b>Function prototype</b>	can_error_enum can_error_get(void);
<b>Function descriptions</b>	get CAN error type
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
can_error_enum	0..7

Example:

```
/* get CAN error type */
```

```
can_error_get ();
```

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

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

**Table 3-64. Function can\_receive\_error\_number\_get**

<b>Function name</b>	can_receive_error_number_get
<b>Function prototype</b>	uint8_t can_receive_error_number_get(void);
<b>Function descriptions</b>	get CAN receive error number
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint8_t	0..255

Example:

```
/* get CAN receive error number */
```

```
uint8_t error_num;
```

```
error_num = can_receive_error_number_get ();
```

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

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

**Table 3-65. Function can\_transmit\_error\_number\_get**

<b>Function name</b>	can_transmit_error_number_get
----------------------	-------------------------------

<b>Function prototype</b>	uint8_t can_transmit_error_number_get(void);
<b>Function descriptions</b>	get CAN transmit error number
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint8_t	0..255

Example:

```

/* get CAN transmit error number */

uint8_t error_num;

error_num = can_transmit_error_number_get ();

```

### can\_flag\_get

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

**Table 3-66. Function can\_flag\_get**

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

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

Example:

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

```
can_flag_get (CAN_FLAG_MTF0);
```

### can\_flag\_clear

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

**Table 3-67. Function can\_flag\_clear**

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

Example:

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

```
can_flag_clear (CAN_FLAG_MTE0);
```

### can\_interrupt\_enable

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

Table 3-68. Function can\_interrupt\_enable

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

Example:

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

```
can_interrupt_enable (CAN_INT_TME);
```

### can\_interrupt\_disable

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

Table 3-69. Function can\_interrupt\_disable

<b>Function name</b>	can_interrupt_disable
<b>Function prototype</b>	void can_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable CAN interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	Interrupt type

<i>CAN_INT_TME</i>	transmit mailbox empty interrupt enable
<i>CAN_INT_RFNE0</i>	receive FIFO0 not empty interrupt enable
<i>CAN_INT_RFF0</i>	receive FIFO0 full interrupt enable
<i>CAN_INT_RFO0</i>	receive FIFO0 overfull interrupt enable
<i>CAN_INT_RFNE1</i>	receive FIFO1 not empty interrupt enable
<i>CAN_INT_RFF1</i>	receive FIFO1 full interrupt enable
<i>CAN_INT_RFO1</i>	receive FIFO1 overfull interrupt enable
<i>CAN_INT_WERR</i>	warning error interrupt enable
<i>CAN_INT_PERR</i>	passive error interrupt enable
<i>CAN_INT_BO</i>	bus-off interrupt enable
<i>CAN_INT_ERRN</i>	error number interrupt enable
<i>CAN_INT_ERR</i>	error interrupt enable
<i>CAN_INT_WU</i>	wakeup interrupt enable
<i>CAN_INT_SLPW</i>	sleep working interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
can_interrupt_disable (CAN_INT_TME);
```

### can\_interrupt\_flag\_get

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

**Table 3-70. Function can\_interrupt\_flag\_get**

<b>Function name</b>	can_interrupt_flag_get
<b>Function prototype</b>	FlagStatus can_interrupt_flag_get(can_interrupt_flag_enum flag);
<b>Function descriptions</b>	get CAN interrupt flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	CAN interrupt flags
<i>CAN_INT_FLAG_SLPWF</i>	status change interrupt flag of sleep working mode entering
<i>CAN_INT_FLAG_WUWF</i>	status change interrupt flag of wakeup from sleep working mode
<i>CAN_INT_FLAG_ERRIF</i>	error interrupt flag
<i>CAN_INT_FLAG_MTF2</i>	mailbox 2 transmit finished interrupt flag
<i>CAN_INT_FLAG_MTF1</i>	mailbox 1 transmit finished interrupt flag
<i>CAN_INT_FLAG_MTF0</i>	mailbox 0 transmit finished interrupt flag

CAN_INT_FLAG_RFO0	receive FIFO0 overfull interrupt flag
CAN_INT_FLAG_RFF0	receive FIFO0 full interrupt flag
CAN_INT_FLAG_RFO1	receive FIFO1 overfull interrupt flag
CAN_INT_FLAG_RFF1	receive FIFO1 full interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get CAN mailbox 0 transmit finished interrupt flag */
```

```
can_interrupt_flag_get (CAN_INT_FLAG_MTF0);
```

### can\_interrupt\_flag\_clear

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

**Table 3-71. Function can\_interrupt\_flag\_clear**

<b>Function name</b>	can_interrupt_flag_clear
<b>Function prototype</b>	void can_interrupt_flag_clear(can_interrupt_flag_enum flag);
<b>Function descriptions</b>	clear CAN interrupt flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
CANx(x=0,1)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
<b>Input parameter{in}</b>	
<b>flag</b>	CAN interrupt flags
CAN_INT_FLAG_SLPIF	status change interrupt flag of sleep working mode entering
CAN_INT_FLAG_WUIF	status change interrupt flag of wakeup from sleep working mode
CAN_INT_FLAG_ERRIF	error interrupt flag
CAN_INT_FLAG_MTF2	mailbox 2 transmit finished interrupt flag
CAN_INT_FLAG_MTF1	mailbox 1 transmit finished interrupt flag
CAN_INT_FLAG_MTF0	mailbox 0 transmit finished interrupt flag
CAN_INT_FLAG_RFO0	receive FIFO0 overfull interrupt flag
CAN_INT_FLAG_RFF0	receive FIFO0 full interrupt flag
CAN_INT_FLAG_RFO1	receive FIFO1 overfull interrupt flag
CAN_INT_FLAG_RFF1	receive FIFO1 full interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	



-	-
---	---

Example:

```
/* clear CAN mailbox 0 transmit finished interrupt flag */
```

```
can_interrupt_flag_clear (CAN_INT_FLAG_MTF0);
```

## 3.4. CAU

The Cryptographic Acceleration Unit supports acceleration of DES, Triple-DES or AES (128,192, or 256) algorithms. The CAU registers are listed in chapter [3.4.1](#) the CAU firmware functions are introduced in chapter [3.4.2](#).

### 3.4.1. Descriptions of Peripheral registers

CAU registers are listed in the table shown as below:

**Table 3-72. CAU Registers**

Registers	Descriptions
CAU_CTL	control register
CAU_STAT0	status register 0
CAU_DI	data input register
CAU_DO	data output register
CAU_DMAEN	DMA enable register
CAU_INTEN	interrupt enable register
CAU_STAT1	status register 1
CAU_INTF	interrupt flag register
CAU_KEY0H	key 0 high register
CAU_KEY0L	key 0 low register
CAU_KEY1H	key 1 high register
CAU_KEY1L	key 1 low register
CAU_KEY2H	key 2 high register
CAU_KEY2L	key 2 low register
CAU_KEY3H	key 3 high register
CAU_KEY3L	key 3 low register
CAU_IV0H	initial vector 0 high register
CAU_IV0L	initial vector 0 low register
CAU_IV1H	initial vector 1 high register
CAU_IV1L	initial vector 1 low register
CAU_GCMCCMCT XSx (x = 0..7)	GCM or CCM mode context switch register x
CAU_GCMCTXSx (x = 0..7)	GCM mode context switch register x

### 3.4.2. Descriptions of Peripheral functions

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

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

Function name	Function description
cau_deinit	reset the CAU peripheral
cau_struct_para_init	initialize the CAU encrypt and decrypt parameter struct with the default values
cau_key_struct_para_init	initialize the key parameter struct with the default values
cau_iv_struct_para_init	initialize the vectors parameter struct with the default values
cau_context_struct_para_init	initialize the context parameter struct with the default values
cau_enable	enable the CAU peripheral
cau_disable	disable the CAU peripheral
cau_dma_enable	enable the CAU DMA interface
cau_dma_disable	disable the CAU DMA interface
cau_init	initialize the CAU peripheral
cau_aes_keysize_config	configure key size if use AES algorithm
cau_key_init	initialize the key parameters
cau_iv_init	initialize the vectors parameters
cau_phase_config	configure phase
cau_fifo_flush	flush the IN and OUT FIFOs
cau_enable_state_get	return whether CAU peripheral is enabled or disabled
cau_data_write	write data to the IN FIFO
cau_data_read	return the last data entered into the output FIFO
cau_context_save	save context before context switching
cau_context_restore	restore context after context switching
cau_aes_ecb	encrypt and decrypt using AES in ECB mode
cau_aes_cbc	encrypt and decrypt using AES in CBC mode
cau_aes_ctr	encrypt and decrypt using AES in CTR mode
cau_aes_cfb	encrypt and decrypt using AES in CFB mode
cau_aes_ofb	encrypt and decrypt using AES in OFB mode
cau_aes_gcm	encrypt and decrypt using AES in GCM mode
cau_aes_ccm	encrypt and decrypt using AES in CCM mode
cau_tdes_ecb	encrypt and decrypt using TDES in ECB mode
cau_tdes_cbc	encrypt and decrypt using TDES in CBC mode
cau_des_ecb	encrypt and decrypt using DES in ECB mode
cau_des_cbc	encrypt and decrypt using DES in CBC mode
cau_flag_get	get the CAU flag status
cau_interrupt_enable	enable the CAU interrupts
cau_interrupt_disable	disable the CAU interrupts
cau_interrupt_flag_get	get the interrupt flag

## Structure cau\_key\_parameter\_struct

**Table 3-74. Structure cau\_key\_parameter\_struct**

Member name	Function description
key_0_high	key 0 high
key_0_low	key 0 low
key_1_high	key 1 high
key_1_low	key 1 low
key_2_high	key 2 high
key_2_low	key 2 low
key_3_high	key 3 high
key_3_low	key 3 low

## Structure cau\_iv\_parameter\_struct

**Table 3-75. Structure cau\_iv\_parameter\_struct**

Member name	Function description
iv_0_high	init vector 0 high
iv_0_low	init vector 0 low
iv_1_high	init vector 1 high
iv_1_low	init vector 1 low

## Structure cau\_context\_parameter\_struct

**Table 3-76. Structure cau\_context\_parameter\_struct**

Member name	Function description
ctl_config	current configuration
iv_0_high	init vector 0 high
iv_0_low	init vector 0 low
iv_1_high	init vector 1 high
iv_1_low	init vector 1 low
key_0_high	key 0 high
key_0_low	key 0 low
key_1_high	key 1 high
key_1_low	key 1 low
key_2_high	key 2 high
key_2_low	key 2 low
key_3_high	key 3 high
key_3_low	key 3 low
gcmccmctxs[8]	GCM or CCM mode context switch
gcmctxs[8]	GCM mode context switch

## Structure cau\_parameter\_struct

**Table 3-77. Structure cau\_parameter\_struct**

Member name	Function description
alg_dir	algorithm directory
*key	key
key_size	key size in bytes
*iv	initialization vector
iv_size	iv size in bytes
*input	input data
in_length	input data length in bytes
*aad	additional authentication data
aad_size	aad size

## cau\_deinit

The description of cau\_deinit is shown as below:

**Table 3-78. Function cau\_deinit**

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

Example:

```
/* reset the CAU peripheral */
```

```
cau_deinit();
```

## cau\_struct\_para\_init

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

**Table 3-79. Function cau\_struct\_para\_init**

Function name	cau_struct_para_init
Function prototype	void cau_struct_para_init(cau_parameter_struct *cau_parameter);
Function descriptions	initialize the CAU encrypt and decrypt parameter struct with the default

	values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>cau_parameter</b>	structure for encrypt and decrypt parameters, refer to structure <a href="#">Table 3-77. Structure cau_parameter_struct</a>
<b>Return value</b>	
-	-

Example:

```
cau_parameter_struct text;

/* initialize CAU encrypt and decrypt parameter struct */
cau_struct_para_init(&text);
```

### cau\_key\_struct\_para\_init

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

**Table 3-80. Function cau\_key\_struct\_para\_init**

<b>Function name</b>	cau_key_struct_para_init
<b>Function prototype</b>	void cau_key_struct_para_init(cau_key_parameter_struct *key_initpara);
<b>Function descriptions</b>	initialize the key parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>key_initpara</b>	structure for keys initialization of the cau, refer to structure <a href="#">Table 3-74. Structure cau_key_parameter_struct</a>
<b>Return value</b>	
-	-

Example:

```
/* initialize the key parameter struct */
cau_key_parameter_struct key_initpara;
cau_key_struct_para_init(&key_initpara);
```

### cau\_iv\_struct\_para\_init

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

Table 3-81. Function cau\_iv\_struct\_para\_init

Function name	cau_iv_struct_para_init
Function prototype	void cau_iv_struct_para_init(cau_iv_parameter_struct *iv_initpara);
Function descriptions	initialize the vectors parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
iv_initpara	structure for vectors initialization of the cau, refer to structure <a href="#">Table 3-75. Structure cau_iv_parameter_struct</a>
Return value	
-	-

Example:

```
/* initialize the vectors parameter struct */
cau_iv_parameter_struct iv_initpara;
cau_iv_struct_para_init(&iv_initpara);
```

### cau\_context\_struct\_para\_init

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

Table 3-82. Function cau\_context\_struct\_para\_init

Function name	cau_context_struct_para_init
Function prototype	void cau_context_struct_para_init(cau_context_parameter_struct *cau_context);
Function descriptions	initialize the context parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
cau_context	structure for cau context swapping, refer to structure <a href="#">Table 3-76. Structure cau_context_parameter_struct</a>
Return value	
-	-

Example:

```
/* initialize the context parameter struct */
cau_context_parameter_struct context_initpara;
cau_context_struct_para_init(&context_initpara);
```

## cau\_enable

The description of cau\_enable is shown as below:

**Table 3-83. Function cau\_enable**

<b>Function name</b>	cau_enable
<b>Function prototype</b>	void cau_enable(void);
<b>Function descriptions</b>	enable the CAU peripheral
<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 CAU peripheral */
cau_enable();
```

## cau\_disable

The description of cau\_disable is shown as below:

**Table 3-84. Function cau\_disable**

<b>Function name</b>	cau_disable
<b>Function prototype</b>	void cau_disable(void);
<b>Function descriptions</b>	disable the CAU peripheral
<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 CAU peripheral */
cau_disable();
```

## cau\_dma\_enable

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

**Table 3-85. Function cau\_dma\_enable**

<b>Function name</b>	cau_dma_enable
<b>Function prototype</b>	void cau_dma_enable(uint32_t dma_req);
<b>Function descriptions</b>	enable the CAU DMA interface
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_req</b>	specify the CAU DMA transfer request to be enabled
CAU_DMA_INFIFO	DMA for incoming(Rx) data transfer
CAU_DMA_OUTFIFO	DMA for outgoing(Tx) data transfer
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the CAU DMA interface */
cau_dma_enable(CAU_DMA_INFIFO);
```

## cau\_dma\_disable

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

**Table 3-86. Function cau\_dma\_disable**

<b>Function name</b>	cau_dma_disable
<b>Function prototype</b>	void cau_dma_disable(uint32_t dma_req);
<b>Function descriptions</b>	disable the CAU DMA interface
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_req</b>	specify the CAU DMA transfer request to be disabled
CAU_DMA_INFIFO	DMA for incoming(Rx) data transfer
CAU_DMA_OUTFIFO	DMA for outgoing(Tx) data transfer
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the CAU DMA interface */
```



```
cau_dma_disable(CAU_DMA_INFIFO);
```

## cau\_init

The description of cau\_init is shown as below:

**Table 3-87. Function cau\_init**

<b>Function name</b>	cau_init
<b>Function prototype</b>	void cau_init(uint32_t algo_dir, uint32_t algo_mode, uint32_t swapping);
<b>Function descriptions</b>	initialize the CAU peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>algo_dir</b>	algorithm direction
CAU_ENCRYPT	encrypt
CAU_DECRYPT	decrypt
<b>Input parameter{in}</b>	
<b>algo_mode</b>	algorithm mode selection
CAU_MODE_TDES_ECB	TDES-ECB (3DES Electronic codebook)
CAU_MODE_TDES_CBC	TDES-CBC (3DES Cipher block chaining)
CAU_MODE_DES_ECB	DES-ECB (simple DES Electronic codebook)
CAU_MODE_DES_CBC	DES-CBC (simple DES Cipher block chaining)
CAU_MODE_AES_ECB	AES-ECB (AES Electronic codebook)
CAU_MODE_AES_CBC	AES-CBC (AES Cipher block chaining)
CAU_MODE_AES_CTR	AES-CTR (AES counter mode)
CAU_MODE_AES_KEY	AES decryption key preparation mode
CAU_MODE_AES_GCM	AES-GCM (AES Galois/counter mode)
CAU_MODE_AES_CCM	AES-CCM (AES combined cipher machine mode)
CAU_MODE_AES_CFB	AES-CFB (cipher feedback mode)
CAU_MODE_AES_OFB	AES-OFB (output feedback mode)
<b>Input parameter{in}</b>	
<b>swapping</b>	data swapping selection

<i>CAU_SWAPPING_32BIT</i>	no swapping
<i>CAU_SWAPPING_16BIT</i>	half-word swapping
<i>CAU_SWAPPING_8BIT</i>	bytes swapping
<i>CAU_SWAPPING_1BIT</i>	bit swapping
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the CAU peripheral */
```

```
cau_init(CAU_ENCRYPT, CAU_MODE_TDES_ECB, CAU_SWAPPING_32BIT);
```

### cau\_aes\_keysize\_config

The description of cau\_aes\_keysize\_config is shown as below:

**Table 3-88. Function cau\_aes\_keysize\_config**

<b>Function name</b>	cau_aes_keysize_config
<b>Function prototype</b>	void cau_aes_keysize_config(uint32_t key_size);
<b>Function descriptions</b>	configure key size if used AES algorithm
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>key_size</b>	key length selection when aes mode
<i>CAU_KEYSIZE_128BIT</i>	128 bit key length
<i>CAU_KEYSIZE_192BIT</i>	192 bit key length
<i>CAU_KEYSIZE_256BIT</i>	256 bit key length
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure key size if used AES algorithm */
```

```
cau_aes_keysize_config(CAU_KEYSIZE_128BIT);
```

### cau\_key\_init

The description of cau\_key\_init is shown as below:

Table 3-89. Function cau\_key\_init

Function name	cau_key_init
Function prototype	void cau_key_init(cau_key_parameter_struct* key_initpara);
Function descriptions	initialize the key parameters
Precondition	-
The called functions	-
Input parameter{in}	
key_initpara	structure for keys initialization of the cau, refer to structure <a href="#">Table 3-74. Structure cau_key_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize the key parameters */

cau_key_parameter_struct key_initpara;

key_initpara->key_0_high = 0x12345678;

key_initpara->key_0_low = 0x12345678;

key_initpara->key_1_high = 0x12345678;

key_initpara->key_1_low = 0x12345678;

key_initpara->key_2_high = 0x12345678;

key_initpara->key_2_low = 0x12345678;

key_initpara->key_3_high = 0x12345678;

key_initpara->key_4_low = 0x12345678;

cau_key_init(&key_initpara);

```

### cau\_iv\_init

The description of cau\_iv\_init is shown as below:

Table 3-90. Function cau\_iv\_init

Function name	cau_iv_init
Function prototype	void cau_iv_init(cau_iv_parameter_struct* iv_initpara);
Function descriptions	initialize the vectors parameters
Precondition	-
The called functions	-
Input parameter{in}	
iv_initpara	structure for vectors initialization of the cau, refer to structure <a href="#">Table 3-75.</a>

	<a href="#"><u>Structure cau_iv_parameter_struct</u></a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the vectors parameters */
cau_iv_parameter_struct iv_initpara;
iv_initpara->iv_0_high = 0x12345678;
iv_initpara->iv_0_low = 0x12345678;
iv_initpara->iv_1_high = 0x12345678;
iv_initpara->iv_1_low = 0x12345678;
cau_iv_init(&iv_initpara);
```

### cau\_phase\_config

The description of cau\_phase\_config is shown as below:

**Table 3-91. Function cau\_phase\_config**

<b>Function name</b>	cau_phase_config
<b>Function prototype</b>	void cau_phase_config(uint32_t phase);
<b>Function descriptions</b>	configure phase
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>phase</b>	gcm or ccm phase
CAU_PREPARE_PHASE	prepare phase
CAU_AAD_PHASE	AAD phase
CAU_ENCRYPT_DECRYPT_PHASE	encryption/decryption phase
CAU_TAG_PHASE	tag phase
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select prepare phase */
```

```
cau_phase_config(CAU_PREPARE_PHASE);
```

## cau\_fifo\_flush

The description of cau\_fifo\_flush is shown as below:

**Table 3-92. Function cau\_fifo\_flush**

<b>Function name</b>	cau_fifo_flush
<b>Function prototype</b>	void cau_fifo_flush(void);
<b>Function descriptions</b>	flush the IN and OUT FIFOs
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* flush the IN and OUT FIFOs */
```

```
cau_fifo_flush();
```

## cau\_enable\_state\_get

The description of cau\_enable\_state\_get is shown as below:

**Table 3-93. Function cau\_enable\_state\_get**

<b>Function name</b>	cau_enable_state_get
<b>Function prototype</b>	ControlStatus cau_enable_state_get(void);
<b>Function descriptions</b>	return whether CAU peripheral is enabled or disabled
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ControlStatus</b>	ENABLE or DISABLE

Example:

```
/* return whether CAU peripheral is enabled or disabled */
```

```
ControlStatus state = DISABLE;
```

```
state = cau_enable_state_get();
```

## cau\_data\_write

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

**Table 3-94. Function cau\_data\_write**

<b>Function name</b>	cau_data_write
<b>Function prototype</b>	void cau_data_write(uint32_t data);
<b>Function descriptions</b>	write data to the IN FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>data</b>	data to write: 0 - 0xFFFFFFFF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* write data to the IN FIFO */
```

```
cau_data_write(0x10);
```

## cau\_data\_read

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

**Table 3-95. Function cau\_data\_read**

<b>Function name</b>	cau_data_read
<b>Function prototype</b>	uint32_t cau_data_read(void);
<b>Function descriptions</b>	return the last data entered into the output FIFO
<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>	0x0 – 0xFFFFFFFF

Example:

```
/* return the last data entered into the output FIFO */
```

```
uint32_t data;
```

```
data = cau_data_read();
```

### cau\_context\_save

The description of cau\_context\_save is shown as below:

**Table 3-96. Function cau\_context\_save**

<b>Function name</b>	cau_context_save
<b>Function prototype</b>	void cau_context_save(cau_context_parameter_struct *cau_context, cau_key_parameter_struct* key_initpara);
<b>Function descriptions</b>	save context before context switching
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>key_initpara</b>	structure for keys initialization of the cau, refer to structure <a href="#">Table 3-74. Structure cau_key_parameter_struct</a>
<b>Output parameter{out}</b>	
<b>cau_context</b>	structure for cau context swapping, refer to structure <a href="#">Table 3-76. Structure cau_context_parameter_struct</a>
<b>Return value</b>	
-	-

Example:

```
cau_context_parameter_struct context;

cau_key_parameter_struct key;

cau_parameter_struct cau_parameter;

uint32_t keyaddr;

.....

keyaddr = (uint32_t)(cau_parameter->key);

cau_key_struct_para_init(&key);

key.key_1_high = __REV(*(uint32_t*)(keyaddr));

keyaddr += 4U;

key.key_1_low = __REV(*(uint32_t*)(keyaddr));

/* save context before context switching */

cau_context_save(&context, &key);
```

### cau\_context\_restore

The description of cau\_context\_restore is shown as below:

Table 3-97. Function cau\_context\_restore

Function name	cau_context_restore
Function prototype	void cau_context_restore(cau_context_parameter_struct *cau_context);
Function descriptions	restore context after context switching
Precondition	-
The called functions	-
Input parameter{in}	
cau_context	structure for cau context swapping, refer to structure <a href="#">Table 3-76. Structure cau_context_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
cau_context_parameter_struct context;
```

```
.....
```

```
cau_context_save(&context, &key);
```

```
.....
```

```
/* restore context after context switching */
```

```
cau_context_restore(&context);
```

### cau\_aes\_ecb

The description of cau\_aes\_ecb is shown as below:

Table 3-98. Function cau\_aes\_ecb

Function name	cau_aes_ecb
Function prototype	ErrStatus cau_aes_ecb(cau_parameter_struct *cau_parameter, uint8_t *output);
Function descriptions	encrypt and decrypt using AES in ECB mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	structure for encrypt and decrypt parameters, refer to structure <a href="#">Table 3-77. Structure cau_parameter_struct</a>
Output parameter{out}	
output	pointer to the returned buffer
Return value	
ErrStatus	SUCCESS or ERROR

Example:



```

cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

key_addr = key_select[i];

key_size = keysize[i];

text.alg_dir = CAU_ENCRYPT;

text.key = key_addr;

text.key_size = key_size;

text.input = plaintext;

text.in_length = TEXT_SIZE;

/* encryption in ECB mode */

status = cau_aes_ecb(&text, encrypt_result);

```

### cau\_aes\_cbc

The description of cau\_aes\_cbc is shown as below:

**Table 3-99. Function cau\_aes\_cbc**

Function name	cau_aes_cbc
Function prototype	ErrStatus cau_aes_cbc(cau_parameter_struct *cau_parameter, uint8_t *output);
Function descriptions	encrypt and decrypt using AES in CBC mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	structure for encrypt and decrypt parameters, refer to structure <a href="#">Table 3-77. Structure cau_parameter_struct</a>
Output parameter{out}	
output	pointer to the returned buffer
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```

cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

```

```

.....

key_addr = key_select[i];

key_size = keysize[i];

text.alg_dir = CAU_ENCRYPT;

text.key = key_addr;

text.key_size = key_size;

text.iv = vectors;

text.input = plaintext;

text.in_length = TEXT_SIZE;

/* encryption in CBC mode */

status = cau_aes_cbc(&text, encrypt_result);

```

### cau\_aes\_ctr

The description of cau\_aes\_ctr is shown as below:

**Table 3-100. Function cau\_aes\_ctr**

<b>Function name</b>	cau_aes_ctr
<b>Function prototype</b>	ErrStatus cau_aes_ctr(cau_parameter_struct *cau_parameter, uint8_t *output);
<b>Function descriptions</b>	encrypt and decrypt using AES in CTR mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cau_parameter</b>	structure for encrypt and decrypt parameters, refer to structure <a href="#">Table 3-77. Structure cau_parameter_struct</a>
<b>Output parameter{out}</b>	
<b>output</b>	pointer to the returned buffer
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```

cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

key_addr = key_select[i];

```

```

key_size = keysize[i];

text.alg_dir = CAU_ENCRYPT;

text.key = key_addr;

text.key_size = key_size;

text.iv = vectors;

text.input = plaintext;

text.in_length = TEXT_SIZE;

/* encryption in CTR mode */

status = cau_aes_ctr(&text, encrypt_result);

```

### cau\_aes\_cfb

The description of cau\_aes\_cfb is shown as below:

**Table 3-101. Function cau\_aes\_cfb**

<b>Function name</b>	cau_aes_cfb
<b>Function prototype</b>	ErrStatus cau_aes_cfb(cau_parameter_struct *cau_parameter, uint8_t *output);
<b>Function descriptions</b>	encrypt and decrypt using AES in CFB mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cau_parameter</b>	structure for encrypt and decrypt parameters, refer to structure <a href="#">Table 3-77. Structure cau_parameter_struct</a>
<b>Output parameter{out}</b>	
<b>output</b>	pointer to the returned buffer
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```

cau_parameter_struct cau_cfb_parameter;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

/* encryption in CFB mode */

cau_cfb_parameter.alg_dir = CAU_ENCRYPT;

cau_cfb_parameter.key = (uint8_t *)key_128;

```

```

cau_cfb_parameter.key_size = KEY_SIZE;

cau_cfb_parameter.iv = (uint8_t *)vectors;

cau_cfb_parameter.iv_size = IV_SIZE;

cau_cfb_parameter.input = (uint8_t *)plaintext;

cau_cfb_parameter.in_length = PLAINTEXT_SIZE;

status = cau_aes_cfb(&cau_cfb_parameter, encrypt_result);

```

## cau\_aes\_ofb

The description of cau\_aes\_ofb is shown as below:

**Table 3-102. Function cau\_aes\_ofb**

<b>Function name</b>	cau_aes_ofb
<b>Function prototype</b>	ErrStatus cau_aes_ofb(cau_parameter_struct *cau_parameter, uint8_t *output);
<b>Function descriptions</b>	encrypt and decrypt using AES in OFB mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cau_parameter</b>	structure for encrypt and decrypt parameters, refer to structure <a href="#">Table 3-77. Structure cau_parameter_struct</a>
<b>Output parameter{out}</b>	
<b>output</b>	pointer to the returned buffer
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```

cau_parameter_struct cau_ofb_parameter;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

/* encryption in OFB mode */

cau_ofb_parameter.alg_dir = CAU_ENCRYPT;

cau_ofb_parameter.key = (uint8_t *)key_128;

cau_ofb_parameter.key_size = KEY_SIZE;

cau_ofb_parameter.iv = (uint8_t *)vectors;

cau_ofb_parameter.iv_size = IV_SIZE;

```

```

cau_ofb_parameter.input = (uint8_t *)plaintext;

cau_ofb_parameter.in_length = PLAINTEXT_SIZE;

status = cau_aes_ofb(&cau_ofb_parameter, encrypt_result);

```

### cau\_aes\_gcm

The description of cau\_aes\_gcm is shown as below:

**Table 3-103. Function cau\_aes\_gcm**

<b>Function name</b>	cau_aes_gcm
<b>Function prototype</b>	ErrStatus cau_aes_gcm(cau_parameter_struct *cau_parameter, uint8_t *output, uint8_t *tag);
<b>Function descriptions</b>	encrypt and decrypt using AES in GCM mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cau_parameter</b>	structure for encrypt and decrypt parameters, refer to structure <a href="#">Table 3-77. Structure cau_parameter_struct</a>
<b>Output parameter{out}</b>	
<b>output</b>	pointer to the returned buffer
<b>Output parameter{out}</b>	
<b>tag</b>	pointer to the returned tag buffer
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```

cau_parameter_struct cau_gcm_parameter;

uint8_t encrypt_result[TEXT_SIZE];

uint8_t gcm_tag[GCM_TAG_SIZE];

ErrStatus status;

.....

/* encryption in GCM mode */

cau_gcm_parameter.alg_dir = CAU_ENCRYPT;

cau_gcm_parameter.key = (uint8_t *)key_128;

cau_gcm_parameter.key_size = KEY_SIZE;

cau_gcm_parameter.iv = (uint8_t *)vectors;

cau_gcm_parameter.iv_size = IV_SIZE;

cau_gcm_parameter.input = (uint8_t *)plaintext;

```

```

cau_gcm_parameter.in_length = PLAINTEXT_SIZE;

cau_gcm_parameter.aad = (uint8_t *)aadmessage;

cau_gcm_parameter.aad_size = AAD_SIZE;

status = cau_aes_gcm(&cau_gcm_parameter, encrypt_result, gcm_tag);

```

### cau\_aes\_ccm

The description of cau\_aes\_ccm is shown as below:

**Table 3-104. Function cau\_aes\_ccm**

Function name	cau_aes_ccm
Function prototype	ErrStatus cau_aes_ccm(cau_parameter_struct *cau_parameter, uint8_t *output, uint8_t tag[], uint32_t tag_size, uint8_t aad_buf[]);
Function descriptions	encrypt and decrypt using AES in CCM mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	structure for encrypt and decrypt parameters, refer to structure <a href="#">Table 3-77. Structure cau_parameter_struct</a>
Input parameter{in}	
tag_size	tag size (in bytes)
Output parameter{out}	
output	pointer to the returned buffer
Output parameter{out}	
tag	pointer to the returned tag buffer
Output parameter{out}	
aad_buf	pointer to the user buffer used when formatting aad block
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```

cau_parameter_struct cau_ccm_parameter;

uint8_t encrypt_result[TEXT_SIZE];

uint8_t ccm_tag[CCM_TAG_SIZE];

uint8_t aad_buf[AAD_SIZE + 21];

ErrStatus status;

.....

/* encryption in CCM mode */

cau_ccm_parameter.alg_dir = CAU_ENCRYPT;

```

```

cau_ccm_parameter.key = (uint8_t *)ccm_key_128;

cau_ccm_parameter.key_size = KEY_SIZE;

cau_ccm_parameter.iv = (uint8_t *)ccm_vectors;

cau_ccm_parameter.iv_size = CCM_IV_SIZE;

cau_ccm_parameter.input = (uint8_t *)plaintext;

cau_ccm_parameter.in_length = PLAINTEXT_SIZE;

cau_ccm_parameter.aad = (uint8_t *)aadmessage;

cau_ccm_parameter.aad_size = AAD_SIZE;

status = cau_aes_ccm(&cau_ccm_parameter, encrypt_result, ccm_tag, CCM_TAG_SIZE,
(uint8_t *)aad_buf);

```

### cau\_tdes\_ecb

The description of cau\_tdes\_ecb is shown as below:

**Table 3-105. Function cau\_tdes\_ecb**

Function name	cau_tdes_ecb
Function prototype	ErrStatus cau_tdes_ecb(cau_parameter_struct *cau_parameter, uint8_t *output);
Function descriptions	encrypt and decrypt using TDES in ECB mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	structure for encrypt and decrypt parameters, refer to structure <a href="#">Table 3-77. Structure cau_parameter_struct</a>
Output parameter{out}	
output	pointer to the returned buffer
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```

cau_parameter_struct text;

uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

text.alg_dir = CAU_ENCRYPT;

text.key = tdes_key;

```

```

text.input = plaintext;

text.in_length = DATA_SIZE;

/* encryption in ECB mode */

status = cau_tdes_ecb(&text, encrypt_result);

```

### cau\_tdes\_cbc

The description of cau\_tdes\_cbc is shown as below:

**Table 3-106. Function cau\_tdes\_cbc**

<b>Function name</b>	cau_tdes_cbc
<b>Function prototype</b>	ErrStatus cau_tdes_cbc(cau_parameter_struct *cau_parameter, uint8_t *output);
<b>Function descriptions</b>	encrypt and decrypt using TDES in CBC mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cau_parameter</b>	structure for encrypt and decrypt parameters, refer to structure <a href="#">Table 3-77. Structure cau_parameter_struct</a>
<b>Output parameter{out}</b>	
<b>output</b>	pointer to the returned buffer
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```

cau_parameter_struct text;

uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

text.alg_dir = CAU_ENCRYPT;

text.key = tdes_key;

text.iv = vectors;

text.input = plaintext;

text.in_length = DATA_SIZE;

/* encryption in CBC mode */

status = cau_tdes_cbc(&text, encrypt_result);

```



## cau\_des\_ecb

The description of cau\_des\_ecb is shown as below:

**Table 3-107. Function cau\_des\_ecb**

<b>Function name</b>	cau_des_ecb
<b>Function prototype</b>	ErrStatus cau_des_ecb(cau_parameter_struct *cau_parameter, uint8_t *output);
<b>Function descriptions</b>	encrypt and decrypt using DES in ECB mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cau_parameter</b>	structure for encrypt and decrypt parameters, refer to structure <a href="#">Table 3-77. Structure cau_parameter_struct</a>
<b>Output parameter{out}</b>	
<b>output</b>	pointer to the returned buffer
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
cau_parameter_struct text;

uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

text.alg_dir = CAU_ENCRYPT;

text.key = des_key;

text.input = plaintext;

text.in_length = DATA_SIZE;

/* encryption in ECB mode */

status = cau_des_ecb(&text, encrypt_result);
```

## cau\_des\_cbc

The description of cau\_des\_cbc is shown as below:

**Table 3-108. Function cau\_des\_cbc**

<b>Function name</b>	cau_des_cbc
<b>Function prototype</b>	ErrStatus cau_des_cbc(cau_parameter_struct *cau_parameter, uint8_t *output);
<b>Function descriptions</b>	encrypt and decrypt using DES in CBC mode

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cau_parameter</b>	structure for encrypt and decrypt parameters, refer to structure <a href="#">Table 3-77. Structure cau_parameter_struct</a>
<b>Output parameter{out}</b>	
<b>output</b>	pointer to the returned buffer
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
cau_parameter_struct text;

uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

text.alg_dir = CAU_ENCRYPT;

text.key = des_key;

text.iv = vectors;

text.input = plaintext;

text.in_length = DATA_SIZE;

/* encryption in CBC mode */

status = cau_des_cbc(&text, encrypt_result);
```

### cau\_flag\_get

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

**Table 3-109. Function cau\_flag\_get**

<b>Function name</b>	cau_flag_get
<b>Function prototype</b>	FlagStatus cau_flag_get(uint32_t flag);
<b>Function descriptions</b>	get the CAU flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	CAU flag status
CAU_FLAG_INFIFO_EMPTY	input FIFO empty
CAU_FLAG_INFIFO_NOT_FULL	input FIFO is not full

CAU_FLAG_OUTFIFO_NO_EMPTY	output FIFO not empty
CAU_FLAG_OUTFIFO_FULL	output FIFO is full
CAU_FLAG_BUSY	the CAU core is busy
CAU_FLAG_INFIFO	input FIFO flag status
CAU_FLAG_OUTFIFO	output FIFO flag status
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the CAU flag status */
```

```
FlagStatus status = RESET;
```

```
status = cau_flag_get(CAU_FLAG_INFIFO_EMPTY);
```

### cau\_interrupt\_enable

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

**Table 3-110. Function cau\_interrupt\_enable**

<b>Function name</b>	cau_interrupt_enable
<b>Function prototype</b>	void cau_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable the CAU interrupts
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify the CAU interrupt source to be enabled
CAU_INT_INFIFO	input FIFO interrupt
CAU_INT_OUTFIFO	output FIFO interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable cau interrupt */
```

```
cau_interrupt_enable(CAU_INT_INFIFO);
```

### cau\_interrupt\_disable

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

Table 3-111. Function cau\_interrupt\_disable

<b>Function name</b>	cau_interrupt_disable
<b>Function prototype</b>	void cau_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable the CAU interrupts
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify the CAU interrupt source to be disabled
CAU_INT_INFIFO	input FIFO interrupt
CAU_INT_OUTFIFO	output FIFO interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable cau interrupt */
cau_interrupt_disable(CAU_INT_INFIFO);
```

### cau\_interrupt\_flag\_get

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

Table 3-112. Function cau\_interrupt\_flag\_get

<b>Function name</b>	cau_interrupt_flag_get
<b>Function prototype</b>	FlagStatus cau_interrupt_flag_get(uint32_t int_flag);
<b>Function descriptions</b>	get the interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	CAU interrupt flag
CAU_INT_FLAG_INFIFO	input FIFO interrupt
CAU_INT_FLAG_OUTFIFO	output FIFO interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the CAU interrupt flag status */
FlagStatus status = RESET;
```

```
status = cau_interrupt_flag_get(CAU_INT_FLAG_INFIFO);
```

## 3.5. 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 [3.5.1](#), the CMP firmware functions are introduced in chapter [3.5.2](#).

### 3.5.1. Descriptions of Peripheral registers

CMP registers are listed in the table shown as below:

**Table 3-113. CMP Registers**

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

### 3.5.2. Descriptions of Peripheral functions

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

**Table 3-114. CMP firmware function**

Function name	Function description
cmp_deinit	CMP deinit
cmp_mode_init	CMP mode init
cmp_noninverting_input_select	CMP noninverting input select
cmp_output_init	CMP output init
cmp_blanking_init	CMP output blanking function init
cmp_enable	enable CMP
cmp_disable	disable CMP
cmp_window_enable	enable the window mode
cmp_window_disable	disable the window mode
cmp_lock_enable	lock the CMP
cmp_voltage_scaler_enable	enable the voltage scaler
cmp_voltage_scaler_disable	disable the voltage scaler
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 3-115. 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 3-116. 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 3-115. 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 3-117. 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}	
cmp_periph	refer to enum <a href="#">Table 3-115. Enum cmp_enum</a>
Input parameter{in}	
operating_mode	operating mode
CMP_MODE_HIGHSP	high speed mode

<i>EED</i>	
<i>CMP_MODE_MIDDLE SPEED</i>	medium speed mode
<i>CMP_MODE_LOWSPE ED</i>	low speed mode
<b>Input parameter{in}</b>	
<b>inverting_input</b>	inverting input select
<i>CMP_INVERTING_INP UT_1_4VREFINT</i>	VREFINT *1/4 input
<i>CMP_INVERTING_INP UT_1_2VREFINT</i>	VREFINT *1/2 input
<i>CMP_INVERTING_INP UT_3_4VREFINT</i>	VREFINT *3/4 input
<i>CMP_INVERTING_INP UT_VREFINT</i>	VREFINT input
<i>CMP_INVERTING_INP UT_PA0_PA2</i>	PA0 for CMP0 or PA2 for CMP1 as inverting input
<i>CMP_INVERTING_INP UT_DAC0_OUT0</i>	PA4(DAC) input
<i>CMP_INVERTING_INP UT_PB3</i>	PB3 input only for CMP1
<b>Input parameter{in}</b>	
<b>output_hysteresis</b>	hysteresis level
<i>CMP_HYSTERESIS_N O</i>	output no hysteresis
<i>CMP_HYSTERESIS_L OW</i>	output low hysteresis
<i>CMP_HYSTERESIS_M IDDLE</i>	output middle hysteresis
<i>CMP_HYSTERESIS_HI GH</i>	output high hysteresis
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

## cmp\_noninverting\_input\_select

The description of cmp\_noninverting\_input\_select is shown as below:

**Table 3-118. Function cmp\_noninverting\_input\_select**

<b>Function name</b>	cmp_noninverting_input_select
<b>Function prototype</b>	void cmp_noninverting_input_select(cmp_enum cmp_periph, uint32_t noninverting_input)
<b>Function descriptions</b>	CMP noninverting input select
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-115. Enum cmp_enum</a>
<b>Input parameter{in}</b>	
<b>noninverting_input</b>	noninverting input select
CMP_NONINVERTING_INPUT_PA3	CMP noninverting input PA3 for CMP1
CMP_NONINVERTING_INPUT_PB4	CMP noninverting input PB4 for CMP1
CMP_NONINVERTING_INPUT_PB5	CMP noninverting input PB5 for CMP1
CMP_NONINVERTING_INPUT_PB6	CMP noninverting input PB6 for CMP1
CMP_NONINVERTING_INPUT_PB7	CMP noninverting input PB7 for CMP1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* selecte the noninverting input for CMP1*/
```

```
cmp_noninverting_input_select (CMP1, CMP_NONINVERTING_INPUT_PA3);
```

## cmp\_output\_init

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

**Table 3-119. Function cmp\_output\_init**

<b>Function name</b>	cmp_output_init
<b>Function prototype</b>	void cmp_output_init(cmp_enum cmp_periph, uint32_t output_selection, uint32_t output_polarity)
<b>Function descriptions</b>	CMP output init
<b>Precondition</b>	-



The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-115. Enum cmp_enum</a>
Input parameter{in}	
output_selction	CMP output select
CMP_OUTPUT_NONE	output no selection
CMP_OUTPUT_TIMER1_IC3	CMP output TIMER1_CH3 input capture
CMP_OUTPUT_TIMER2_IC0	CMP output TIMER2_CH0 input capture
Input parameter{in}	
output_polarity	CMP output polarity
CMP_OUTPUT_POLARITY_INVERTED	output is inverted
CMP_OUTPUT_POLARITY_NONINVERTED	output is not inverted
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

### cmp\_blanking\_init

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

**Table 3-120. Function cmp\_blanking\_init**

Function name	cmp_blanking_init
Function prototype	void cmp_blanking_init(cmp_enum cmp_periph, uint32_t blanking_source_selection);
Function descriptions	CMP output blanking function init
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-115. Enum cmp_enum</a>
Input parameter{in}	
blanking_source_selection	blanking source selection
CMP_BLANKING_NON	CMP no blanking source

<i>E</i>	
<i>CMP_BLANKING_TIM</i> <i>ER1_OC1</i>	CMP TIMER1_CH1 output compare signal selected as blanking source
<i>CMP_BLANKING_TIM</i> <i>ER2_OC1</i>	CMP TIMER2_CH1 output compare signal selected as blanking source
<i>CMP_BLANKING_TIM</i> <i>ER8_OC1</i>	CMP TIMER8_CH1 output compare signal selected as blanking source
<i>CMP_BLANKING_TIM</i> <i>ER11_OC1</i>	CMP TIMER11_CH1 output compare signal selected as blanking source
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize CMP0 blanking function */
cmp_blanking_init(CMP0, CMP_BLANKING_NONE);
```

### cmp\_enable

The description of cmp\_enable is shown as below:

**Table 3-121. Function cmp\_enable**

<b>Function name</b>	cmp_enable
<b>Function prototype</b>	void cmp_enable(cmp_enum cmp_periph);
<b>Function descriptions</b>	enable CMP
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-115. Enum cmp_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### cmp\_disable

The description of cmp\_disable is shown as below:

Table 3-122. Function cmp\_disable

Function name	cmp_disable
Function prototype	void cmp_disable(cmp_enum cmp_periph);
Function descriptions	disable CMP
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-115. Enum cmp_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### cmp\_window\_enable

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

Table 3-123. Function cmp\_window\_enable

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

Example:

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

### cmp\_window\_disable

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

Table 3-124. Function cmp\_window\_disable

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

Example:

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

### cmp\_lock\_enable

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

Table 3-125. Function cmp\_lock\_enable

Function name	cmp_lock_enable
Function prototype	void cmp_lock_enable(cmp_enum cmp_periph);
Function descriptions	lock the CMP
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-115. Enum cmp_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### cmp\_voltage\_scaler\_enable

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

Table 3-126. Function cmp\_voltage\_scaler\_enable

Function name	cmp_voltage_scaler_enable
Function prototype	void cmp_voltage_scaler_enable(cmp_enum cmp_periph);
Function descriptions	enable the voltage scaler
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-115. Enum cmp_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

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 3-127. Function cmp\_voltage\_scaler\_disable

Function name	cmp_voltage_scaler_disable
Function prototype	void cmp_voltage_scaler_disable(cmp_enum cmp_periph);
Function descriptions	disable the voltage scaler
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-115. Enum cmp_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

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 3-128. Function cmp\_scaler\_bridge\_enable

Function name	cmp_scaler_bridge_enable
Function prototype	void cmp_scaler_bridge_enable(cmp_enum cmp_periph);
Function descriptions	enable the scaler bridge
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-115. Enum cmp_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

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 3-129. Function cmp\_scaler\_bridge\_disable

Function name	cmp_scaler_bridge_disable
Function prototype	void cmp_scaler_bridge_disable(cmp_enum cmp_periph);
Function descriptions	disable the scaler bridge
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-115. Enum cmp_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

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 3-130. Function cmp\_output\_level\_get

Function name	cmp_output_level_get
Function prototype	uint32_t cmp_output_level_get(cmp_enum cmp_periph);
Function descriptions	get output level
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-115. Enum cmp_enum</a>
Output parameter{out}	
-	-
Return value	
uint32_t	the output level
CMP_OUTPUTLEVEL_HIGH	comparator output high
CMP_OUTPUTLEVEL_LOW	comparator output low

Example:

```
uint32_t level;

/* get CMP0 output level */

level = cmp_output_level_get(CMP0);
```

## 3.6. 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 [3.6.1](#), the CRC firmware functions are introduced in chapter [3.6.2](#).

### 3.6.1. Descriptions of Peripheral registers

CRC registers are listed in the table shown as below:

Table 3-131. CRC Registers

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

### 3.6.2. Descriptions of Peripheral functions

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

**Table 3-132. CRC firmware function**

Function name	Function description
crc_deinit	deinit CRC calculation unit
crc_reverse_output_data_enable	enable the reverse operation of output data
crc_reverse_output_data_disable	disable the reverse operation of output data
crc_data_register_reset	reset data register to the value of initialization data register
crc_data_register_read	read the data register
crc_free_data_register_read	read the free data register
crc_free_data_register_write	write the free data register
crc_init_data_register_write	write the initialization value register
crc_input_data_reverse_config	configure the CRC input data function
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 3-133. 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\_reverse\_output\_data\_enable

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



**Table 3-134. 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\_reverse\_output\_data\_disable**

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

**Table 3-135. Function crc\_reverse\_output\_data\_disable**

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

Example:

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

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

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

Table 3-136. Function `crc_data_register_reset`

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

Example:

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

### `crc_data_register_read`

The description of `crc_data_register_read` is shown as below:

Table 3-137. Function `crc_data_register_read`

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

Example:

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

### `crc_free_data_register_read`

The description of `crc_free_data_register_read` is shown as below:

Table 3-138. Function `crc_free_data_register_read`

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

Example:

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

### `crc_free_data_register_write`

The description of `crc_free_data_register_write` is shown as below:

Table 3-139. Function `crc_free_data_register_write`

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

Example:

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

### `crc_init_data_register_write`

The description of `crc_init_data_register_write` is shown as below:

Table 3-140. Function `crc_init_data_register_write`

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

Example:

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

### `crc_input_data_reverse_config`

The description of `crc_input_data_reverse_config` is shown as below:

Table 3-141. Function `crc_input_data_reverse_config`

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

Example:

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

```
crc_input_data_reverse_config(CRC_INPUT_DATA_WORD);
```

### crc\_polynomial\_size\_set

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

**Table 3-142. 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 3-143. 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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

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 3-144. Function crc\_single\_data\_calculate**

<b>Function name</b>	crc_single_data_calculate
<b>Function prototype</b>	uint32_t crc_single_data_calculate(uint32_t sdata, uint8_t data_format);
<b>Function descriptions</b>	CRC calculate single data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sdata</b>	specify input data
<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>	
<b>uint32_t</b>	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 3-145. 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
<i>INPUT_FORMAT_WORD</i>	input data in word format
<i>INPUT_FORMAT_HALFWORD</i>	input data in half-word format
<i>INPUT_FORMAT_BYTE</i>	input data in byte format
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	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);

```

## 3.7. CTC

The CTC unit trims the frequency of the IRC48M which is based on an external accurate reference signal source. It can adjust the calibration value to provide a precise IRC48M clock automatically or manually. The CTC registers are listed in chapter [3.7.1](#), the CTC firmware functions are introduced in chapter [3.7.2](#).

### 3.7.1. Descriptions of Peripheral registers

CTC registers are listed in the table shown as below:

**Table 3-146. CTC Registers**

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

### 3.7.2. Descriptions of Peripheral functions

CTC registers are listed in the table shown as below:

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

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

#### ctc\_deinit

The description of ctc\_deinit is shown as below:

**Table 3-148. Function ctc\_deinit**

Function name	ctc_deinit
---------------	------------



<b>Function prototype</b>	void ctc_deinit(void)
<b>Function descriptions</b>	reset CTC clock trim controller
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset CTC */
```

```
ctc_deinit();
```

### ctc\_counter\_enable

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

**Table 3-149. Function ctc\_counter\_enable**

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

Example:

```
/* enable CTC trim counter*/
```

```
ctc_counter_enable();
```

### ctc\_counter\_disable

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

**Table 3-150. Function ctc\_counter\_disable**

<b>Function name</b>	ctc_counter_disable
<b>Function prototype</b>	void ctc_counter_disable(void);

<b>Function descriptions</b>	disable CTC counter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
ctc_counter_disable();
```

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

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

**Table 3-151. Function ctc\_irc48m\_trim\_value\_config**

<b>Function name</b>	ctc_irc48m_trim_value_config
<b>Function prototype</b>	void ctc_irc48m_trim_value_config(uint8_t trim_value);
<b>Function descriptions</b>	configure the IRC48M trim value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>trim_value</b>	0~127
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

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

**Table 3-152. Function ctc\_software\_refsource\_pulse\_generate**

<b>Function name</b>	ctc_software_refsource_pulse_generate
<b>Function prototype</b>	void ctc_software_refsource_pulse_generate(void);
<b>Function descriptions</b>	generate software reference source sync pulse

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

Example:

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

```
ctc_software_refsource_pulse_generate();
```

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

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

**Table 3-153. Function ctc\_hardware\_trim\_mode\_config**

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

Example:

```
/* enable CTC hardware trim */
```

```
ctc_hardware_trim_mode_config(CTC_HARDWARE_TRIM_MODE_ENABLE);
```

### ctc\_refsource\_polarity\_config

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

Table 3-154. Function `ctc_refsource_polarity_config`

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

Example:

```
/* set reference source polarity */
ctc_refsource_polarity_config(CTC_REFSOURCE_POLARITY_RISING);
```

### **`ctc_refsource_signal_select`**

The description of `ctc_refsource_signal_select` is shown as below:

Table 3-155. Function `ctc_refsource_signal_select`

<b>Function name</b>	<code>ctc_refsource_signal_select</code>
<b>Function prototype</b>	<code>void ctc_refsource_signal_select(uint32_t refs);</code>
<b>Function descriptions</b>	select reference signal source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>refs</b>	reference signal source
<code>CTC_REFSOURCE_GPIO</code>	GPIO is selected
<code>CTC_REFSOURCE_LXTAL</code>	LXTAL is selected
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
ctc_refsource_signal_select(CTC_REFSOURCE_LXTAL);
```

### ctc\_refsource\_prescaler\_config

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

**Table 3-156. Function ctc\_refsource\_prescaler\_config**

<b>Function name</b>	ctc_refsource_prescaler_config
<b>Function prototype</b>	void ctc_refsource_prescaler_config(uint32_t prescaler);
<b>Function descriptions</b>	configure reference signal source prescaler
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>prescaler</b>	Prescaler factor
CTC_REFSOURCE_P SC_OFF	reference signal not divided
CTC_REFSOURCE_P SC_DIV2	reference signal divided by 2
CTC_REFSOURCE_P SC_DIV4	reference signal divided by 4
CTC_REFSOURCE_P SC_DIV8	reference signal divided by 8
CTC_REFSOURCE_P SC_DIV16	reference signal divided by 16
CTC_REFSOURCE_P SC_DIV32	reference signal divided by 32
CTC_REFSOURCE_P SC_DIV64	reference signal divided by 64
CTC_REFSOURCE_P SC_DIV128	reference signal divided by 128
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
ctc_refsource_prescaler_config(CTC_REFSOURCE_PSC_DIV2);
```

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

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

Table 3-157. Function `ctc_clock_limit_value_config`

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

Example:

```
/* configure clock trim base limit value */
ctc_clock_limit_value_config(0x1F);
```

### `ctc_counter_reload_value_config`

The description of `ctc_counter_reload_value_config` is shown as below:

Table 3-158. Function `ctc_counter_reload_value_config`

Function name	<code>ctc_counter_reload_value_config</code>
Function prototype	<code>void ctc_counter_reload_value_config(uint16_t reload_value);</code>
Function descriptions	configure CTC counter reload value
Precondition	-
The called functions	-
Input parameter{in}	
reload_value	0x0000 - 0xFFFF
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure CTC counter reload value */
ctc_counter_reload_value_config(0x00FF);
```

### `ctc_counter_capture_value_read`

The description of `ctc_counter_capture_value_read` is shown as below:

Table 3-159. Function `ctc_counter_capture_value_read`

<b>Function name</b>	<code>ctc_counter_capture_value_read</code>
<b>Function prototype</b>	<code>uint16_t ctc_counter_capture_value_read(void);</code>
<b>Function descriptions</b>	read CTC counter capture value when reference sync pulse occurred
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	the 16-bit CTC counter capture value (0x0000 - 0xFFFF)

Example:

```
/* read CTC counter capture value */

uint16_t ctc_value = 0;

ctc_value = ctc_counter_capture_value_read();
```

### **`ctc_counter_direction_read`**

The description of `ctc_counter_direction_read` is shown as below:

Table 3-160. Function `ctc_counter_direction_read`

<b>Function name</b>	<code>ctc_counter_direction_read</code>
<b>Function prototype</b>	<code>FlagStatus ctc_counter_direction_read(void);</code>
<b>Function descriptions</b>	read CTC trim counter direction when reference sync pulse occurred
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET(count-down) / RESET(count-up)

Example:

```
/* read ctc counter direction */

FlagStatus ctc_direction = SET;

ctc_direction = ctc_counter_direction_read();
```

## ctc\_counter\_reload\_value\_read

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

**Table 3-161. Function ctc\_counter\_reload\_value\_read**

<b>Function name</b>	ctc_counter_reload_value_read
<b>Function prototype</b>	uint16_t ctc_counter_reload_value_read(void);
<b>Function descriptions</b>	read CTC counter reload value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	Read 16-bit data of counter reload value (0x0000 - 0xFFFF)

Example:

```
/* read CTC counter reload value */
uint16_t ctc_reload_value = 0;
ctc_reload_value = ctc_counter_reload_value_read();
```

## ctc\_irc48m\_trim\_value\_read

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

**Table 3-162. Function ctc\_irc48m\_trim\_value\_read**

<b>Function name</b>	ctc_irc48m_trim_value_read
<b>Function prototype</b>	uint8_t ctc_irc48m_trim_value_read(void);
<b>Function descriptions</b>	read the IRC48M trim value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint8_t	the 7-bit IRC48M trim value (0-127)

Example:

```
/* read the IRC48M trim value */
uint8_t ctc_trim_value = 0;
```



```
ctc_trim_value = ctc_irc48m_trim_value_read();
```

## ctc\_flag\_get

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

**Table 3-163. Function ctc\_flag\_get**

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

Example:

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

```
FlagStatus state = ctc_flag_get(CTC_FLAG_CKOK);
```

## ctc\_flag\_clear

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

**Table 3-164. Function ctc\_flag\_clear**

<b>Function name</b>	ctc_flag_clear
<b>Function prototype</b>	void ctc_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear CTC flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	CTC status flag
CTC_FLAG_CKOK	clock trim OK interrupt flag
CTC_FLAG_CKWARN	clock trim warning interrupt flag

<i>CTC_FLAG_ERR</i>	error interrupt flag
<i>CTC_FLAG_EREFP</i>	expect reference interrupt flag
<i>CTC_FLAG_CKERR</i>	clock trim error bit
<i>CTC_FLAG_REFMIS</i>	reference sync pulse miss flag
<i>CTC_FLAG_TRIMERR</i>	trim value error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### ctc\_interrupt\_enable

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

**Table 3-165. Function ctc\_interrupt\_enable**

<b>Function name</b>	ctc_interrupt_enable
<b>Function prototype</b>	void ctc_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable the CTC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	CTC interrupt
<i>CTC_INT_CKOK</i>	clock trim OK interrupt
<i>CTC_INT_CKWARN</i>	clock trim warning interrupt
<i>CTC_INT_ERR</i>	error interrupt
<i>CTC_INT_EREFP</i>	expect reference interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable CTC clock trim OK interrupt */
ctc_interrupt_enable(CTC_INT_CKOK);
```

### ctc\_interrupt\_disable

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

Table 3-166. Function `ctc_interrupt_disable`

<b>Function name</b>	<code>ctc_interrupt_disable</code>
<b>Function prototype</b>	<code>void ctc_interrupt_disable(uint32_t interrupt);</code>
<b>Function descriptions</b>	disable the CTC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	CTC interrupt
<code>CTC_INT_CKOK</code>	clock trim OK interrupt
<code>CTC_INT_CKWARN</code>	clock trim warning interrupt
<code>CTC_INT_ERR</code>	error interrupt
<code>CTC_INT_EREFS</code>	expect reference interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable CTC clock trim OK interrupt */
```

```
ctc_interrupt_disable(CTC_INT_CKOK);
```

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

The description of `ctc_interrupt_flag_get` is shown as below:

Table 3-167. Function `ctc_interrupt_flag_get`

<b>Function name</b>	<code>ctc_interrupt_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus ctc_interrupt_flag_get(uint32_t int_flag);</code>
<b>Function descriptions</b>	get CTC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	CTC interrupt flag
<code>CTC_INT_FLAG_CKOK</code>	clock trim OK interrupt
<code>CTC_INT_FLAG_CKWARN</code>	clock trim warning interrupt
<code>CTC_INT_FLAG_ERR</code>	error interrupt
<code>CTC_INT_FLAG_EREFS</code>	expect reference interrupt
<code>CTC_INT_FLAG_CKEERR</code>	clock trim error bit interrupt
<code>CTC_INT_FLAG_REFMISS</code>	reference sync pulse miss interrupt

<i>CTC_INT_FLAG_TRIM ERR</i>	trim value error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

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

```
FlagStatus state = ctc_interrupt_flag_get(CTC_INT_FLAG_CKOK);
```

### ctc\_interrupt\_flag\_clear

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

**Table 3-168. Function ctc\_interrupt\_flag\_clear**

<b>Function name</b>	ctc_interrupt_flag_clear
<b>Function prototype</b>	void ctc_interrupt_flag_clear(uint32_t int_flag);
<b>Function descriptions</b>	clear CTC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	CTC interrupt flag
<i>CTC_INT_FLAG_CKOK</i>	clock trim OK interrupt
<i>CTC_INT_FLAG_CKWARN</i>	clock trim warning interrupt
<i>CTC_INT_FLAG_ERR</i>	error interrupt
<i>CTC_INT_FLAG_EREFP</i>	expect reference interrupt
<i>CTC_INT_FLAG_CKE RR</i>	clock trim error bit interrupt
<i>CTC_INT_FLAG_REF MISS</i>	reference sync pulse miss interrupt
<i>CTC_INT_FLAG_TRIM ERR</i>	trim value error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
ctc_interrupt_flag_clear(CTC_INT_FLAG_CKOK);
```

## 3.8. DBG

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

### 3.8.1. Descriptions of Peripheral registers

DBG registers are listed in the table shown as below:

**Table 3-169. DBG Registers**

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

### 3.8.2. Descriptions of Peripheral functions

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

**Table 3-170. 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 3-171. 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 (only available in GD32L235xx)
DBG_TIMER1_HOLD	hold TIMER1 counter when core is halted
DBG_TIMER2_HOLD	hold TIMER2 counter when core is halted
DBG_CAN_HOLD	hold CAN kept when core is halted (only available in GD32L235xx)
DBG_I2C0_HOLD	hold I2C0 smbus when core is halted
DBG_I2C1_HOLD	hold I2C1 smbus when core is halted
DBG_TIMER5_HOLD	hold TIMER5 counter when core is halted

Member name	Function description
DBG_TIMER6_HOLD	hold TIMER6 counter when core is halted
DBG_TIMER8_HOLD	hold TIMER8 counter when core is halted
DBG_TIMER11_HOLD	hold TIMER11 counter when core is halted
DBG_TIMER14_HOLD	hold TIMER14 counter when core is halted (only available in GD32L235xx)
DBG_TIMER40_HOLD	hold TIMER40 counter when core is halted (only available in GD32L235xx)
DBG_RTC_HOLD	hold RTC counter when core is halted
DBG_LPTIMER_HOLD	hold LPTIMER counter when core is halted(only available in GD32L233xx)
DBG_LPTIMER0_HOLD	hold LPTIMER0 counter when core is halted(only available in GD32L235xx)
DBG_I2C2_HOLD	hold I2C2 smbus when core is halted
DBG_LPTIMER1_HOLD	hold LPTIMER1 counter when core is halted (only available in GD32L235xx)

### dbg\_deinit

The description of dbg\_deinit is shown as below:

**Table 3-172. 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 3-173. Function dbg\_id\_get**

Function name	dbg_id_get
---------------	------------

<b>Function prototype</b>	uint32_t dbg_id_get(void);
<b>Function descriptions</b>	Read DBG_ID code register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	DBG_ID code (0-0xFFFFFFFF)

Example:

```
/* read DBG_ID code register */
```

```
uint32_t id_value = 0;
```

```
id_value = dbg_id_get();
```

### dbg\_low\_power\_enable

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

**Table 3-174. Function dbg\_low\_power\_enable**

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

Example:

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

```
dbg_low_power_enable(DBG_LOW_POWER_SLEEP);
```

## dbg\_low\_power\_disable

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

**Table 3-175. 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 3-176. 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 3-171. 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=1,2,5,6,8,11, hold TIMERx counter when core is halted
<i>DBG_I2Cx_HOLD</i>	x=0,1,2, hold I2Cx smbus when core is halted
<i>DBG_RTC_HOLD</i>	hold RTC counter when core is halted



<i>DBG_LPTIMER_HOLD</i>	hold LPTIMER 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_TIMER1_HOLD);
```

### dbg\_periph\_disable

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

**Table 3-177. 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 3-171. 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=1,2,5,6,8,11, hold TIMERx counter when core is halted
<i>DBG_I2Cx_HOLD</i>	x=0,1,2, hold I2Cx smbus when core is halted
<i>DBG_RTC_HOLD</i>	hold RTC counter when core is halted
<i>DBG_LPTIMER_HOLD</i>	hold LPTIMER 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_TIMER1_HOLD);
```

## 3.9. DAC

The Digital-to-analog converter converts 12-bit digital data to a voltage on the external pins. The DAC registers are listed in chapter [3.9.1](#) the DAC firmware functions are introduced in chapter [3.9.2](#).

### 3.9.1. Peripheral register description

DAC registers are listed in the table shown as below:

**Table 3-178. DAC Registers**

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

### 3.9.2. Descriptions of Peripheral functions

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

**Table 3-179. DAC firmware functions**

Function name	Function description
dac_deinit	deinitialize DAC
dac_enable	enable DAC
dac_disable	disable DAC
dac_dma_enable	enable DAC DMA function
dac_dma_disable	disable DAC DMA function
dac_gpio_connect_config	configure gpio connection
dac_output_buffer_enable	enable DAC output buffer
dac_output_buffer_disable	disable DAC output buffer
dac_output_value_get	get DAC output value
dac_data_set	set DAC data holding register value
dac_trigger_enable	enable DAC trigger
dac_trigger_disable	disable DAC trigger
dac_trigger_source_config	configure DAC trigger source
dac_software_trigger_enable	enable DAC software trigger
dac_wave_mode_config	configure DAC wave mode
dac_lfsr_noise_config	configure DAC LFSR noise mode
dac_triangle_noise_config	configure DAC triangle noise mode
dac_flag_get	get DAC flag
dac_flag_clear	clear DAC flag
dac_interrupt_enable	enable DAC interrupt
dac_interrupt_disable	disable DAC interrupt
dac_interrupt_flag_get	get DAC interrupt flag
dac_interrupt_flag_clear	clear DAC interrupt flag

## dac\_deinit

The description of dac\_deinit is shown as below:

**Table 3-180. Function dac\_deinit**

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

Example:

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

## dac\_enable

The description of dac\_enable is shown as below:

**Table 3-181. Function dac\_enable**

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

Example:

```
/* enable DAC0_OUT0 */
```

```
dac_enable(DAC0, DAC_OUT0);
```

### **dac\_disable**

The description of `dac_disable` is shown as below:

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

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

Example:

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

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

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

The description of `dac_dma_enable` is shown as below:

**Table 3-183. Function `dac_dma_enable`**

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

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

Example:

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

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

The description of `dac_dma_disable` is shown as below:

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

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

Example:

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

### **dac\_gpio\_connect\_config**

The description of `dac_gpio_connect_config` is shown as below:

**Table 3-185. Function `dac_gpio_connect_config`**

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

<b>Function descriptions</b>	configure GPIO connection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b>dac_out</b>	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0)
<b>Input parameter{in}</b>	
<b>gpio_connect</b>	DAC_OUTx connect GPIO mode
<i>PIN_PERIPHERAL</i>	DAC_OUTx connected to the external pin and on chip peripherals(CMP)
<i>PIN_PERIPHERAL_BUFFER</i>	Whether DAC_OUTx is connected to external pin and on chip peripherals(CMP) depends on the output buffer
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DAC0_OUT0 GPIO connection working in PIN_PERIPHERAL */
dac_gpio_connect_config (DAC0, DAC_OUT0, PIN_PERIPHERAL);
```

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

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

**Table 3-186. Function dac\_output\_buffer\_enable**

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

Example:

```
/* enable DAC0_OUT0 output buffer */
dac_output_buffer_enable(DAC0, DAC_OUT0);
```

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

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

**Table 3-187. Function dac\_output\_buffer\_disable**

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

Example:

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

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

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

**Table 3-188. Function dac\_output\_value\_get**

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

<b>dac_out</b>	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	DAC output data (0~4095)

Example:

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

```
uint16_t data = 0;
```

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

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

The description of `dac_data_set` is shown as below:

**Table 3-189. Function `dac_data_set`**

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

Example:

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



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

## dac\_trigger\_enable

The description of `dac_trigger_enable` is shown as below:

**Table 3-190. Function `dac_trigger_enable`**

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

Example:

```
/* enable DAC0_OUT0 trigger */
```

```
dac_trigger_enable(DAC0, DAC_OUT0);
```

## dac\_trigger\_disable

The description of `dac_trigger_disable` is shown as below:

**Table 3-191. Function `dac_trigger_disable`**

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

-	-
Return value	
-	-

Example:

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

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

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

The description of `dac_trigger_source_config` is shown as below:

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

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

Example:

```
/* configure DAC0_OUT0 trigger source */
```

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

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

The description of `dac_software_trigger_enable` is shown as below:

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

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

Example:

```
/* enable DAC0_OUT0 software trigger */
```

```
dac_software_trigger_enable(DAC0, DAC_OUT0);
```

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

The description of `dac_wave_mode_config` is shown as below:

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

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

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

Example:

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

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

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

The description of `dac_lfsr_noise_config` is shown as below:

**Table 3-195. Function `dac_lfsr_noise_config`**

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

-	-
---	---

Example:

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

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

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

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

**Table 3-196. Function dac\_triangle\_noise\_config**

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

Example:

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

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

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

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

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

Function name	dac_flag_get
Function prototype	FlagStatus dac_flag_get(uint32_t dac_periph, uint32_t flag);

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

Example:

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

```
FlagStatus flag;
```

```
flag = dac_flag_get(DAC0, DAC_FLAG_DDUDR0);
```

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

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

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

<b>Function name</b>	dac_flag_clear
<b>Function prototype</b>	void dac_flag_clear(uint32_t dac_periph, uint32_t flag);
<b>Function descriptions</b>	clear DAC flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b>flag</b>	DAC flag
<i>DAC_FLAG_DDUDR0</i>	DACx_OUT0 DMA underrun flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

The description of `dac_interrupt_enable` is shown as below:

**Table 3-199. Function `dac_interrupt_enable`**

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

Example:

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

```
dac_interrupt_enable (DAC0, DAC_INT_DDUDR0);
```

### **dac\_interrupt\_disable**

The description of `dac_interrupt_disable` is shown as below:

**Table 3-200. Function `dac_interrupt_disable`**

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

-	-
Return value	
-	-

Example:

```
/* disable DAC0 interrupt */
```

```
dac_interrupt_disable (DAC0, DAC_INT_DDUDR0);
```

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

The description of `dac_interrupt_flag_get` is shown as below:

**Table 3-201. Function `dac_interrupt_flag_get`**

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

Example:

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

```
FlagStatus flag;
```

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

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

The description of `dac_interrupt_flag_clear` is shown as below:

**Table 3-202. Function `dac_interrupt_flag_clear`**

<b>Function name</b>	<code>dac_interrupt_flag_clear</code>
<b>Function prototype</b>	<code>void dac_interrupt_flag_clear(uint32_t dac_periph, uint32_t int_flag);</code>
<b>Function descriptions</b>	clear DAC interrupt flag



<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b>int_flag</b>	DAC interrupt flag
<i>DAC_INT_FLAG_DDUDR0</i>	DACx_OUT0 DMA underrun interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

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

### 3.10.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

**Table 3-203. DMA Registers**

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

Registers	Descriptions
DMA_CHxMADDR (x=0..6)	Channel x memory base address register

DMAMUX registers are listed in the table shown as below:

**Table 3-204. DMAMUX Registers**

Registers	Descriptions
DMAMUX_RM_CHx CFG (x=0..6)	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..6)	Request generator channel x configuration register
DMAMUX_RG_INT F	Request generator channel interrupt flag register
DMAMUX_RG_INT C	Rquest generator channel interrupt flag clear register

### 3.10.2. Descriptions of Peripheral functions

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

**Table 3-205. 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

Function name	Function description
dma_periph_width_config	configure transfer data size of peripheral
dma_memory_increase_enable	enable next address increasement algorithm of memory
dma_memory_increase_disable	disable next address increasement algorithm of memory
dma_periph_increase_enable	enable next address increasement algorithm of peripheral
dma_periph_increase_disable	disable next address increasement algorithm of peripheral
dma_transfer_direction_config	configure the direction of data transfer on the channel
dma_flag_get	check DMA flag is set or not
dma_flag_clear	clear DMA a channel flag
dma_interrupt_enable	enable DMA interrupt
dma_interrupt_disable	disable DMA interrupt
dma_interrupt_flag_get	check DMA flag and interrupt enable bit is set or not
dma_interrupt_flag_clear	clear DMA a channel flag

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

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

Function name	Function description
dmamux_sync_struct_para_init	initialize the parameters of DMAMUX synchronization mode structure with the default values
dmamux_synchronization_init	initialize DMAMUX request multiplexer channel synchronization mode
dmamux_synchronization_enable	enable synchronization mode
dmamux_synchronization_disable	disable synchronization mode
dmamux_event_generation_enable	enable event generation
dmamux_event_generation_disable	disable event generation
dmamux_gen_struct_para_init	initialize the parameters of DMAMUX request generator structure with the default values
dmamux_request_generator_init	initialize DMAMUX request generator channel
dmamux_request_generator_channel_enable	enable DMAMUX request generator channel
dmamux_request_generator_channel_disable	disable DMAMUX request generator channel
dmamux_synchronization_polarity_config	configure synchronization input polarity
dmamux_request_forward_number_config	configure number of DMA requests to forward
dmamux_sync_id_config	configure synchronization input identification
dmamux_request_id_config	configure multiplexer input identification
dmamux_trigger_polarity_config	configure trigger input polarity
dmamux_request_generate_number_config	configure number of DMA requests to be generated
dmamux_trigger_id_config	configure trigger input identification

Function name	Function description
dmamux_flag_get	get DMAMUX flag
dmamux_flag_clear	clear DMAMUX flag
dmamux_interrupt_enable	enable DMAMUX interrupt
dmamux_interrupt_disable	disable DMAMUX interrupt
dmamux_interrupt_flag_get	get DMAMUX interrupt flag
dmamux_interrupt_flag_clear	clear DMAMUX interrupt flag

### Structure dma\_parameter\_struct

**Table 3-207. 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 3-208. 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 3-209. 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 3-210. Enum dmamux\_interrupt\_enum**

Member name	Function description
DMAMUX_INT_MU	DMAMUX request multiplexer channel 0 synchronization overrun interrupt

XCH0_SO	
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_MU XCH3_SO	DMAMUX request multiplexer channel 3 synchronization overrun interrupt
DMAMUX_INT_MU XCH4_SO	DMAMUX request multiplexer channel 4 synchronization overrun interrupt
DMAMUX_INT_MU XCH5_SO	DMAMUX request multiplexer channel 5 synchronization overrun interrupt
DMAMUX_INT_MU XCH6_SO	DMAMUX request multiplexer channel 6 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 3-211. 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_M UXCH3_SO	DMAMUX request multiplexer channel 3 synchronization overrun flag
DMAMUX_FLAG_M UXCH4_SO	DMAMUX request multiplexer channel 4 synchronization overrun flag
DMAMUX_FLAG_M UXCH5_SO	DMAMUX request multiplexer channel 5 synchronization overrun flag
DMAMUX_FLAG_M UXCH6_SO	DMAMUX request multiplexer channel 6 synchronization overrun flag
DMAMUX_FLAG_G ENCH0_TO	DMAMUX request generator channel 0 trigger overrun flag
DMAMUX_FLAG_G	DMAMUX request generator channel 1 trigger overrun flag

ENCH1_TO	
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 3-212. Enum dmamux\_interrupt\_flag\_enum**

Member name	Function description
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_MUXCH3_SO	DMAMUX request multiplexer channel 3 synchronization overrun interrupt flag
DMAMUX_INT_FLAG_MUXCH4_SO	DMAMUX request multiplexer channel 4 synchronization overrun interrupt flag
DMAMUX_INT_FLAG_MUXCH5_SO	DMAMUX request multiplexer channel 5 synchronization overrun interrupt flag
DMAMUX_INT_FLAG_MUXCH6_SO	DMAMUX request multiplexer channel 6 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

### Enum dma\_channel\_enum

**Table 3-213. Enum dma\_channel\_enum**

Member name	Function description
DMA_CH0	DMA Channel 0
DMA_CH1	DMA Channel 1
DMA_CH2	DMA Channel 2
DMA_CH3	DMA Channel 3
DMA_CH4	DMA Channel 4
DMA_CH5	DMA Channel 5

DMA_CH6	DMA Channel 6
---------	---------------

### Enum dmamux\_multiplexer\_channel\_enum

**Table 3-214. 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
DMAMUX_MUXCH 3	DMAMUX request multiplexer Channel3
DMAMUX_MUXCH 4	DMAMUX request multiplexer Channel4
DMAMUX_MUXCH 5	DMAMUX request multiplexer Channel5
DMAMUX_MUXCH 6	DMAMUX request multiplexer Channel6

### Enum dmamux\_generator\_channel\_enum

**Table 3-215. 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 3-216. 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..6)	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-217. Function dma\_struct\_para\_init**

<b>Function name</b>	dma_struct_para_init
<b>Function prototype</b>	void dma_struct_para_init(dma_parameter_struct* init_struct);
<b>Function descriptions</b>	initialize the parameters of DMA struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>init_struct</b>	the initialization data needed to initialize DMA channel, refer to <a href="#">Table 3-207. Structure dma_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

## dma\_init

The description of dma\_init is shown as below:

**Table 3-218. 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>	-
Input parameter{in}	
<b>channelx</b>	DMA channel



<i>DMA_CHx(x=0..6)</i>	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-207. 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 3-219. 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..6)</i>	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-220. 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..6)</i>	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-221. Function dma\_memory\_to\_memory\_enable**

<b>Function name</b>	dma_memory_to_memory_enable
<b>Function prototype</b>	void dma_memory_to_memory_enable(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..6)</i>	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-222. Function dma\_memory\_to\_memory\_disable**

<b>Function name</b>	dma_memory_to_memory_disable
<b>Function prototype</b>	void dma_memory_to_memory_disable(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..6)</i>	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-223. 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..6)</i>	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-224. 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..6)</i>	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-225. 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..6)</i>	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-226. 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..6)</i>	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-227. 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..6)</i>	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-228. 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..6)	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-229. 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..6)</i>	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-230. 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..6)</i>	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-231. 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..6)	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-232. 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..6)</i>	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-233. 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..6)</i>	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-234. Function dma\_periph\_increase\_enable

Function name	dma_periph_increase_enable
Function prototype	void dma_periph_increase_enable(dma_channel_enum channelx);
Function descriptions	enable next address increasement algorithm of peripheral
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 3-213. Enum dma_channel_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

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 3-235. Function dma\_periph\_increase\_disable

Function name	dma_periph_increase_disable
Function prototype	void dma_periph_increase_disable(dma_channel_enum channelx);
Function descriptions	disable next address increasement algorithm of peripheral
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 3-213. Enum dma_channel_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

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 3-236. Function dma\_transfer\_direction\_config

<b>Function name</b>	dma_transfer_direction_config
<b>Function prototype</b>	void dma_transfer_direction_config(dma_channel_enum channelx, uint32_t direction);
<b>Function descriptions</b>	configure the direction of data transfer on the channel
<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
DMA_CHx(x=0..6)	DMA channel selection, refer to <a href="#">Table 3-213. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>direction</b>	specify the direction of data transfer
DMA_PERIPHERAL_TO_MEMORY	read from peripheral and write to memory
DMA_MEMORY_TO_PERIPHERAL	read from memory and write to peripheral
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA channel0 transfer direction */
dma_transfer_direction_config(DMA_CH0, DMA_PERIPHERAL_TO_MEMORY);
```

## dma\_flag\_get

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

Table 3-237. Function dma\_flag\_get

<b>Function name</b>	dma_flag_get
<b>Function prototype</b>	FlagStatus dma_flag_get(dma_channel_enum channelx, uint32_t flag);
<b>Function descriptions</b>	check DMA flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
DMA_CHx(x=0..6)	DMA channel selection, refer to <a href="#">Table 3-213. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>flag</b>	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 3-238. 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..6)</i>	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-239. 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..6)</i>	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-240. 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..6)</i>	DMA channel selection, refer to <a href="#">Table 3-213. 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 3-241. 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..6)	DMA channel selection, refer to <a href="#">Table 3-213. 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_FTF);
}
```

## dma\_interrupt\_flag\_clear

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

**Table 3-242. 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..6)</i>	DMA channel selection, refer to <a href="#">Table 3-213. 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_FTF);
}

```

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

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

**Table 3-243. 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 3-208. 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 3-244. 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..6)	DMAMUX channel selection, refer to <a href="#">Table 3-214. 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 3-208. 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 3-245. 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..6)	DMAMUX channel selection, refer to <a href="#">Table 3-214. 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 3-246. 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..6)	DMAMUX channel selection, refer to <a href="#">Table 3-214. 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 3-247. Function dmamux\_event\_generation\_enable**

<b>Function name</b>	dmamux_event_generation_enable
<b>Function prototype</b>	void dmamux_event_generation_enable(dmamux_multiplexer_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..6)	DMAMUX channel selection, refer to <a href="#">Table 3-214. Enum dmamux_multiplexer_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 3-248. Function dmamux\_event\_generation\_disable**

<b>Function name</b>	dmamux_event_generation_disable
<b>Function prototype</b>	void dmamux_event_generation_disable(dmamux_multiplexer_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..6)	DMAMUX channel selection, refer to <a href="#">Table 3-214. Enum dmamux_multiplexer_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 3-249. 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 3-209. 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 3-250. 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 3-215. 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 3-209. 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 3-251. 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 3-215. 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 3-252. 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 3-215. 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 3-253. 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..6)	DMAMUX channel selection, refer to <a href="#">Table 3-214. 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 3-254. 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..6)	DMAMUX channel selection, refer to <a href="#">Table 3-214. 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 3-255. 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..6)	DMAMUX channel selection, refer to <a href="#">Table 3-214. 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_EXTI</i> 13	synchronization input is EXTI13
<i>DMAMUX_SYNC_EXTI</i> 14	synchronization input is EXTI14
<i>DMAMUX_SYNC_EXTI</i> 15	synchronization input is EXTI15
<i>DMAMUX_SYNC_EVT</i> 0_OUT	synchronization input is Evt0_out
<i>DMAMUX_SYNC_EVT</i> 1_OUT	synchronization input is Evt1_out
<i>DMAMUX_SYNC_EVT</i> 2_OUT	synchronization input is Evt2_out
<i>DMAMUX_SYNC_EVT</i> 3_OUT	synchronization input is Evt3_out
<i>DMAMUX_SYNC_TIMER</i> 11_CH0_O	synchronization input is TIMER11_CH0_O
<i>DMAMUX_SYNC_EXTI</i> 0	synchronization input is EXTI0
<i>DMAMUX_SYNC_EXTI</i> 1	synchronization input is EXTI1
<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 3-256. Function dmamux\_request\_id\_config**

<b>Function name</b>	dmamux_request_id_config
<b>Function prototype</b>	void dmamux_request_id_config(dmamux_multiplexer_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</i> (x=0..6)	DMAMUX channel selection, refer to <a href="#">Table 3-214. Enum dmamux_multiplexer_channel_enum</a>



Input parameter{in}	
id	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>DMA_REQUEST_GENERATOR2</i>	DMAMUX request generator 2
<i>DMA_REQUEST_GENERATOR3</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_I2C2_RX</i>	DMAMUX I2C2 RX request
<i>DMA_REQUEST_I2C2_TX</i>	DMAMUX I2C2 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_TIMER1_CH0</i>	DMAMUX TIMER1 CH0 request
<i>DMA_REQUEST_TIMER1_CH1</i>	DMAMUX TIMER1 CH1 request
<i>DMA_REQUEST_TIMER1_CH2</i>	DMAMUX TIMER1 CH2 request
<i>DMA_REQUEST_TIMER1_CH3</i>	DMAMUX TIMER1 CH3 request
<i>DMA_REQUEST_TIMER1_UP</i>	DMAMUX TIMER1 UP request

<i>DMA_REQUEST_TIME</i> <i>R2_CH0</i>	DMAMUX TIMER2 CH0 request
<i>DMA_REQUEST_TIME</i> <i>R2_CH1</i>	DMAMUX TIMER2 CH1 request
<i>DMA_REQUEST_TIME</i> <i>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>R5_UP</i>	DMAMUX TIMER5 UP request
<i>DMA_REQUEST_TIME</i> <i>R6_UP</i>	DMAMUX TIMER6 UP request
<i>DMA_REQUEST_CAU</i> <i>_IN</i>	DMAMUX CAU IN request
<i>DMA_REQUEST_CAU</i> <i>_OUT</i>	DMAMUX CAU OUT 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>T3_RX</i>	DMAMUX UART3 RX request
<i>DMA_REQUEST_UAR</i> <i>T3_TX</i>	DMAMUX UART3 TX request
<i>DMA_REQUEST_UAR</i> <i>T4_RX</i>	DMAMUX UART4 RX request
<i>DMA_REQUEST_UAR</i> <i>T4_TX</i>	DMAMUX UART4 TX request
<i>DMA_REQUEST_LPU</i> <i>ART_RX</i>	DMAMUX LPUART RX request, only for GD32L233
<i>DMA_REQUEST_LPU</i> <i>ART_TX</i>	DMAMUX LPUART TX request, only for GD32L233
<i>DMA_REQUEST_LPU</i> <i>ART0_RX</i>	DMAMUX LPUART0 RX request, only for GD32L235
<i>DMA_REQUEST_LPU</i>	DMAMUX LPUART0 TX request, only for GD32L235

ART0_TX	
DMA_REQUEST_LPU ART1_RX	DMAMUX LPUART1 RX request, only for GD32L235
DMA_REQUEST_LPU ART1_TX	DMAMUX LPUART1 TX request, only for GD32L235
DMA_REQUEST_TIME R0_CH0	DMAMUX TIMER0 CH0 request, only for GD32L235
DMA_REQUEST_TIME R0_CH1	DMAMUX TIMER0 CH1 request, only for GD32L235
DMA_REQUEST_TIME R0_CH2	DMAMUX TIMER0 CH2 request, only for GD32L235
DMA_REQUEST_TIME R0_CH3	DMAMUX TIMER0 CH3 request, only for GD32L235
DMA_REQUEST_TIME R0_UP	DMAMUX TIMER0 UP request, only for GD32L235
DMA_REQUEST_TIME R0_COM	DMAMUX TIMER0 COM request, only for GD32L235
DMA_REQUEST_TIME R14_CH0	DMAMUX TIMER14 CH0 request, only for GD32L235
DMA_REQUEST_TIME R14_CH1	DMAMUX TIMER14 CH1 request, only for GD32L235
DMA_REQUEST_TIME R14_TRIG	DMAMUX TIMER14 TRIG request, only for GD32L235
DMA_REQUEST_TIME R14_UP	DMAMUX TIMER14 UP request, only for GD32L235
DMA_REQUEST_TIME R14_COM	DMAMUX TIMER14 COM request, only for GD32L235
DMA_REQUEST_TIME R40_CH0	DMAMUX TIMER40 CH0 request, only for GD32L235
DMA_REQUEST_TIME R40_CH1	DMAMUX TIMER40 CH1 request, only for GD32L235
DMA_REQUEST_TIME R40_TRIG	DMAMUX TIMER40 TRIG request, only for GD32L235
DMA_REQUEST_TIME R40_UP	DMAMUX TIMER40 UP request, only for GD32L235
DMA_REQUEST_TIME R40_COM	DMAMUX TIMER40 COM request, only for GD32L235
Output parameter{out}	
-	-
Return value	
-	-

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 3-257. 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 3-215. 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 3-258. 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_e

	num 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 3-215. 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>	
-	-
<b>Return value</b>	
-	-

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 3-259. 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 3-215. Enum dmamux_generator_channel_enum</a>
<b>Input parameter{in}</b>	
<b>id</b>	trigger input identification
DMAMUX_TRIGGER_EXTI0	trigger input is EXTI0
DMAMUX_TRIGGER_EXTI1	trigger input is EXTI1
DMAMUX_TRIGGER_EXTI2	trigger input is EXTI2
DMAMUX_TRIGGER_EXTI3	trigger input is EXTI3

<i>EXTI3</i>	
<i>DMAMUX_TRIGGER_</i> <i>EXTI4</i>	trigger input is EXTI4
<i>DMAMUX_TRIGGER_</i> <i>EXTI5</i>	trigger input is EXTI5
<i>DMAMUX_TRIGGER_</i> <i>EXTI6</i>	trigger input is EXTI6
<i>DMAMUX_TRIGGER_</i> <i>EXTI7</i>	trigger input is EXTI7
<i>DMAMUX_TRIGGER_</i> <i>EXTI8</i>	trigger input is EXTI8
<i>DMAMUX_TRIGGER_</i> <i>EXTI9</i>	trigger input is EXTI9
<i>DMAMUX_TRIGGER_</i> <i>EXTI10</i>	trigger input is EXTI10
<i>DMAMUX_TRIGGER_</i> <i>EXTI11</i>	trigger input is EXTI11
<i>DMAMUX_TRIGGER_</i> <i>EXTI12</i>	trigger input is EXTI12
<i>DMAMUX_TRIGGER_</i> <i>EXTI13</i>	trigger input is EXTI13
<i>DMAMUX_TRIGGER_</i> <i>EXTI14</i>	trigger input is EXTI14
<i>DMAMUX_TRIGGER_</i> <i>EXTI15</i>	trigger input is EXTI15
<i>DMAMUX_TRIGGER_</i> <i>EVT0_OUT</i>	trigger input is Evt0_out
<i>DMAMUX_TRIGGER_</i> <i>EVT1_OUT</i>	trigger input is Evt1_out
<i>DMAMUX_TRIGGER_</i> <i>EVT2_OUT</i>	trigger input is Evt2_out
<i>DMAMUX_TRIGGER_</i> <i>EVT3_OUT</i>	trigger input is Evt3_out
<i>DMAMUX_TRIGGER_</i> <i>TIMER11_CH0_O</i>	trigger input is TIMER11_CH0_O
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-260. 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>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flag type, refer to <a href="#">Table 3-211. Enum dmamux_flag_enum</a>
<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 3-261. 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 3-211. Enum dmamux_flag_enum</a>
<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 3-262. 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, refer to <a href="#">Table 3-210. Enum dmamux_interrupt_enum</a>
<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 3-263. Function dmamux\_interrupt\_disable**

<b>Function name</b>	dmamux_interrupt_disable
<b>Function prototype</b>	void dmamux_interrupt_disable(dmamux_interrupt_enum interrupt);
<b>Function descriptions</b>	disable DMAMUX interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which interrupt to disable, refer to <a href="#">Table 3-210. Enum dmamux_interrupt_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-264. Function dmamux\_interrupt\_flag\_get**

<b>Function name</b>	dmamux_interrupt_flag_get
<b>Function prototype</b>	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 3-212. Enum dmamux_interrupt_flag_enum</a>
<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 3-265. Function dmamux\_interrupt\_flag\_clear**

<b>Function name</b>	dmamux_interrupt_flag_clear
<b>Function prototype</b>	FlagStatus dmamux_interrupt_flag_clear(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 3-212. Enum dmamux_interrupt_flag_enum</a>
<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);
}
```

## 3.11. EXTI

EXTI is the interrupt/event controller in the MCU. It contains up to 30 independent edge detectors (for GD32L233xx devices) or 32 independent edge detectors (for GD32L235xx devices) and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter [3.11.1](#), the EXTI firmware functions are introduced in chapter [3.11.2](#).

### 3.11.1. Descriptions of Peripheral registers

EXTI registers are listed in the table shown as below:

**Table 3-266. EXTI Registers**

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

### 3.11.2. Descriptions of Peripheral functions

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

**Table 3-267. 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 3-268. exti\_line\_enum for GD32L233xx devices**

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
EXTI_20	EXTI line 20
EXTI_21	EXTI line 21
EXTI_22	EXTI line 22
EXTI_23	EXTI line 23
EXTI_24	EXTI line 24
EXTI_25	EXTI line 25
EXTI_26	EXTI line 26
EXTI_27	EXTI line 27
EXTI_28	EXTI line 28
EXTI_29	EXTI line 29

**Table 3-269. exti\_line\_enum for GD32L235xx devices**

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

enum name	Function description
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
EXTI_20	EXTI line 20
EXTI_21	EXTI line 21
EXTI_22	EXTI line 22
EXTI_23	EXTI line 23
EXTI_24	EXTI line 24
EXTI_25	EXTI line 25
EXTI_26	EXTI line 26
EXTI_27	EXTI line 27
EXTI_28	EXTI line 28
EXTI_29	EXTI line 29
EXTI_30	EXTI line 30
EXTI_31	EXTI line 31

### Enum exti\_mode\_enum

Table 3-270. exti\_mode\_enum

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

### Enum exti\_trig\_type\_enum

Table 3-271. 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 3-272. 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 3-273. Function exti\_init**

Function name	exti_init
Function prototype	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
Function descriptions	initialize the EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, for GD32L233xx devices which refer to <a href="#">Table 3-268. exti_line_enum for GD32L233xx devices</a> , for for GD32L235xx devices which refer to <a href="#">Table 3-269. exti_line_enum for GD32L235xx devices</a>
Input parameter{in}	
mode	EXTI mode, refer to <a href="#">Table 3-270. exti_mode_enum</a>
Input parameter{in}	
trig_type	trigger type, refer to <a href="#">Table 3-271. exti_trig_type_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### exti\_interrupt\_enable

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

**Table 3-274. 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, for GD32L233xx devices which refer to <a href="#">Table 3-268. exti_line_enum for GD32L233xx devices</a> , for for GD32L235xx devices which refer to <a href="#">Table 3-269. exti_line_enum for GD32L235xx devices</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 3-275. 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, for GD32L233xx devices which refer to <a href="#">Table 3-268. exti_line_enum for GD32L233xx devices</a> , for for GD32L235xx devices which refer to <a href="#">Table 3-269. exti_line_enum for GD32L235xx devices</a>
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

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

```
exti_interrupt_disable(EXTI_0);
```

### exti\_event\_enable

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

**Table 3-276. Function exti\_event\_enable**

Function name	exti_event_enable
Function prototype	void exti_event_enable(exti_line_enum linex);
Function descriptions	enable the events from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, for GD32L233xx devices which refer to <a href="#">Table 3-268. exti_line_enum for GD32L233xx devices</a> , for for GD32L235xx devices which refer to <a href="#">Table 3-269. exti_line_enum for GD32L235xx devices</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
exti_event_enable(EXTI_0);
```

### exti\_event\_disable

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

**Table 3-277. Function exti\_event\_disable**

Function name	exti_event_disable
Function prototype	void exti_event_disable(exti_line_enum linex);
Function descriptions	disable the events from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, for GD32L233xx devices which refer to <a href="#">Table 3-268. exti_line_enum for GD32L233xx devices</a> , for for GD32L235xx devices which refer to <a href="#">Table 3-269. exti_line_enum for GD32L235xx devices</a>

Output parameter{out}	
-	-
Return value	
-	-

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 3-278. Function exti\_software\_interrupt\_enable**

Function name	exti_software_interrupt_enable
Function prototype	void exti_software_interrupt_enable(exti_line_enum linex);
Function descriptions	enable software interrupt event from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, for GD32L233xx devices which refer to <a href="#">Table 3-268. exti_line_enum for GD32L233xx devices</a> , for for GD32L235xx devices which refer to <a href="#">Table 3-269. exti_line_enum for GD32L235xx devices</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### exti\_software\_interrupt\_disable

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

**Table 3-279. Function exti\_software\_interrupt\_disable**

Function name	exti_software_interrupt_disable
Function prototype	void exti_software_interrupt_disable(exti_line_enum linex);
Function descriptions	disable software interrupt event from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	



<b>linex</b>	EXTI line x, for GD32L233xx devices which refer to <a href="#">Table 3-268. exti_line_enum for GD32L233xx devices</a> , for for GD32L235xx devices which refer to <a href="#">Table 3-269. exti_line_enum for GD32L235xx devices</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 3-280. 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, for GD32L233xx devices which refer to <a href="#">Table 3-268. exti_line_enum for GD32L233xx devices</a> , for for GD32L235xx devices which refer to <a href="#">Table 3-269. exti_line_enum for GD32L235xx devices</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 3-281. 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>	-

The called functions	-
Input parameter{in}	
linex	EXTI line x, for GD32L233xx devices which refer to <a href="#">Table 3-268. exti_line_enum for GD32L233xx devices</a> , for for GD32L235xx devices which refer to <a href="#">Table 3-269. exti_line_enum for GD32L235xx devices</a>
Output parameter{out}	
-	-
Return value	
-	-

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 3-282. Function exti\_interrupt\_flag\_get**

Function name	exti_interrupt_flag_get
Function prototype	FlagStatus exti_interrupt_flag_get(exti_line_enum linex);
Function descriptions	get EXTI line x interrupt pending flag
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, for GD32L233xx devices which refer to <a href="#">Table 3-268. exti_line_enum for GD32L233xx devices</a> , for for GD32L235xx devices which refer to <a href="#">Table 3-269. exti_line_enum for GD32L235xx devices</a>
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get EXTI line 0 interrupt flag status */
```

```
FlagStatus state = exti_interrupt_flag_get(EXTI_0);
```

### exti\_interrupt\_flag\_clear

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

**Table 3-283. Function exti\_interrupt\_flag\_clear**

Function name	exti_interrupt_flag_clear
Function prototype	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, for GD32L233xx devices which refer to <a href="#">Table 3-268. exti_line_enum for GD32L233xx devices</a> , for GD32L235xx devices which refer to <a href="#">Table 3-269. exti_line_enum for GD32L235xx devices</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear EXTI line 0 interrupt flag status */
```

```
exti_interrupt_flag_clear(EXTI_0);
```

## 3.12. FMC

There is flash controller and option byte for GD32L23x series. The FMC registers are listed in chapter [3.12.1](#) the FMC firmware functions are introduced in chapter [3.12.2](#).

### 3.12.1. Descriptions of Peripheral registers

FMC registers are listed in the table shown as below:

**Table 3-284. FMC Registers**

Registers	Descriptions
FMC_WS	FMC wait state register
FMC_KEY	FMC unlock key register
FMC_OBKEY	FMC option bytes unlock key register
FMC_STAT	FMC status register
FMC_CTL	FMC control register
FMC_ADDR	FMC address register
FMC_ECCCS	FMC ECC control and status register (only available in GD32L235xx)
FMC_OBSTAT	FMC option bytes status register
FMC_WP	FMC erase/write protection register
FMC_SLPKEY	FMC flash sleep or power-down mode unlock key register
FMC_PID	FMC product ID register

### 3.12.2. Descriptions of Peripheral functions

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

Table 3-285. FMC firmware function

Function name	Function description
fmc_unlock	unlock the main FMC operation
fmc_lock	lock the main FMC operation
fmc_wsnt_set	set the wait state
fmc_prefetch_enable	enable pre-fetch
fmc_prefetch_disable	disable pre-fetch
fmc_low_power_enable	enable low power (only available in GD32L233xx)
fmc_low_power_disable	disable low power (only available in GD32L233xx)
fmc_page_erase	erase FMC page
fmc_mass_erase	erase FMC whole chip
fmc_word_program	FMC program a word at the corresponding address (only available in GD32L233xx)
fmc_fast_program	FMC fast program a row words (32 double words) at the corresponding address (only available in GD32L233xx)
fmc_doubleword_program	FMC program a doubleword at the corresponding address (only available in GD32L235xx)
ob_unlock	unlock the option byte operation
ob_lock	lock the option byte operation
ob_erase	erase the option byte
ob_write_protection_enable	enable write protection
ob_security_protection_config	configure the option bytes security protection
ob_user_write	program option bytes USER
ob_data_program	program option bytes DATA
ob_user_get	get the value of option bytes USER
ob_data_get	get the value of option bytes DATA
ob_write_protection_get	get the value of option bytes write protection
ob_security_protection_flag_get	get option bytes security protection state
fmc_ecc_address_get	get the address where ECC error occur on
fmc_slp_unlock	unlock operation of RUN_SLP bit in FMC_WS register
fmc_sleep_slp_enable	flash enter sleep mode when MCU enter deep-sleep mode or RUN_SLP bit is set
fmc_sleep_slp_disable	flash enter power-down mode(GD32L233) / idle mode(GD32L235) when MCU enter deep-sleep mode or RUN_SLP bit is set
fmc_run_slp_enable	flash enter idle mode when MCU run mode (GD32L235, together with the SLEEP_SLP bit) / flash enter idle mode when MCU run mode or enter low-power run mode (GD32L233, together with the SLEEP_SLP bit)
fmc_run_slp_disable	flash enter sleep mode when MCU run mode (GD32L235, together with the SLEEP_SLP bit) / flash enter sleep or power-down mode when MCU run mode

Function name	Function description
	or enter low-power run mode (GD32L233, together with the SLEEP_SLP bit)
fmc_sleep_mode_enter	flash enter sleep mode when MCU run mode. Please note that this function needs to run in SRAM.
fmc_pd_mode_enter	flash enter power down mode when MCU run mode/low-power run mode (only available in GD32L233xx). Please note that this function needs to run in SRAM.
fmc_slp_mode_exit	flash exit sleep or power-down mode when MCU run mode/low-power run mode(GD32L233) flash exit sleep mode when MCU run mode mode(GD32L235). Please note that this function needs to run in SRAM.
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\_enum

Table 3-286. fmc\_state\_enum

enum name	enum description
FMC_READY	the operation has been completed
FMC_BUSY	the operation is in progress
FMC_PGERR	program error
FMC_PGAERR	program alignment error
FMC_WPERR	erase/program protection error
FMC_TOERR	timeout error
FMC_OB_HSPC	high security protection
FMC_SLP	sleep or power-down status

### fmc\_flag\_enum

Table 3-287. fmc\_flag\_enum

enum name	enum description
FMC_FLAG_BUSY	flash busy flag
FMC_FLAG_PGER R	flash program error flag
FMC_FLAG_PGAE RR	flash program alignment error flag
FMC_FLAG_WPER	flash erase/program protection error flag

enum name	enum description
R	
FMC_FLAG_SLP	flash sleep or power-down flag (only available in GD32L233xx)
FMC_FLAG_POWERDOWN	flash power-down flag (only available in GD32L235xx)
FMC_FLAG_SLEEP	flash sleep flag (only available in GD32L235xx)
FMC_FLAG_ECCCOR	one bit error detected and correct flag (only available in GD32L235xx)
FMC_FLAG_ECCDET	two bit error detect flag (only available in GD32L235xx)
FMC_FLAG_SYSECC	an ECC error is detected in system memory flag (only available in GD32L235xx)
FMC_FLAG_MFEC	an ECC error is detected in main Flash flag (only available in GD32L235xx)
FMC_FLAG_OTPECC	an ECC error is detected in OTP flag (only available in GD32L235xx)
FMC_FLAG_OBEC	an ECC error is detected in reading option bytes flag (only available in GD32L235xx)
FMC_FLAG_OBECDET	option bytes two bit error detect in loading option bytes to register flag (only available in GD32L235xx)
FMC_FLAG_OBER	option bytes error flag

### fmc\_interrupt\_flag\_enum

**Table 3-288. fmc\_interrupt\_flag\_enum**

enum name	enum description
FMC_INT_FLAG_PGERR	flash program error interrupt flag
FMC_INT_FLAG_PGERR	flash program alignment error interrupt flag
FMC_INT_FLAG_WPERR	flash erase/program protection error interrupt flag
FMC_INT_FLAG_END	flash end of operation interrupt flag
FMC_INT_FLAG_ECCCOR	one bit error detected and correct interrupt flag (only available in GD32L235xx)
FMC_INT_FLAG_ECCDET	two bits error detect interrupt flag (only available in GD32L235xx)
FMC_INT_FLAG_OBECDET	option bytes two bit error detect in loading option bytes to register interrupt flag (only available in GD32L235xx)

## fmc\_interrupt\_enum

**Table 3-289. fmc\_interrupt\_enum**

enum name	enum description
FMC_INT_ERR	FMC error interrupt
FMC_INT_END	FMC end of operation interrupt
FMC_INT_ECCCOR	FMC one bit error correct interrupt (only available in GD32L235xx)
FMC_INT_ECCDET	FMC two bits error interrupt (only available in GD32L235xx)

## fmc\_unlock

The description of fmc\_unlock is shown as below:

**Table 3-290. Function fmc\_unlock**

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

Example:

```
/* unlock the main FMC operation */
fmc_unlock();
```

## fmc\_lock

The description of fmc\_lock is shown as below:

**Table 3-291. Function fmc\_lock**

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

-	-
Return value	
-	-

Example:

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

```
fmc_lock();
```

### fmc\_wscnt\_set

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

**Table 3-292. 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 counter value
<i>FMC_WAIT_STATE_0</i>	0 wait state added
<i>FMC_WAIT_STATE_1</i>	1 wait state added
<i>FMC_WAIT_STATE_2</i>	2 wait state added
<i>FMC_WAIT_STATE_3</i>	3 wait state added (only available in GD32L233xx)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
fmc_wscnt_set(FMC_WAIT_STATE_1);
```

### fmc\_prefetch\_enable

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

**Table 3-293. 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>	-



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

Example:

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

```
fmc_prefetch_enable();
```

### fmc\_prefetch\_disable

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

**Table 3-294. Function fmc\_prefetch\_disable**

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

Example:

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

```
fmc_prefetch_disable();
```

### fmc\_low\_power\_enable

The description of fmc\_low\_power\_enable is shown as below:

**Table 3-295. Function fmc\_low\_power\_enable**

Function name	fmc_low_power_enable
Function prototype	void fmc_low_power_enable(void);
Function descriptions	enable low power
Precondition	-
The called functions	-
Input parameter{in}	

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

Example:

```
/* enable low power */
```

```
fmc_low_power_enable();
```

### fmc\_low\_power\_disable

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

**Table 3-296. Function fmc\_low\_power\_disable**

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

Example:

```
/* disable low power */
```

```
fmc_low_power_disable();
```

### fmc\_page\_erase

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

**Table 3-297. Function fmc\_page\_erase**

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

Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-286. fmc_state_enum</a> .

Example:

```
fmc_unlock();
```

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

```
fmc_state_enum state = fmc_page_erase(0x08004000);
```

### fmc\_mass\_erase

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

**Table 3-298. Function fmc\_mass\_erase**

Function name	fmc_mass_erase
Function prototype	fmc_state_enum fmc_mass_erase(void);
Function descriptions	erase whole chip
Precondition	fmc_unlock
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-286. fmc_state_enum</a> .

Example:

```
fmc_unlock();
```

```
/* erase whole chip */
```

```
fmc_state_enum state = fmc_mass_erase();
```

### fmc\_word\_program

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

**Table 3-299. Function fmc\_word\_program**

Function name	fmc_word_program
Function prototype	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
Function descriptions	program a word at the corresponding address
Precondition	fmc_unlock

The called functions	-
Input parameter{in}	
address	the address to be programmed
Input parameter{in}	
data	the data to be programmed
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-286. fmc_state_enum</a> .

Example:

```
fmc_unlock();
```

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

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

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

### fmc\_fast\_program

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

**Table 3-300. Function fmc\_fast\_program**

Function name	fmc_fast_program
Function prototype	fmc_state_enum fmc_fast_program(uint32_t address, uint64_t data[]);
Function descriptions	FMC fast program one row data (32 double-word) starting at the corresponding address
Precondition	fmc_unlock
The called functions	-
Input parameter{in}	
address	the address to be programmed
Input parameter{in}	
data	the data to be programmed
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-286. fmc_state_enum</a> .

Example:

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

```
static uint64_t data_buffer[32] = {
```

```

0x0000000000000000U,    0x1111111111111111U,    0x2222222222222222U,
0x3333333333333333U,

    0x4444444444444444U,    0x5555555555555555U,    0x6666666666666666U,
0x7777777777777777U,

    0x8888888888888888U,    0x9999999999999999U,    0xAAAAAAAAAAAAAAAAAU,
0xBBBBBBBBBBBBBBBBBBU,

    0xCCCCCCCCCCCCCCCCCU,                                0xDDDDDDDDDDDDDDDDDDDU,
0xEEEEEEEEEEEEEEEEU, 0xFFFFFFFFFFFUFU,

    0x0011001100110011U,    0x2233223322332233U,    0x4455445544554455U,
0x6677667766776677U,

    0x8899889988998899U, 0xAABBAABBAABBAABBU, 0xCCDDCCDDCCDDCCDDU,
0xEEFFEEFFEEFFEEFFU,

    0x2200220022002200U,    0x3311331133113311U,    0x6644664466446644U,
0x7755775577557755U,

    0xAA88AA88AA88AA88U, 0xBB99BB99BB99BB99U, 0xEECCEECCEECCEECU,
0xFFDDFFDDFFDDFFDDU

};

fmc_unlock();

fmc_page_erase(0x08004000);

/* program flash */

fmc_state_enum fmc_state = fmc_fast_program(0x08004000, data_buffer);

```

### fmc\_doubleword\_program

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

**Table 3-301. 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 corresponding address
Precondition	fmc_unlock
The called functions	-
Input parameter{in}	
address	the address to be programmed
Input parameter{in}	
data	the data to be programmed
Output parameter{out}	

-	-
<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 3-286. fmc_state_enum</a> .

Example:

```
fmc_unlock();
```

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

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

```
fmc_state_enum          fmc_state          =          fmc_
doubleword_program(0x08004000,0xaabbccddaabbccdd);
```

## ob\_unlock

The description of ob\_unlock is shown as below:

**Table 3-302. Function ob\_unlock**

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

Example:

```
fmc_unlock();
```

```
/* unlock the option byte operation */
```

```
ob_unlock();
```

## ob\_lock

The description of ob\_lock is shown as below:

**Table 3-303. Function ob\_lock**

<b>Function name</b>	ob_lock
<b>Function prototype</b>	void ob_lock(void);
<b>Function descriptions</b>	lock the option byte operation

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

Example:

```
fmc_lock();
```

```
/* lock the option byte operation */
```

```
ob_lock();
```

### ob\_erase

The description of ob\_erase is shown as below:

**Table 3-304. Function ob\_erase**

<b>Function name</b>	ob_erase
<b>Function prototype</b>	void ob_erase(void);
<b>Function descriptions</b>	erase the option byte
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<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 3-286. fmc_state_enum</a> .

Example:

```
fmc_unlock();
```

```
ob_unlock();
```

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

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

### ob\_write\_protection\_enable

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

Table 3-305. Function ob\_write\_protection\_enable

Function name	ob_write_protection_enable
Function prototype	fmc_state_enum ob_write_protection_enable(uint32_t ob_wp);
Function descriptions	enable erase/write protection
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
ob_wp	specify sector to be write protected
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-286. fmc_state_enum</a> .

Example:

```
fmc_unlock();
ob_unlock();
/* enable write protection */
fmc_state_enum fmc_state = ob_write_protection_enable(OB_WP_1);
```

### ob\_security\_protection\_config

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

Table 3-306. Function ob\_security\_protection\_config

Function name	ob_security_protection_config
Function prototype	fmc_state_enum ob_security_protection_config(uint8_t ob_spc);
Function descriptions	configure security protection
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
ob_spc	specify security protection
FMC_NSPC	no security protection
FMC_LSPC	low security protection
FMC_HSPC	high security protection
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-286. fmc_state_enum</a> .

Example:



```
fmc_state_enum fmc_state;

fmc_unlock();

ob_unlock();

/* enable security protection */

fmc_state = ob_security_protection_config(FMC_USPC);
```

## ob\_user\_write

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

**Table 3-307. Function ob\_user\_write**

Function name	ob_user_write
Function prototype	fmc_state_enum ob_user_write(uint8_t ob_fwdgt, uint8_t ob_deepsleep, uint8_t ob_stdby, uint8_t ob_bor_th); (only available in GD32L233xx)/ fmc_state_enum ob_user_write(uint8_t ob_fwdgt, uint8_t ob_deepsleep, uint8_t ob_stdby, uint8_t ob_bor_th, uint8_t ob_sram_parity_check); (only available in GD32L235xx)
Function descriptions	program option bytes USER
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
<b>ob_fwdgt</b>	option bytes free watchdog value
<i>OB_FWDGT_SW</i>	software free watchdog
<i>OB_FWDGT_HW</i>	hardware free watchdog
Input parameter{in}	
<b>ob_deepsleep</b>	option bytes deepsleep reset value
<i>OB_DEEPSLEEP_NRS</i> <i>T</i>	no reset when entering deepsleep mode
<i>OB_DEEPSLEEP_RST</i>	generate a reset instead of entering deepsleep mode
Input parameter{in}	
<b>ob_stdby</b>	option bytes standby reset value
<i>OB_STDBY_NRST</i>	no reset when entering standby mode
<i>OB_STDBY_RST</i>	generate a reset instead of entering standby mode
Input parameter{in}	
<b>ob_bor_th</b>	option bytes BOR threshold value
<i>OB_BOR_TH_VALUE0</i>	BOR threshold value 0
<i>OB_BOR_TH_VALUE1</i>	BOR threshold value 1
<i>OB_BOR_TH_VALUE2</i>	BOR threshold value 2
<i>OB_BOR_TH_VALUE3</i>	BOR threshold value 3
<i>OB_BOR_TH_VALUE4</i>	BOR threshold value 4
Input parameter{in}	
<b>ob_sram_parity_chec</b>	option bytes sram parity check value (only available in GD32L235xx)

<b>k</b>	
<i>OB_SRAM_PARITY_CHECK_ENABLE</i>	enable SRAM parity check
<i>OB_SRAM_PARITY_CHECK_DISABLE</i>	disable SRAM parity check
<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 3-286. fmc_state_enum</a> .

Example:

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* program the FMC user option byte */
```

```
fmc_state_enum fmc_state = ob_user_write(OB_FWDGT_HW, OB_DEEPSLEEP_Nrst,
OB_STDBY_Nrst, OB_BOR_TH_VALUE0);
```

### ob\_data\_program

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

**Table 3-308. Function ob\_data\_program**

<b>Function name</b>	ob_data_program
<b>Function prototype</b>	fmc_state_enum ob_data_program(uint16_t data);
<b>Function descriptions</b>	program option bytes DATA
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>data</b>	the data to be programmed
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-286. fmc_state_enum</a> .

Example:

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* program option bytes data */
```

```
fmc_state_enum fmc_state = ob_data_program(0xddd22);
```

## ob\_user\_get

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

**Table 3-309. Function ob\_user\_get**

<b>Function name</b>	ob_user_get
<b>Function prototype</b>	uint8_t ob_user_get(void);
<b>Function descriptions</b>	get the value of option bytes USER
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint8_t	the FMC user option byte values(0x00 – 0xFF)

Example:

```
/* get the FMC user option byte */
```

```
uint8_t user = ob_user_get();
```

## ob\_data\_get

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

**Table 3-310. Function ob\_data\_get**

<b>Function name</b>	ob_data_get
<b>Function prototype</b>	uint16_t ob_data_get(void);
<b>Function descriptions</b>	get the value of option bytes DATA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	the FMC data option byte values(0x0 – 0xFFFF)

Example:

```
/* get the FMC data option byte */
```

```
uint16_t data = ob_data_get();
```

## ob\_write\_protection\_get

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

**Table 3-311. Function ob\_write\_protection\_get**

<b>Function name</b>	ob_write_protection_get
<b>Function prototype</b>	uint32_t ob_write_protection_get(void);
<b>Function descriptions</b>	get the value of option bytes write protection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	the FMC write protection option byte value(0x0 – 0xFFFF FFFF)

Example:

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

```
uint32_t wp = ob_write_protection_get();
```

## ob\_security\_protection\_flag\_get

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

**Table 3-312. Function ob\_security\_protection\_flag\_get**

<b>Function name</b>	ob_security_protection_flag_get
<b>Function prototype</b>	FlagStatus ob_security_protection_flag_get(void);
<b>Function descriptions</b>	get the FMC option bytes security protection state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
FlagStatus	SET or RESET

Example:

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

```
FlagStatus spc_flag = ob_security_protection_flag_get();
```

## fmc\_ecc\_address\_get

The description of fmc\_ecc\_address\_get is shown as below:

**Table 3-313. Function fmc\_ecc\_address\_get**

<b>Function name</b>	fmc_ecc_address_get
<b>Function prototype</b>	uint16_t fmc_ecc_address_get(void);
<b>Function descriptions</b>	get the address where ECC error occur on
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	0 ~ 0x3FFF

Example:

```
/* get the address where ECC error occur on */
```

```
uint16_t ecc_addr = fmc_ecc_address_get();
```

## fmc\_slp\_unlock

The description of fmc\_slp\_unlock is shown as below:

**Table 3-314. Function fmc\_slp\_unlock**

<b>Function name</b>	fmc_slp_unlock
<b>Function prototype</b>	void fmc_slp_unlock(void);
<b>Function descriptions</b>	unlock operation of RUN_SLP bit in FMC_WS 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 RUN_SLP bit in FMC_WS register */
```

```
fmc_slp_unlock();
```

## fmc\_sleep\_slp\_enable

The description of fmc\_sleep\_slp\_enable is shown as below:

**Table 3-315. Function fmc\_sleep\_slp\_enable**

<b>Function name</b>	fmc_sleep_slp_enable
<b>Function prototype</b>	void fmc_sleep_slp_enable(void);
<b>Function descriptions</b>	flash enter sleep/power-down mode during MCU run/low-power run mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* flash enter sleep/power-down mode during MCU run/low-power run mode */
```

```
fmc_sleep_slp_enable();
```

## fmc\_sleep\_slp\_disable

The description of fmc\_sleep\_slp\_disable is shown as below:

**Table 3-316. Function fmc\_sleep\_slp\_disable**

<b>Function name</b>	fmc_sleep_slp_disable
<b>Function prototype</b>	void fmc_sleep_slp_disable(void);
<b>Function descriptions</b>	For GD32L233, flash enter power-down mode when MCU enter deep-sleep mode or RUN_SLP bit is set. For GD32L235, flash enter idle mode when MCU enter deep-sleep mode or RUN_SLP bit is set
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* for GD32L233, flash enter power-down mode when MCU enter deep-sleep mode or RUN_SLP bit is set */
```

```
fmc_sleep_slp_disable();
```

### fmc\_run\_slp\_enable

The description of fmc\_run\_slp\_enable is shown as below:

**Table 3-317. Function fmc\_run\_slp\_enable**

<b>Function name</b>	fmc_run_slp_enable
<b>Function prototype</b>	void fmc_run_slp_enable(void);
<b>Function descriptions</b>	For GD32L233, flash enter idle mode when MCU run mode or enter low-power run mode (together with the SLEEP_SLP bit). For GD32L235, flash enter idle mode when MCU run mode (together with the SLEEP_SLP bit).
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
fmc_slp_unlock();
```

```
/* for GD32L235, flash enter idle mode when MCU run mode (together with the SLEEP_SLP bit) */
```

```
fmc_run_slp_enable();
```

### fmc\_run\_slp\_disable

The description of fmc\_run\_slp\_disable is shown as below:

**Table 3-318. Function fmc\_run\_slp\_disable**

<b>Function name</b>	fmc_run_slp_disable
<b>Function prototype</b>	void fmc_run_slp_disable(void);
<b>Function descriptions</b>	For GD32L233, flash enter sleep or power-down mode when MCU run mode or enter low-power run mode (together with the SLEEP_SLP bit). For GD32L235, flash enter sleep mode when MCU run mode (together with the SLEEP_SLP bit).
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
fmc_slp_unlock();
```

```
/* for GD32L235, flash enter sleep mode when MCU run mode (together with the SLEEP_SLP bit) */
```

```
fmc_run_slp_disable();
```

### fmc\_sleep\_mode\_enter

The description of fmc\_sleep\_mode\_enter is shown as below:

**Table 3-319. Function fmc\_sleep\_mode\_enter**

<b>Function name</b>	fmc_sleep_mode_enter
<b>Function prototype</b>	void fmc_sleep_mode_enter(void);
<b>Function descriptions</b>	flash enter sleep mode when MCU run mode. Please note that this function needs to run in SRAM.
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* flash enter sleep mode when MCU run mode */
```

```
fmc_sleep_mode_enter ();
```

### fmc\_pd\_mode\_enter

The description of fmc\_pd\_mode\_enter is shown as below:

**Table 3-320. Function fmc\_pd\_mode\_enter**

<b>Function name</b>	fmc_pd_mode_enter
<b>Function prototype</b>	void fmc_pd_mode_enter(void);
<b>Function descriptions</b>	flash enter power down mode when MCU run mode/low-power run mode. Please note that this function needs to run in SRAM.
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	



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

Example:

```
/* for GD32L235, flash enter sleep mode when MCU run mode (together with the SLEEP_SLP
bit) */
```

```
fmc_pd_mode_enter ();
```

### fmc\_slp\_mode\_exit

The description of fmc\_slp\_mode\_exit is shown as below:

**Table 3-321. Function fmc\_slp\_mode\_exit**

<b>Function name</b>	fmc_slp_mode_exit
<b>Function prototype</b>	void fmc_slp_mode_exit(void);
<b>Function descriptions</b>	For GD32L233, flash exit sleep or power-down mode when MCU run mode/low-power run mode. For GD32L235, flash exit sleep mode when MCU run mode mode. Please note that this function needs to run in SRAM.
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* for GD32L235, flash exit sleep mode when MCU run mode mode */
```

```
fmc_slp_mode_exit ();
```

### fmc\_flag\_get

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

**Table 3-322. Function fmc\_flag\_get**

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

Input parameter{in}	
flag	FMC flag, the enum members can refer to members of the enum <a href="#">Table 3-287. fmc_flag_enum</a> .
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

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

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

### fmc\_flag\_clear

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

**Table 3-323. Function fmc\_flag\_clear**

Function name	fmc_flag_clear
Function prototype	void fmc_flag_clear(fmc_flag_enum flag);
Function descriptions	clear the FMC flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	FMC flag
FMC_FLAG_PGERR	flash program error flag
FMC_FLAG_PGAERR	flash program alignment error flag
FMC_FLAG_WPERR	flash erase/program protection error flag
FMC_FLAG_ECCCOR	one bit error detected and correct flag (only available in GD32L235xx)
FMC_FLAG_ECCDET	two bit error detect flag (only available in GD32L235xx)
FMC_FLAG_SYSECC	an ECC error is detected in system memory flag (only available in GD32L235xx)
FMC_FLAG_MFECC	an ECC error is detected in main Flash flag (only available in GD32L235xx)
FMC_FLAG_OTPECC	an ECC error is detected in OTP flag (only available in GD32L235xx)
FMC_FLAG_OBECC	an ECC error is detected in reading option bytes flag (only available in GD32L235xx)
FMC_FLAG_OBECCDET	option bytes two bit error detect in loading option bytes to register flag (only available in GD32L235xx)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
fmc_flag_clear(FMC_FLAG_END);
```

## fmc\_interrupt\_enable

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

**Table 3-324. Function fmc\_interrupt\_enable**

<b>Function name</b>	fmc_interrupt_enable
<b>Function prototype</b>	void fmc_interrupt_enable(fmc_interrupt_enum interrupt);
<b>Function descriptions</b>	enable FMC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	FMC interrupt, the enum members can refer to members of the enum <a href="#">Table 3-289. fmc_interrupt_enum</a> .
<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 3-325. Function fmc\_interrupt\_disable**

<b>Function name</b>	fmc_interrupt_disable
<b>Function prototype</b>	void fmc_interrupt_disable(fmc_interrupt_enum interrupt);
<b>Function descriptions</b>	disable FMC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	FMC interrupt, the enum members can refer to members of the enum <a href="#">Table 3-289. fmc_interrupt_enum</a> .
<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 3-326. 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 int_flag);
<b>Function descriptions</b>	get FMC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	FMC interrupt flag, the enum members can refer to members of the enum <a href="#">Table 3-288. fmc_interrupt_flag_enum</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* check FMC program operation error flag is set or not */
```

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

### fmc\_interrupt\_flag\_clear

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

**Table 3-327. Function fmc\_interrupt\_flag\_clear**

<b>Function name</b>	fmc_interrupt_flag_clear
<b>Function prototype</b>	void fmc_interrupt_flag_clear(fmc_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	clear the FMC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	FMC interrupt flag, the enum members can refer to members of the enum <a href="#">Table 3-288. fmc_interrupt_flag_enum</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear FMC program operation error flag */
```

```
fmc_interrupt_flag_get(FMC_INT_FLAG_PGERR);
```

### 3.13. FWDGT

The free watchdog timer (FWDGT) is a hardware timing circuitry that can be used to detect system failures due to software malfunctions. It's suitable for the situation that requires an independent environment and lower timing accuracy. The FWDGT registers are listed in chapter [3.13.1](#) the FWDGT firmware functions are introduced in chapter [3.13.2](#).

#### 3.13.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

**Table 3-328. FWDGT Registers**

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

#### 3.13.2. Descriptions of Peripheral functions

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

**Table 3-329. FWDGT firmware function**

Function name	Function description
fwdgt_write_enable	enable write access to FWDGT_PSC and FWDGT_RLD and FWDGT_WND
fwdgt_write_disable	disable write access to FWDGT_PSC and FWDGT_RLD and FWDGT_WND
fwdgt_enable	start the FWDGT counter
fwdgt_prescaler_value_config	configure the FWDGT counter prescaler value
fwdgt_reload_value_config	configure the FWDGT counter reload value
fwdgt_window_value_config	configure the FWDGT counter window value
fwdgt_counter_reload	reload the counter of FWDGT
fwdgt_config	configure counter reload value, and prescaler divider value
fwdgt_flag_get	get flag state of FWDGT

##### **fwdgt\_write\_enable**

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

Table 3-330. Function fwdgt\_write\_enable

Function name	fwdgt_write_enable
Function prototype	void fwdgt_write_enable(void);
Function descriptions	enable write access to FWDGT_PSC and FWDGT_RLD 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 3-331. 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 3-332. Function fwdgt\_enable

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

Example:

```
/* start the free watchdog timer counter */
```

```
fwdgt_enable();
```

### fwdgt\_prescaler\_value\_config

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

Table 3-333. 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_DIV4</i>	FWDGT prescaler set to 4
<i>FWDGT_PSC_DIV8</i>	FWDGT prescaler set to 8
<i>FWDGT_PSC_DIV16</i>	FWDGT prescaler set to 16
<i>FWDGT_PSC_DIV32</i>	FWDGT prescaler set to 32
<i>FWDGT_PSC_DIV64</i>	FWDGT prescaler set to 64
<i>FWDGT_PSC_DIV128</i>	FWDGT prescaler set to 128
<i>FWDGT_PSC_DIV256</i>	FWDGT prescaler set to 256
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR / SUCCESS

Example:

```
/* set FWDGT prescaler to 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 3-334. 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 0xFFFF */
```

ErrStatus flag;

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

### fwdgt\_window\_value\_config

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

**Table 3-335. 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 0xFFFF */
```

```
ErrStatus flag;
```

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

### fwdgt\_counter\_reload

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

**Table 3-336. Function fwdgt\_counter\_reload**

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

Example:

```
/* reload FWDGT counter */
```

```
fwdgt_counter_reload();
```

### fwdgt\_config

The description of fwdgt\_config is shown as below:

**Table 3-337. Function fwdgt\_config**

<b>Function name</b>	fwdgt_config
<b>Function prototype</b>	ErrStatus fwdgt_config(uint16_t reload_value, uint8_t prescaler_div);
<b>Function descriptions</b>	configure counter reload value, and prescaler divider value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>reload_value</b>	specify reload value(0x0000 - 0x0FFF)
<b>Input parameter{in}</b>	
<b>prescaler_div</b>	FWDGT prescaler value
<i>FWDGT_PSC_DIV4</i>	FWDGT prescaler set to 4
<i>FWDGT_PSC_DIV8</i>	FWDGT prescaler set to 8
<i>FWDGT_PSC_DIV16</i>	FWDGT prescaler set to 16
<i>FWDGT_PSC_DIV32</i>	FWDGT prescaler set to 32

<i>FWDGT_PSC_DIV64</i>	FWDGT prescaler set to 64
<i>FWDGT_PSC_DIV128</i>	FWDGT prescaler set to 128
<i>FWDGT_PSC_DIV256</i>	FWDGT prescaler set to 256
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

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

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

### fwdgt\_flag\_get

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

**Table 3-338. Function fwdgt\_flag\_get**

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

Example:

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

```
FlagStatus status;
```

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

## 3.14. GPIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter [3.14.1](#), the GPIO firmware functions are introduced in chapter [3.14.2](#).

### 3.14.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

**Table 3-339. 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
GPIOx_AFSEL1	GPIO alternate function selected register 1
GPIOx_BC	GPIO bit clear register
GPIOx_TG	GPIO port bit toggle register

### 3.14.2. Descriptions of Peripheral functions

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

**Table 3-340. 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 3-341. 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 3-342. Function gpio\_mode\_set

Function name	gpio_mode_set
Function prototype	void gpio_mode_set(uint32_t gpio_periph, uint32_t mode, uint32_t pull_up_down, uint32_t pin);
Function descriptions	set GPIO mode
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C,D,F)
Input parameter{in}	
mode	gpio pin mode
GPIO_MODE_INPUT	input mode
GPIO_MODE_OUTPUT	output mode
GPIO_MODE_AF	alternate function mode
GPIO_MODE_ANALOG	analog mode
Input parameter{in}	
pull_up_down	gpio pin with pull-up or pull-down resistor
GPIO_PUPD_NONE	floating mode, no pull-up and pull-down resistors

<i>GPIO_PUPD_PULLUP</i>	with pull-up resistor
<i>GPIO_PUPD_PULLDOWN</i>	with pull-down resistor
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<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 3-343. 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_2MHZ</i>	output max speed 2MHz
<i>GPIO_OSPEED_10MHZ</i>	output max speed 10MHz
<i>GPIO_OSPEED_50MHZ</i>	output max speed 50MHz
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin

<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config PA0 as 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 3-344. Function gpio\_bit\_set**

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

Example:

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

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

### gpio\_bit\_reset

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

**Table 3-345. Function gpio\_bit\_reset**

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

Example:

```
/* reset PA0 */
```

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

## gpio\_bit\_write

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

**Table 3-346. Function gpio\_bit\_write**

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

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 3-347. Function gpio\_port\_write**

Function name	gpio_port_write
Function prototype	void gpio_port_write(uint32_t gpio_periph,uint16_t data);
Function descriptions	write data to the specified GPIO port
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C,D,F)
Input parameter{in}	
data	specify the value to be written to the port output data register
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

## gpio\_input\_bit\_get

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

**Table 3-348. Function gpio\_input\_bit\_get**

Function name	gpio_input_bit_get
Function prototype	FlagStatus gpio_input_bit_get(uint32_t gpio_periph,uint32_t pin);
Function descriptions	get GPIO pin input status
Precondition	-
The called functions	-



Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,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 3-349. 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>	-
Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,F)
Output parameter{out}	
-	-
Return value	
<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 3-350. 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 3-351. 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 3-352. Function gpio\_af\_set**

Function name	gpio_af_set
Function prototype	void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);
Function descriptions	set GPIO alternate function
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>alt_func_num</b>	GPIO pin af function, please refer to specific device datasheet
<i>GPIO_AF_0</i>	<i>SPI0, SPI1, TIMER0, RTC_OUT, CK_OUT, SWDIO, SWCLK, LPTIMER1, TIMER14</i>
<i>GPIO_AF_1</i>	<i>TIMER1, TIMER40, TIMER2, TIMER0, LPTIMER0, TIMER11, TIMER8, LPTIMER1, TIMER14</i>
<i>GPIO_AF_2</i>	<i>LPTIMER0, TIMER8, LPUART0, TIMER14, TIMER11, TIMER0</i>
<i>GPIO_AF_3</i>	<i>SLCD</i>
<i>GPIO_AF_4</i>	<i>I2C0, I2C1, I2C2, UART4, CAN</i>
<i>GPIO_AF_5</i>	<i>SPI0, SPI1, I2S1, I2C2</i>
<i>GPIO_AF_6</i>	<i>SPI1, CMP0, CMP1, I2S1, LPTIMER1, TIMER14</i>
<i>GPIO_AF_7</i>	<i>USART1, TIMER11, USART0, LPUART1, CAN, UART3, UART4, LPUART0</i>
<i>GPIO_AF_8</i>	<i>CTC, LPUART0, UART4, I2C1, CMP1, UART3, TIMER11, CAN, USART1</i>
<i>GPIO_AF_9</i>	<i>EVENTOUT</i>
<i>GPIO_AF_10</i>	<i>TIMER1, TIMER8, TIMER11, TIMER14, TIMER40, I2C1, LPUART1</i>
<i>GPIO_AF_11</i>	<i>TIMER0, TIMER40, I2C1, I2C2, CAN, TIMER11</i>
Input parameter{in}	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/*set PA0 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 3-353. Function gpio\_pin\_lock**

Function name	gpio_pin_lock
Function prototype	void gpio_pin_lock(uint32_t gpio_periph, uint32_t pin);
Function descriptions	lock GPIO pin bit
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,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:

```
/* lock PA0 */
```

```
gpio_pin_lock(GPIOA, GPIO_PIN_0);
```

## gpio\_bit\_toggle

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

**Table 3-354. Function gpio\_bit\_toggle**

Function name	gpio_bit_toggle
Function prototype	void gpio_bit_toggle(uint32_t gpio_periph, uint32_t pin);
Function descriptions	toggle GPIO pin status
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>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	GPIO_PIN_ALL
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* toggle PA0 */
gpio_bit_toggle(GPIOA, GPIO_PIN_0);
```

### gpio\_port\_toggle

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

**Table 3-355. Function gpio\_port\_toggle**

<b>Function name</b>	gpio_port_toggle
<b>Function prototype</b>	void gpio_port_toggle(uint32_t gpio_periph);
<b>Function descriptions</b>	toggle GPIO port 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)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* toggle GPIOA */
gpio_port_toggle(GPIOA);
```

## 3.15. I2C

The I2C (inter-integrated circuit) module provides an I2C interface which is an industry standard two-line serial interface for MCU to communicate with external I2C interface. The

I2C registers are listed in chapter [3.15.1](#), the I2C firmware functions are introduced in chapter [3.15.2](#).

### 3.15.1. Descriptions of Peripheral registers

I2C registers are listed in the table shown as below:

**Table 3-356. 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	Status clear register
I2C_PEC	PEC register
I2C_RDATA	Receive data register
I2C_TDATA	Transmit data register
I2C_CTL2	Control register 2

### 3.15.2. Descriptions of Peripheral functions

I2C firmware functions are listed in the table shown as below:

**Table 3-357. 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

Function name	Function description
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_receved_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
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_extented_clock_timeout_enable	enable extended clock timeout detection

Function name	Function description
i2c_extented_clock_timeout_disable	disable extended clock timeout detection
i2c_clock_timeout_enable	enable clock timeout detection
i2c_clock_timeout_disable	disable clock timeout detection
i2c_bus_timeout_b_config	configure bus timeout B
i2c_bus_timeout_a_config	configure bus timeout A
i2c_idle_clock_timeout_config	configure idle clock timeout detection
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 3-358. 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
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	overflow/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 3-359. Function i2c\_deinit

Function name	i2c_deinit
Function prototype	void i2c_deinit(uint32_t i2c_periph);
Function descriptions	reset I2C
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
i2c_periph	I2C peripheral



<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset I2C0 */
i2c_deinit(I2C0);
```

### i2c\_timing\_config

The description of i2c\_timing\_config is shown as below:

**Table 3-360. 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>	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
<b>psc</b>	0-0xf, timing prescaler
Input parameter{in}	
<b>scl_dely</b>	0-0xf, data setup time
Input parameter{in}	
<b>sda_dely</b>	0-0xf, data hold time
Output parameter{out}	
-	-
Return value	
-	-

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 3-361. Function `i2c_digital_noise_filter_config`

Function name	<code>i2c_digital_noise_filter_config</code>
Function prototype	<code>void i2c_digital_noise_filter_config(uint32_t i2c_periph, uint32_t filter_length);</code>
Function descriptions	configure digital noise filter
Precondition	-
The called functions	-
Input parameter{in}	
<code>i2c_periph</code>	I2C peripheral
<code>I2Cx</code>	(x=0,1,2)
Input parameter{in}	
<code>filter_length</code>	<code>filter_length</code>
<code>FILTER_DISABLE</code>	digital filter is disabled
<code>FILTER_LENGTH_1</code>	digital filter is enabled and filter spikes with a length of up to 1 $t_{I2CCLK}$
<code>FILTER_LENGTH_2</code>	digital filter is enabled and filter spikes with a length of up to 2 $t_{I2CCLK}$
<code>FILTER_LENGTH_3</code>	digital filter is enabled and filter spikes with a length of up to 3 $t_{I2CCLK}$
<code>FILTER_LENGTH_4</code>	digital filter is enabled and filter spikes with a length of up to 4 $t_{I2CCLK}$
<code>FILTER_LENGTH_5</code>	digital filter is enabled and filter spikes with a length of up to 5 $t_{I2CCLK}$
<code>FILTER_LENGTH_6</code>	digital filter is enabled and filter spikes with a length of up to 6 $t_{I2CCLK}$
<code>FILTER_LENGTH_7</code>	digital filter is enabled and filter spikes with a length of up to 7 $t_{I2CCLK}$
<code>FILTER_LENGTH_8</code>	digital filter is enabled and filter spikes with a length of up to 8 $t_{I2CCLK}$
<code>FILTER_LENGTH_9</code>	digital filter is enabled and filter spikes with a length of up to 9 $t_{I2CCLK}$
<code>FILTER_LENGTH_10</code>	digital filter is enabled and filter spikes with a length of up to 10 $t_{I2CCLK}$
<code>FILTER_LENGTH_11</code>	digital filter is enabled and filter spikes with a length of up to 11 $t_{I2CCLK}$
<code>FILTER_LENGTH_12</code>	digital filter is enabled and filter spikes with a length of up to 12 $t_{I2CCLK}$
<code>FILTER_LENGTH_13</code>	digital filter is enabled and filter spikes with a length of up to 13 $t_{I2CCLK}$
<code>FILTER_LENGTH_14</code>	digital filter is enabled and filter spikes with a length of up to 14 $t_{I2CCLK}$
<code>FILTER_LENGTH_15</code>	digital filter is enabled and filter spikes with a length of up to 15 $t_{I2CCLK}$
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 digital filter filters spikes with a length of up to 1 tI2CCLK */
```

```
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 3-362. Function `i2c_analog_noise_filter_enable`

Function name	<code>i2c_analog_noise_filter_enable</code>
---------------	---

<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=0,1,2)
<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 3-363. Function i2c\_analog\_noise\_filter\_disable**

<b>Function name</b>	i2c_analog_noise_filter_disable
<b>Function prototype</b>	void i2c_analog_noise_filter_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable analog noise filter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-364. 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=0,1,2)
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 3-365. Function i2c\_master\_addressing

Function name	i2c_master_addressing
Function prototype	void i2c_master_addressing(uint32_t i2c_periph, uint32_t address, uint32_t trans_direction);
Function descriptions	configure i2c slave addresss and transfer direction in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
address	0-0x3FF except reserved address, I2C slave address to be sent
Input parameter{in}	
trans_direction	I2C transfer direction in master mode
I2C_MASTER_TRANS MIT	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 3-366. 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=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* 10-bit address header executes read direction only in master receive mode */
```

```
i2c_address10_header_enable(I2C0);
```

### i2c\_address10\_header\_disable

The description of i2c\_address10\_header\_disable is shown as below:

**Table 3-367. 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>	-

Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

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 3-368. 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=0,1,2)
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 3-369. 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>	-

The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
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 3-370. Function i2c\_automatic\_end\_enable**

Function name	i2c_automatic_end_enable
Function prototype	void i2c_automatic_end_enable(uint32_t i2c_periph);
Function descriptions	enable I2C automatic end mode in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
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 3-371. 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

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable 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 3-372. Function i2c\_slave\_response\_to\_gcall\_enable**

<b>Function name</b>	i2c_slave_response_to_gcall_enable
<b>Function prototype</b>	void i2c_slave_response_to_gcall_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable the response to a general call
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the response to a general call */
```

```
i2c_slave_response_to_gcall_enable(I2C0);
```

### i2c\_slave\_response\_to\_gcall\_disable

The description of i2c\_slave\_response\_to\_gcall\_disable is shown as below:

**Table 3-373. Function i2c\_slave\_response\_to\_gcall\_disable**

<b>Function name</b>	i2c_slave_response_to_gcall_disable
<b>Function prototype</b>	void i2c_slave_response_to_gcall_disable(uint32_t i2c_periph);



<b>Function descriptions</b>	disable the response to a general call
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the response to a general call */
i2c_slave_response_to_gcall_disable(I2C0);
```

### i2c\_stretch\_scl\_low\_enable

The description of i2c\_stretch\_scl\_low\_enable is shown as below:

**Table 3-374. Function i2c\_stretch\_scl\_low\_enable**

<b>Function name</b>	i2c_stretch_scl_low_enable
<b>Function prototype</b>	void i2c_stretch_scl_low_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable to stretch SCL low when data is not ready in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable to stretch SCL low when data is not ready in slave mode */
i2c_stretch_scl_low_enable(I2C0);
```

### i2c\_stretch\_scl\_low\_disable

The description of i2c\_stretch\_scl\_low\_disable is shown as below:

**Table 3-375. Function i2c\_stretch\_scl\_low\_disable**

<b>Function name</b>	i2c_stretch_scl_low_disable
----------------------	-----------------------------

<b>Function prototype</b>	void i2c_stretch_scl_low_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable to stretch SCL low when data is not ready in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable to stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_disable(I2C0);
```

### i2c\_address\_config

The description of i2c\_address\_config is shown as below:

**Table 3-376. 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=0,1,2)
<b>Input parameter{in}</b>	
<b>address</b>	I2C address
<b>Input parameter{in}</b>	
<b>addr_format</b>	7bits or 10bits
<i>I2C_ADDFORMAT_7BITS</i>	7bits
<i>I2C_ADDFORMAT_10BITS</i>	10bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure i2c slave address */
```

```
i2c_address_config(I2C0, 0x82, I2C_ADDFORMAT_7BITS);
```

### i2c\_address\_bit\_compare\_config

The description of i2c\_address\_bit\_compare\_config is shown as below:

**Table 3-377. Function i2c\_address\_bit\_compare\_config**

Function name	i2c_address_bit_compare_config
Function prototype	void i2c_address_bit_compare_config(uint32_t i2c_periph, uint32_t compare_bits);
Function descriptions	define which bits of ADDRESS[7:1] need to compare with the incoming address byte
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
compare_bits	the bits need to compare
ADDRESS_BIT1_COMPARE	address bit1 needs compare
ADDRESS_BIT2_COMPARE	address bit2 needs compare
ADDRESS_BIT3_COMPARE	address bit3 needs compare
ADDRESS_BIT4_COMPARE	address bit4 needs compare
ADDRESS_BIT5_COMPARE	address bit5 needs compare
ADDRESS_BIT6_COMPARE	address bit6 needs compare
ADDRESS_BIT7_COMPARE	address bit7 needs compare
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-378. 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=0,1,2)
<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 3-379. 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=0,1,2)
<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
<i>ADDRESS2_NO_MAS K</i>	no mask, all the bits must be compared
<i>ADDRESS2_MASK_B/ T1</i>	ADDRESS2[1] is masked, only ADDRESS2[7:2] are compared

ADDRESS2_MASK_BIT1_2	ADDRESS2[2:1] is masked, only ADDRESS2[7:3] are compared
ADDRESS2_MASK_BIT1_3	ADDRESS2[3:1] is masked, only ADDRESS2[7:4] are compared
ADDRESS2_MASK_BIT1_4	ADDRESS2[4:1] is masked, only ADDRESS2[7:5] are compared
ADDRESS2_MASK_BIT1_5	ADDRESS2[5:1] is masked, only ADDRESS2[7:6] are compared
ADDRESS2_MASK_BIT1_6	ADDRESS2[6:1] is masked, only ADDRESS2[7] are compared
ADDRESS2_MASK_ALL	all the ADDRESS2[7:1] bits are masked
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure i2c second slave address */
```

```
i2c_second_address_config(I2C0, 0x82, ADDRESS2_MASK_BIT1_2);
```

### i2c\_second\_address\_disable

The description of i2c\_second\_address\_disable is shown as below:

**Table 3-380. Function i2c\_second\_address\_disable**

Function name	i2c_second_address_disable
Function prototype	void i2c_second_address_disable(uint32_t i2c_periph);
Function descriptions	disable i2c second address in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable i2c second address in slave mode */
```

```
i2c_second_address_disable(I2C0);
```

## i2c\_receved\_address\_get

The description of i2c\_receved\_address\_get is shown as below:

**Table 3-381. Function i2c\_receved\_address\_get**

<b>Function name</b>	i2c_receved_address_get
<b>Function prototype</b>	uint32_t i2c_receved_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=0,1,2)
<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_receved_address_get(I2C0);
```

## i2c\_slave\_byte\_control\_enable

The description of i2c\_slave\_byte\_control\_enable is shown as below:

**Table 3-382. 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=0,1,2)
<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 3-383. 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=0,1,2)
<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 3-384. 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=0,1,2)
<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 3-385. 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)
<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 3-386. 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)
<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 3-387. Function i2c\_enable**

<b>Function name</b>	i2c_enable
<b>Function prototype</b>	void i2c_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 */
i2c_enable(I2C0);
```

## i2c\_disable

The description of i2c\_disable is shown as below:

**Table 3-388. Function i2c\_disable**

<b>Function name</b>	i2c_disable
<b>Function prototype</b>	void i2c_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C0 */
```

```
i2c_disable(I2C0);
```

## i2c\_start\_on\_bus

The description of i2c\_start\_on\_bus is shown as below:

**Table 3-389. Function i2c\_start\_on\_bus**

<b>Function name</b>	i2c_start_on_bus
<b>Function prototype</b>	void i2c_start_on_bus(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a START condition on I2C bus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 send a start condition to I2C bus */
```

```
i2c_start_on_bus(I2C0);
```

## i2c\_stop\_on\_bus

The description of i2c\_stop\_on\_bus is shown as below:

**Table 3-390. Function i2c\_stop\_on\_bus**

<b>Function name</b>	i2c_stop_on_bus
<b>Function prototype</b>	void i2c_stop_on_bus(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a STOP condition on I2C bus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 generate a STOP condition to I2C bus */
```

```
i2c_stop_on_bus(I2C0);
```

## i2c\_data\_transmit

The description of i2c\_data\_transmit is shown as below:

**Table 3-391. Function i2c\_data\_transmit**

<b>Function name</b>	i2c_data_transmit
<b>Function prototype</b>	void i2c_data_transmit(uint32_t i2c_periph, uint32_t data);
<b>Function descriptions</b>	I2C transmit data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>data</b>	transmit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 transmit data */
```

```
i2c_data_transmit(I2C0, 0x80);
```

## i2c\_data\_receive

The description of i2c\_data\_receive is shown as below:

**Table 3-392. Function i2c\_data\_receive**

<b>Function name</b>	i2c_data_receive
<b>Function prototype</b>	uint32_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=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	0x0000..0x00FF

Example:

```
/* I2C0 receive data */
```

```
uint32_t i2c_receiver;
```

```
i2c_receiver = i2c_data_receive(I2C0);
```

### i2c\_reload\_enable

The description of i2c\_reload\_enable is shown as below:

**Table 3-393. 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=0,1,2)
<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 3-394. 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=0,1,2)
<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 3-395. Function i2c\_transfer\_byte\_number\_config**

<b>Function name</b>	i2c_transfer_byte_number_config
<b>Function prototype</b>	void i2c_transfer_byte_number_config(uint32_t i2c_periph, uint32_t byte_number);
<b>Function descriptions</b>	configure number of bytes to be transferred
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>byte_number</b>	0x0-0xFF, number of bytes to be transferred
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure number of bytes to be transferred */
```

```
i2c_transfer_byte_number_config(I2C0, 0xFF);
```

### i2c\_dma\_enable

The description of i2c\_dma\_enable is shown as below:

**Table 3-396. 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=0,1,2)
<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 3-397. 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=0,1,2)
<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 3-398. Function i2c\_pec\_transfer

Function name	i2c_pec_transfer
Function prototype	void i2c_pec_transfer(uint32_t i2c_periph);
Function descriptions	I2C transfers PEC value
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C transfers PEC value */
i2c_pec_transfer(I2C0);
```

### i2c\_pec\_enable

The description of i2c\_pec\_enable is shown as below:

Table 3-399. 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=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C PEC calculation */
i2c_pec_enable(I2C0);
```

### i2c\_pec\_disable

The description of i2c\_pec\_disable is shown as below:

Table 3-400. 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=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C PEC calculation */
i2c_pec_disable(I2C0);
```

### i2c\_pec\_value\_get

The description of i2c\_pec\_value\_get is shown as below:

Table 3-401. 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=0,1,2)
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 3-402. Function i2c\_smbus\_alert\_enable**

<b>Function name</b>	i2c_smbus_alert_enable
<b>Function prototype</b>	void i2c_smbus_alert_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable SMBus Alert
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-403. 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=0,1,2)
<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 3-404. 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=0,1,2)
<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 3-405. 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=0,1,2)
<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 3-406. 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=0,1,2)
<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 3-407. 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=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SMBus Host address */
i2c_smbus_host_addr_disable(I2C0);
```

## i2c\_extented\_clock\_timeout\_enable

The description of i2c\_extented\_clock\_timeout\_enable is shown as below:

**Table 3-408. Function i2c\_extented\_clock\_timeout\_enable**

<b>Function name</b>	i2c_extented_clock_timeout_enable
<b>Function prototype</b>	void i2c_extented_clock_timeout_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable extended clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable extended clock timeout detection */
```

```
i2c_extented_clock_timeout_enable(I2C0);
```

## i2c\_extented\_clock\_timeout\_disable

The description of i2c\_extented\_clock\_timeout\_disable is shown as below:

**Table 3-409. Function i2c\_extented\_clock\_timeout\_disable**

<b>Function name</b>	i2c_extented_clock_timeout_disable
<b>Function prototype</b>	void i2c_extented_clock_timeout_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable extended clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable extended clock timeout detection */
```

```
i2c_extented_clock_timeout_disable(I2C0);
```

## i2c\_clock\_timeout\_enable

The description of i2c\_clock\_timeout\_enable is shown as below:

**Table 3-410. 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=0,1,2)
<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 3-411. 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=0,1,2)
<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 3-412. 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=0,1,2)
<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 3-413. 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=0,1,2)
<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 3-414. 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=0,1,2)
<b>Input parameter{in}</b>	
<b>timeout</b>	bus timeout A
<i>BUSTOA_DETECT_SCL_LOW</i>	BUSTOA is used to detect SCL low timeout
<i>BUSTOA_DETECT_IDLE</i>	BUSTOA is used to detect both SCL and SDA high timeout when the bus is idle
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure idle clock timeout detection */

i2c_idle_clock_timeout_config(I2C0, BUSTOA_DETECT_SCL_LOW);
```

### i2c\_flag\_get

The description of i2c\_flag\_get is shown as below:

**Table 3-415. 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=0,1,2)
<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>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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 3-416. 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



<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>flag</b>	I2C flags
<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_BERR</i>	bus error
<i>I2C_FLAG_LOSTARB</i>	arbitration Lost
<i>I2C_FLAG_OUERR</i>	overflow/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
<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 3-417. 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=0,1,2)
<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

Output parameter{out}	
-	-
Return value	
-	-

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 3-418. Function i2c\_interrupt\_disable**

Function name	i2c_interrupt_disable
Function prototype	void i2c_interrupt_disable(uint32_t i2c_periph, uint32_t interrupt);
Function descriptions	disable I2C interrupt
Precondition	-
The called functions	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
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 3-419. Function `i2c_interrupt_flag_get`

Function name	<code>i2c_interrupt_flag_get</code>
Function prototype	<code>FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);</code>
Function descriptions	get I2C interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
<code>i2c_periph</code>	I2C peripheral
<code>I2Cx</code>	(x=0,1,2)
Input parameter{in}	
<code>int_flag</code>	I2C interrupt flags, refer to <a href="#">Table 3-358. i2c_interrupt_flag_enum</a> .
<code>I2C_INT_FLAG_TI</code>	transmit interrupt flag
<code>I2C_INT_FLAG_RBNE</code>	I2C_RDATA is not empty during receiving interrupt flag
<code>I2C_INT_FLAG_ADDS END</code>	address received matches in slave mode interrupt flag
<code>I2C_INT_FLAG_NACK</code>	not acknowledge interrupt flag
<code>I2C_INT_FLAG_STPD ET</code>	stop condition detected in slave mode interrupt flag
<code>I2C_INT_FLAG_TC</code>	transfer complete in master mode interrupt flag
<code>I2C_INT_FLAG_TCR</code>	transfer complete reload interrupt flag
<code>I2C_INT_FLAG_BERR</code>	bus error interrupt flag
<code>I2C_INT_FLAG_LOSTA RB</code>	arbitration lost interrupt flag
<code>I2C_INT_FLAG_OUER R</code>	overflow/underrun error in slave mode interrupt flag
<code>I2C_INT_FLAG_PEC RR</code>	PEC error interrupt flag
<code>I2C_INT_FLAG_TIMEO UT</code>	timeout interrupt flag
<code>I2C_INT_FLAG_SMBA LT</code>	SMBus Alert interrupt flag
Output parameter{out}	
-	-
Return value	
<code>FlagStatus</code>	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 3-420. Function i2c\_interrupt\_flag\_clear**

<b>Function name</b>	i2c_interrupt_flag_clear
<b>Function prototype</b>	void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	clear I2C interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>int_flag</b>	I2C interrupt flags, refer to <a href="#">Table 3-358. i2c_interrupt_flag_enum</a> .
<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_BERR</i>	bus error interrupt flag
<i>I2C_INT_FLAG_LOSTA RB</i>	arbitration lost interrupt flag
<i>I2C_INT_FLAG_OUER R</i>	overflow/underrun error in slave mode interrupt flag
<i>I2C_INT_FLAG_PECE RR</i>	PEC error interrupt flag
<i>I2C_INT_FLAG_TIMEO UT</i>	timeout interrupt flag
<i>I2C_INT_FLAG_SMBA 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);
```

## 3.16. LPTIMER

The LPTIMER is a 16-bit / 32-bit timer and it is able to keep running in all power modes except for Standby mode with its diversity of clock sources. The LPTIMER registers are listed in chapter [3.16.1](#), the LPTIMER firmware functions are introduced in chapter [3.16.2](#).

### 3.16.1. Descriptions of Peripheral registers

LPTIMER registers are listed in the table shown as below:

**Table 3-421. LPTIMER Registers**

Registers	Descriptions
LPTIMER_INTF	interrupt flag register
LPTIMER_INTC	interrupt flag clear register
LPTIMER_INTEN	interrupt enable register
LPTIMER_CTL0	control register 0
LPTIMER_CTL1	control register 1
LPTIMER_CMPV	compare value register
LPTIMER_CAR	counter auto reload register
LPTIMER_CNT	counter register
LPTIMER_EIRMP	external input remap register
LPTIMER_INHLCMV	input high level counter max value register

### 3.16.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

**Table 3-422. LPTIMER firmware function**

Function name	Function description
lptimer_deinit	deinit the LPTIMER
lptimer_struct_para_init	initialize LPTIMER init parameter struct with a default value
lptimer_init	initialize LPTIMER counter
lptimer_inputremap	configure external input remap
lptimer_register_shadow_enable	enable the LPTIMER_CAR and LPTIMER_CMPV registers shadow function
lptimer_register_shadow_disable	disable the LPTIMER_CAR and LPTIMER_CMPV registers shadow function
lptimer_timeout_enable	enable the LPTIMER TIMEOUT function
lptimer_timeout_disable	disable the LPTIMER TIMEOUT function
lptimer_continue_start	LPTIMER start with countinue mode
lptimer_single_start	LPTIMER start with single mode
lptimer_stop	stop LPTIMER
lptimer_counter_read	read LPTIMER current counter value

Function name	Function description
lptimer_autoreload_read	read LPTIMER auto reload value
lptimer_compare_read	read LPTIMER compare value
lptimer_autoreload_value_config	configure LPTIMER autoreload register value
lptimer_compare_value_config	configure LPTIMER compare value
lptimer_decodemode0_enable	enable decode mode 0
lptimer_decodemode1_enable	enable decode mode 1
lptimer_decodemode_disable	disable decode mode 0/1
lptimer_highlevelcounter_enable	enable external input high level counter
lptimer_highlevelcounter_disable	disable external input high level counter
lptimer_flag_get	get LPTIMER flags
lptimer_flag_clear	clear LPTIMER flags
lptimer_interrupt_enable	enable the LPTIMER interrupt
lptimer_interrupt_disable	disable the LPTIMER interrupt
lptimer_interrupt_flag_get	get LPTIMER interrupt flag
lptimer_interrupt_flag_clear	clear LPTIMER interrupt flag

### Structure lptimer\_parameter\_struct

**Table 3-423. Structure lptimer\_parameter\_struct**

Member name	Function description
clocksource	clock source (LPTIMER_INTERNALCLK, LPTIMER_EXTERNALCLK)
prescaler	counter clock prescaler (LPTIMER_PSC_x, x=1,2,4,8..128)
extclockpolarity	external clock polarity of the active edge for the counter (LPTIMER_EXTERNALCLK_RISING, LPTIMER_EXTERNALCLK_FALLING, LPTIMER_EXTERNALCLK_BOTH)
extclockfilter	external clock sampling time to configure the clock glitch filter (LPTIMER_EXTERNALCLK_FILTEROFF, LPTIMER_EXTERNALCLK_FILTER_2, LPTIMER_EXTERNALCLK_FILTER_4, LPTIMER_EXTERNALCLK_FILTER_8)
triggermode	trigger mode (LPTIMER_TRIGGER_SOFTWARE, LPTIMER_TRIGGER_EXTERNALRISING, LPTIMER_TRIGGER_EXTERNALFALLING, LPTIMER_TRIGGER_EXTERNALBOTH)
extriggersource	external trigger source (LPTIMER_EXTRIGGER_GPIO, LPTIMER_EXTRIGGER_RTCALARM0, LPTIMER_EXTRIGGER_RTCALARM1, LPTIMER_EXTRIGGER_RTCTAMP0, LPTIMER_EXTRIGGER_RTCTAMP1, LPTIMER_EXTRIGGER_RTCTAMP2, LPTIMER_EXTRIGGER_CMP0_OUT, LPTIMER_EXTRIGGER_CMP1_OUT)
extriggerfilter	external trigger filter (LPTIMER_TRIGGER_FILTEROFF, LPTIMER_TRIGGER_FILTER_2, LPTIMER_TRIGGER_FILTER_4, LPTIMER_TRIGGER_FILTER_8)

Member name	Function description
outputpolarity	output polarity (LPTIMER_OUTPUT_NOTINVERTED, LPTIMER_OUTPUT_INVERTED)
outputmode	output mode (LPTIMER_OUTPUT_PWMORSINGLE, LPTIMER_OUTPUT_SET)
countersource	counter source (LPTIMER_COUNTER_INTERNAL, LPTIMER_COUNTER_EXTERNAL)

### lptimer\_deinit

The description of lptimer\_deinit is shown as below:

**Table 3-424. Function lptimer\_deinit**

<b>Function name</b>	lptimer_deinit
<b>Function prototype</b>	void lptimer_deinit(uint32_t lptimer_periph);
<b>Function descriptions</b>	deinit LPTIMER
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
uint32_t lptimer_periph	LPTIMER peripheral
LPTIMER	for GD32L233xx series
LPTIMER0 / 1	for GD32L235xx series
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* deinit LPTIMER */
```

```
lptimer_deinit(LPTIMER);
```

### lptimer\_struct\_para\_init

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

**Table 3-425. Function lptimer\_struct\_para\_init**

<b>Function name</b>	lptimer_struct_para_init
<b>Function prototype</b>	void lptimer_struct_para_init(lptimer_parameter_struct *initpara);
<b>Function descriptions</b>	initialize LPTIMER init parameter struct with a default value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
initpara	LPTIMER init parameter struct, the structure members can refer to <a href="#">Table</a>

	<a href="#">3-423. Structure lptimer_parameter_struct.</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize LPTIMER init parameter struct with a default value */
```

```
lptimer_parameter_struct lptimer_initpara;
```

```
lptimer_struct_para_init(&lptimer_initpara);
```

### lptimer\_init

The description of lptimer\_init is shown as below:

**Table 3-426. Function lptimer\_init**

<b>Function name</b>	lptimer_init
<b>Function prototype</b>	void lptimer_init(uint32_t lptimer_periph, lptimer_parameter_struct *initpara);
<b>Function descriptions</b>	initialize LPTIMER counter
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series
<i>LPTIMER0 / 1</i>	for GD32L235xx series
Input parameter{in}	
<b>initpara</b>	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-423. Structure lptimer_parameter_struct.</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize LPTIMER */
```

```
lptimer_parameter_struct lptimer_structure;
```

```
lptimer_structure.clocksouce = LPTIMER_INTERNALCLK;
```

```
lptimer_structure.prescaler = LPTIMER_PSC_16;
```

```
lptimer_structure.extclockpolarity = LPTIMER_EXTERNALCLK_RISING;
```



```

lptimer_structure.extclockfilter    = LPTIMER_EXTERNALCLK_FILTEROFF;

lptimer_structure.triggermode      = LPTIMER_TRIGGER_SOFTWARE;

lptimer_structure.extriggersource  = LPTIMER_EXTRIGGER_GPIO;

lptimer_structure.extriggerfilter  = LPTIMER_TRIGGER_FILTEROFF;

lptimer_structure.outputpolarity   = LPTIMER_OUTPUT_NOTINVERTED;

lptimer_structure.outputmode       = LPTIMER_OUTPUT_PWMORSINGLE;

lptimer_structure.countersource    = LPTIMER_COUNTER_INTERNAL;

lptimer_init(LPTIMER, lptimer_structure);

```

### lptimer\_inputremap

The description of lptimer\_inputremap is shown as below:

**Table 3-427. Function lptimer\_inputremap**

<b>Function name</b>	lptimer_inputremap
<b>Function prototype</b>	void lptimer_inputremap(uint32_t lptimer_periph, uint32_t input0remap, uint32_t input1remap);
<b>Function descriptions</b>	configure external input remap
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series
<i>LPTIMER0 / 1</i>	for GD32L235xx series
<b>Input parameter{in}</b>	
<b>input0remap</b>	external input0 remap
<i>LPTIMER_INPUT0_GPIO</i> 0	external input is remaped to GPIO
<i>LPTIMER_INPUT0_CMP0_OUT</i>	external input is remaped to CMP0_OUT
<b>Input parameter{in}</b>	
<b>input1remap</b>	external input1 remap
<i>LPTIMER_INPUT1_GPIO</i> 0	external input is remaped to GPIO
<i>LPTIMER_INPUT1_CMP1_OUT</i>	external input is remaped to CMP1_OUT
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LPTIMER inputs is connected to GPIO */

lptimer_inputremap(LPTIMER, LPTIMER_INPUT0_GPIO, LPTIMER_INPUT1_GPIO);
```

### **lptimer\_register\_shadow\_enable**

The description of lptimer\_register\_shadow\_enable is shown as below:

**Table 3-428. Function lptimer\_register\_shadow\_enable**

<b>Function name</b>	lptimer_register_shadow_enable
<b>Function prototype</b>	void lptimer_register_shadow_enable(uint32_t lptimer_periph);
<b>Function descriptions</b>	enable the LPTIMER_CAR and LPTIMER_CMPV registers shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series
<i>LPTIMER0 / 1</i>	for GD32L235xx series
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the LPTIMER_CAR and LPTIMER_CMPV registers shadow function */

lptimer_register_shadow_enable(LPTIMER);
```

### **lptimer\_register\_shadow\_disable**

The description of lptimer\_register\_shadow\_disable is shown as below:

**Table 3-429. Function lptimer\_register\_shadow\_disable**

<b>Function name</b>	lptimer_register_shadow_disable
<b>Function prototype</b>	void lptimer_register_shadow_disable(uint32_t lptimer_periph);
<b>Function descriptions</b>	disable the LPTIMER_CAR and LPTIMER_CMPV registers shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series

<i>LPTIMER0 / 1</i>	for GD32L235xx series
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the LPTIMER_CAR and LPTIMER_CMPV registers shadow function */
```

```
lptimer_register_shadow_disable(LPTIMER);
```

### **lptimer\_timeout\_enable**

The description of lptimer\_timeout\_enable is shown as below:

**Table 3-430. Function lptimer\_timeout\_enable**

<b>Function name</b>	lptimer_timeout_enable
<b>Function prototype</b>	void lptimer_timeout_enable(uint32_t lptimer_periph);
<b>Function descriptions</b>	enable the LPTIMER TIMEOUT function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series
<i>LPTIMER0 / 1</i>	for GD32L235xx series
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the LPTIMER TIMEOUT function */
```

```
lptimer_timeout_enable(LPTIMER);
```

### **lptimer\_timeout\_disable**

The description of lptimer\_timeout\_disable is shown as below:

**Table 3-431. Function lptimer\_timeout\_disable**

<b>Function name</b>	lptimer_timeout_disable
<b>Function prototype</b>	void lptimer_timeout_disable(uint32_t lptimer_periph);
<b>Function descriptions</b>	disable the LPTIMER TIMEOUT function
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series
<i>LPTIMER0 / 1</i>	for GD32L235xx series
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the LPTIMER TIMEOUT function */
```

```
lptimer_timeout_disable(LPTIMER);
```

### **lptimer\_countinue\_start**

The description of lptimer\_countinue\_start is shown as below:

**Table 3-432. Function lptimer\_countinue\_start**

<b>Function name</b>	lptimer_countinue_start
<b>Function prototype</b>	void lptimer_countinue_start(uint32_t lptimer_periph, uint32_t autoreload, uint32_t compare);
<b>Function descriptions</b>	LPTIMER countinue start
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series
<i>LPTIMER0 / 1</i>	for GD32L235xx series
<b>Input parameter{in}</b>	
<b>autoreload</b>	auto reload value, 0x0~0xFFFF for GD32L235xx, 0x0~0xFFFFFFFF for GD32L233xx
<b>Input parameter{in}</b>	
<b>compare</b>	0x0~0xFFFF for GD32L235xx, 0x0~0xFFFFFFFF for GD32L233xx
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* LPTIMER countinue start */
```

```
lptimer_countinue_start(LPTIMER, 0x0000FFFF, 0x00007FFF);
```

## lptimer\_single\_start

The description of lptimer\_single\_start is shown as below:

**Table 3-433. Function lptimer\_single\_start**

<b>Function name</b>	lptimer_single_start
<b>Function prototype</b>	void lptimer_single_start(uint32_t lptimer_periph, uint32_t autoreload, uint32_t compare);
<b>Function descriptions</b>	LPTIMER single start
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series
<i>LPTIMER0 / 1</i>	for GD32L235xx series
<b>Input parameter{in}</b>	
<b>autoreload</b>	auto reload value, 0x0~0xFFFF for GD32L235xx, 0x0~0xFFFFFFFF for GD32L233xx
<b>Input parameter{in}</b>	
<b>compare</b>	0x0~0xFFFF for GD32L235xx, 0x0~0xFFFFFFFF for GD32L233xx
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* LPTIMER single start */
```

```
lptimer_single_start(LPTIMER, 0x0000FFFF, 0x00007FFF);
```

## lptimer\_stop

The description of lptimer\_stop is shown as below:

**Table 3-434. Function lptimer\_stop**

<b>Function name</b>	lptimer_stop
<b>Function prototype</b>	void lptimer_stop(uint32_t lptimer_periph);
<b>Function descriptions</b>	stop LPTIMER
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series
<i>LPTIMER0 / 1</i>	for GD32L235xx series

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* stop LPTIMER */
```

```
lptimer_stop(LPTIMER);
```

### lptimer\_counter\_read

The description of lptimer\_counter\_read is shown as below:

**Table 3-435. Function lptimer\_counter\_read**

Function name	lptimer_counter_read
Function prototype	uint32_t lptimer_counter_read(uint32_t lptimer_periph);
Function descriptions	read LPTIMER current counter value
Precondition	-
The called functions	-
Input parameter{in}	
uint32_t lptimer_periph	LPTIMER peripheral
LPTIMER	for GD32L233xx series
LPTIMER0 / 1	for GD32L235xx series
Output parameter{out}	
-	-
Return value	
uint32_t	counter value (0x0~0xFFFF for GD32L235xx, 0x0~0xFFFFFFFF for GD32L233xx)

Example:

```
/* read LPTIMER current counter value */
```

```
uint32_t i = 0;
```

```
i = lptimer_counter_read(LPTIMER);
```

### lptimer\_autoreload\_read

The description of lptimer\_autoreload\_read is shown as below:

**Table 3-436. Function lptimer\_autoreload\_read**

Function name	lptimer_autoreload_read
Function prototype	uint32_t lptimer_autoreload_read(uint32_t lptimer_periph);
Function descriptions	read LPTIMER auto reload value
Precondition	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series
<i>LPTIMER0 / 1</i>	for GD32L235xx series
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	auto reload value (0x0~0xFFFF for GD32L235xx, 0x0~0xFFFFFFFF for GD32L233xx)

Example:

```

/* read LPTIMER auto reload value */

uint32_t i = 0;

i = lptimer_autoreload_read(LPTIMER);

```

### lptimer\_compare\_read

The description of lptimer\_compare\_read is shown as below:

**Table 3-437. Function lptimer\_compare\_read**

<b>Function name</b>	lptimer_compare_read
<b>Function prototype</b>	uint32_t lptimer_compare_read(uint32_t lptimer_periph);
<b>Function descriptions</b>	read LPTIMER compare value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series
<i>LPTIMER0 / 1</i>	for GD32L235xx series
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	compare value (0x0~0xFFFF for GD32L235xx, 0x0~0xFFFFFFFF for GD32L233xx)

Example:

```

/* read LPTIMER compare value */

uint32_t i = 0;

i = lptimer_compare_read(LPTIMER);

```

## lptimer\_autoreload\_value\_config

The description of lptimer\_autoreload\_value\_config is shown as below:

**Table 3-438. Function lptimer\_autoreload\_value\_config**

<b>Function name</b>	lptimer_autoreload_value_config
<b>Function prototype</b>	void lptimer_autoreload_value_config(uint32_t lptimer_periph, uint32_t autoreload);
<b>Function descriptions</b>	configure LPTIMER autoreload register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series
<i>LPTIMER0 / 1</i>	for GD32L235xx series
<b>Input parameter{in}</b>	
<b>autoreload</b>	autoreload value (0x0~0xFFFF for GD32L235xx, 0x0~0xFFFFFFFF for GD32L233xx)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LPTIMER autoreload register value */
```

```
lptimer_autoreload_value_config(LPTIMER, 0x000000FF);
```

## lptimer\_compare\_value\_config

The description of lptimer\_compare\_value\_config is shown as below:

**Table 3-439. Function lptimer\_compare\_value\_config**

<b>Function name</b>	lptimer_compare_value_config
<b>Function prototype</b>	void lptimer_compare_value_config(uint32_t lptimer_periph, uint32_t compare);
<b>Function descriptions</b>	configure LPTIMER compare value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series
<i>LPTIMER0 / 1</i>	for GD32L235xx series
<b>Input parameter{in}</b>	



<b>counter</b>	compare value (0x0~0xFFFF for GD32L235xx, 0x0~0xFFFFFFFF for GD32L233xx)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LPTIMER compare value */
```

```
lptimer_compare_value_config(LPTIMER, 0x000000FF);
```

### **lptimer\_decodemode0\_enable**

The description of lptimer\_decodemode0\_enable is shown as below:

**Table 3-440. Function lptimer\_decodemode0\_enable**

<b>Function name</b>	lptimer_decodemode0_enable
<b>Function prototype</b>	void lptimer_decodemode0_enable(uint32_t lptimer_periph);
<b>Function descriptions</b>	enable decode mode 0
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series
<i>LPTIMER0 / 1</i>	for GD32L235xx series
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable decode mode 0 */
```

```
lptimer_decodemode0_enable(LPTIMER);
```

### **lptimer\_decodemode1\_enable**

The description of lptimer\_decodemode1\_enable is shown as below:

**Table 3-441. Function lptimer\_decodemode1\_enable**

<b>Function name</b>	lptimer_decodemode1_enable
<b>Function prototype</b>	void lptimer_decodemode1_enable(uint32_t lptimer_periph);
<b>Function descriptions</b>	enable decode mode 1
<b>Precondition</b>	-

The called functions	-
Input parameter{in}	
uint32_t lptimer_periph	LPTIMER peripheral
LPTIMER	for GD32L233xx series
LPTIMER0 / 1	for GD32L235xx series
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable decode mode 1 */
```

```
lptimer_decodemode1_enable(LPTIMER);
```

### lptimer\_decodemode\_disable

The description of lptimer\_decodemode\_disable is shown as below:

**Table 3-442. Function lptimer\_decodemode\_disable**

Function name	lptimer_decodemode_disable
Function prototype	void lptimer_decodemode_disable(uint32_t lptimer_periph);
Function descriptions	disable decode mode 0/1
Precondition	-
The called functions	-
Input parameter{in}	
uint32_t lptimer_periph	LPTIMER peripheral
LPTIMER	for GD32L233xx series
LPTIMER0 / 1	for GD32L235xx series
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable decode mode 0/1 */
```

```
lptimer_decodemode_disable(LPTIMER);
```

### lptimer\_highlevelcounter\_enable

The description of lptimer\_highlevelcounter\_enable is shown as below:

**Table 3-443. Function lptimer\_highlevelcounter\_enable**

Function name	lptimer_highlevelcounter_enable
---------------	---------------------------------

<b>Function prototype</b>	void lptimer_highlevelcounter_enable(uint32_t lptimer_periph, uint32_t maxvalue);
<b>Function descriptions</b>	enable external input high level counter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
uint32_t lptimer_periph	LPTIMER peripheral
LPTIMER	for GD32L233xx series
LPTIMER0 / 1	for GD32L235xx series
<b>Input parameter{in}</b>	
maxvalue	input high level counter max value, 0x0~0x03FFFFFF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable external input high level counter */
```

```
lptimer_highlevelcounter_enable(LPTIMER, 0x00007FFF);
```

### lptimer\_highlevelcounter\_disable

The description of lptimer\_highlevelcounter\_disable is shown as below:

**Table 3-444. Function lptimer\_highlevelcounter\_disable**

<b>Function name</b>	lptimer_highlevelcounter_disable
<b>Function prototype</b>	void lptimer_highlevelcounter_disable(uint32_t lptimer_periph);
<b>Function descriptions</b>	disable external input high level counter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
uint32_t lptimer_periph	LPTIMER peripheral
LPTIMER	for GD32L233xx series
LPTIMER0 / 1	for GD32L235xx series
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable external input high level counter */
```

```
lptimer_highlevelcounter_disable(LPTIMER);
```

## lptimer\_flag\_get

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

**Table 3-445. Function lptimer\_flag\_get**

<b>Function name</b>	lptimer_flag_get
<b>Function prototype</b>	FlagStatus lptimer_flag_get(uint32_t lptimer_periph, uint32_t flag);
<b>Function descriptions</b>	get LPTIMER flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series
<i>LPTIMER0 / 1</i>	for GD32L235xx series
<b>Input parameter{in}</b>	
<b>flag</b>	the LPTIMER flag
<i>LPTIMER_FLAG_CMPVM</i>	compare value register match flag
<i>LPTIMER_FLAG_CARM</i>	counter auto reload register match flag
<i>LPTIMER_FLAG_ETEDV</i>	external trigger edge event flag
<i>LPTIMER_FLAG_CMPVUP</i>	compare value register update flag
<i>LPTIMER_FLAG_CARUP</i>	counter auto reload register update flag
<i>LPTIMER_FLAG_UP</i>	LPTIMER counter direction change down to up flag
<i>LPTIMER_FLAG_DOWN</i>	LPTIMER counter direction change up to down flag
<i>LPTIMER_FLAG_HLCMVUP</i>	input high level counter max value register update flag
<i>LPTIMER_FLAG_INHLCO</i>	LPTIMER_INx(x=0,1) high level counter overflow flag
<i>LPTIMER_FLAG_INHLOE</i>	the high level of LPTIMER_IN0 and LPTIMER_IN1 overlap error flag
<i>LPTIMER_FLAG_INRFOE</i>	the falling and rising edges of LPTIMER_IN0 and LPTIMER_IN1 overlap error flag
<i>LPTIMER_FLAG_IN0E</i>	LPTIMER_IN0 error flag
<i>LPTIMER_FLAG_IN1E</i>	LPTIMER_IN1 error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get LPTIMER flag */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = lptimer_flag_get(LPTIMER, LPTIMER_FLAG_CMPVM);
```

### **lptimer\_flag\_clear**

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

**Table 3-446. Function lptimer\_flag\_clear**

<b>Function name</b>	lptimer_flag_clear
<b>Function prototype</b>	void lptimer_flag_clear(uint32_t lptimer_periph, uint32_t flag);
<b>Function descriptions</b>	clear LPTIMER flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series
<i>LPTIMER0 / 1</i>	for GD32L235xx series
<b>Input parameter{in}</b>	
<b>flag</b>	the LPTIMER flag
<i>LPTIMER_FLAG_CMPVM</i>	compare value register match flag
<i>LPTIMER_FLAG_CARM</i>	counter auto reload register match flag
<i>LPTIMER_FLAG_ETED</i>	external trigger edge event flag
<i>LPTIMER_FLAG_CMPVUP</i>	compare value register update flag
<i>LPTIMER_FLAG_CARUP</i>	counter auto reload register update flag
<i>LPTIMER_FLAG_UP</i>	LPTIMER counter direction change down to up flag
<i>LPTIMER_FLAG_DOWN</i>	LPTIMER counter direction change up to down flag
<i>LPTIMER_FLAG_HLCMVUP</i>	input high level counter max value register update flag
<i>LPTIMER_FLAG_INHLCO</i>	LPTIMER_INx(x=0,1) high level counter overflow flag
<i>LPTIMER_FLAG_INHLOE</i>	the high level of LPTIMER_IN0 and LPTIMER_IN1 overlap error flag
<i>LPTIMER_FLAG_INRFOE</i>	the falling and rising edges of LPTIMER_IN0 and LPTIMER_IN1 overlap error flag
<i>LPTIMER_FLAG_IN0E</i>	LPTIMER_IN0 error flag
<i>LPTIMER_FLAG_IN1E</i>	LPTIMER_IN1 error flag
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* clear LPTIMER flag */
```

```
lptimer_flag_clear(LPTIMER, LPTIMER_FLAG_CMPVM);
```

### lptimer\_interrupt\_enable

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

**Table 3-447. Function lptimer\_interrupt\_enable**

<b>Function name</b>	lptimer_interrupt_enable
<b>Function prototype</b>	void lptimer_interrupt_enable(uint32_t lptimer_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable the LPTIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series
<i>LPTIMER0 / 1</i>	for GD32L235xx series
<b>Input parameter{in}</b>	
<b>interrupt</b>	LPTIMER interrupt source
<i>LPTIMER_INT_CMPVM</i>	compare value register match interrupt
<i>LPTIMER_INT_CARM</i>	counter auto reload register match interrupt
<i>LPTIMER_INT_ETEDEV</i>	external trigger edge event interrupt
<i>LPTIMER_INT_CMPVU P</i>	compare value register update interrupt
<i>LPTIMER_INT_CARUP</i>	counter auto reload register update interrupt
<i>LPTIMER_INT_UP</i>	LPTIMER counter direction change down to up interrupt
<i>LPTIMER_INT_DOWN</i>	LPTIMER counter direction change up to down interrupt
<i>LPTIMER_INT_HLCMV UP</i>	input high level counter max value register update interrupt
<i>LPTIMER_INT_INHLCO</i>	LPTIMER_INx(x=0,1) high level counter overflow interrupt
<i>LPTIMER_INT_INHLOE</i>	the high level of LPTIMER_IN0 and LPTIMER_IN1 overlap error interrupt
<i>LPTIMER_INT_INRFOE</i>	the falling and rising edges of LPTIMER_IN0 and LPTIMER_IN1 overlap error interrupt
<i>LPTIMER_INT_IN0E</i>	LPTIMER_IN0 error interrupt
<i>LPTIMER_INT_IN1E</i>	LPTIMER_IN1 error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* enable the LPTIMER interrupt */
```

```
lptimer_interrupt_enable(LPTIMER, LPTIMER_INT_CMPVM);
```

### **lptimer\_interrupt\_disable**

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

**Table 3-448. Function lptimer\_interrupt\_disable**

<b>Function name</b>	lptimer_interrupt_disable
<b>Function prototype</b>	void lptimer_interrupt_disable(uint32_t lptimer_periph, uint32_t interrupt);
<b>Function descriptions</b>	disable the LPTIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series
<i>LPTIMER0 / 1</i>	for GD32L235xx series
<b>Input parameter{in}</b>	
<b>interrupt</b>	LPTIMER interrupt source
<i>LPTIMER_INT_CMPVM</i>	compare value register match interrupt
<i>LPTIMER_INT_CARM</i>	counter auto reload register match interrupt
<i>LPTIMER_INT_ETEDEV</i>	external trigger edge event interrupt
<i>LPTIMER_INT_CMPVUP</i>	compare value register update interrupt
<i>LPTIMER_INT_CARUP</i>	counter auto reload register update interrupt
<i>LPTIMER_INT_UP</i>	LPTIMER counter direction change down to up interrupt
<i>LPTIMER_INT_DOWN</i>	LPTIMER counter direction change up to down interrupt
<i>LPTIMER_INT_HLCMVUP</i>	input high level counter max value register update interrupt
<i>LPTIMER_INT_INHOCO</i>	LPTIMER_INx(x=0,1) high level counter overflow interrupt
<i>LPTIMER_INT_INHLOE</i>	the high level of LPTIMER_IN0 and LPTIMER_IN1 overlap error interrupt
<i>LPTIMER_INT_INRFOE</i>	the falling and rising edges of LPTIMER_IN0 and LPTIMER_IN1 overlap error interrupt
<i>LPTIMER_INT_IN0E</i>	LPTIMER_IN0 error interrupt
<i>LPTIMER_INT_IN1E</i>	LPTIMER_IN1 error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the LPTIMER interrupt */
```

```
lptimer_interrupt_disable(LPTIMER, LPTIMER_INT_CMPVM);
```

### lptimer\_interrupt\_flag\_get

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

**Table 3-449. Function lptimer\_interrupt\_flag\_get**

Function name	lptimer_interrupt_flag_get
Function prototype	FlagStatus lptimer_interrupt_flag_get(uint32_t lptimer_periph, uint32_t int_flag);
Function descriptions	get LPTIMER interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
uint32_t lptimer_periph	LPTIMER peripheral
LPTIMER	for GD32L233xx series
LPTIMER0 / 1	for GD32L235xx series
Input parameter{in}	
int_flag	the LPTIMER interrupt flag
LPTIMER_INT_FLAG_CMPVM	compare value register match interrupt flag
LPTIMER_INT_FLAG_CARM	counter auto reload register match interrupt flag
LPTIMER_INT_FLAG_ETEDEV	external trigger edge event interrupt flag
LPTIMER_INT_FLAG_CMPVUP	compare value register update interrupt flag
LPTIMER_INT_FLAG_ARUP	counter auto reload register update interrupt flag
LPTIMER_INT_FLAG_UP	LPTIMER counter direction change down to up interrupt flag
LPTIMER_INT_FLAG_DOWN	LPTIMER counter direction change up to down interrupt flag
LPTIMER_INT_FLAG_HLCMVUP	input high level counter max value register update interrupt flag
LPTIMER_INT_FLAG_I_NHLCO	LPTIMER_INx(x=0,1) high level counter overflow interrupt flag
LPTIMER_INT_FLAG_I_NHLOE	the high level of LPTIMER_IN0 and LPTIMER_IN1 overlap error interrupt flag
LPTIMER_INT_FLAG_I_NRFOE	the falling and rising edges of LPTIMER_IN0 and LPTIMER_IN1 overlap error interrupt flag
LPTIMER_INT_FLAG_I	LPTIMER_IN0 error interrupt flag



<i>N0E</i>	
<i>LPTIMER_INT_FLAG_I</i> <i>N1E</i>	LPTIMER_IN1 error interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get LPTIMER interrupt flag */
```

```
FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = lptimer_interrupt_flag_get(LPTIMER, LPTIMER_INT_FLAG_CMPVM);
```

### **lptimer\_interrupt\_flag\_clear**

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

**Table 3-450. Function lptimer\_interrupt\_flag\_clear**

<b>Function name</b>	lptimer_interrupt_flag_clear
<b>Function prototype</b>	void lptimer_interrupt_flag_clear(uint32_t lptimer_periph, uint32_t int_flag);
<b>Function descriptions</b>	clear LPTIMER interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uint32_t lptimer_periph</b>	LPTIMER peripheral
<i>LPTIMER</i>	for GD32L233xx series
<i>LPTIMER0 / 1</i>	for GD32L235xx series
<b>Input parameter{in}</b>	
<b>int_flag</b>	the LPTIMER interrupt flag
<i>LPTIMER_INT_FLAG_C</i> <i>MPVM</i>	compare value register match interrupt flag
<i>LPTIMER_INT_FLAG_C</i> <i>ARM</i>	counter auto reload register match interrupt flag
<i>LPTIMER_INT_FLAG_E</i> <i>TEDEV</i>	external trigger edge event interrupt flag
<i>LPTIMER_INT_FLAG_C</i> <i>MPVUP</i>	compare value register update interrupt flag
<i>LPTIMER_INT_FLAG_C</i> <i>ARUP</i>	counter auto reload register update interrupt flag
<i>LPTIMER_INT_FLAG_U</i> <i>P</i>	LPTIMER counter direction change down to up interrupt flag
<i>LPTIMER_INT_FLAG_D</i> <i>OWN</i>	LPTIMER counter direction change up to down interrupt flag

<i>LPTIMER_INT_FLAG_H</i> <i>LCMVUP</i>	input high level counter max value register update interrupt flag
<i>LPTIMER_INT_FLAG_I</i> <i>NHLCO</i>	LPTIMER_INx(x=0,1) high level counter overflow interrupt flag
<i>LPTIMER_INT_FLAG_I</i> <i>NHLOE</i>	the high level of LPTIMER_IN0 and LPTIMER_IN1 overlap error interrupt flag
<i>LPTIMER_INT_FLAG_I</i> <i>NRFOE</i>	the falling and rising edges of LPTIMER_IN0 and LPTIMER_IN1 overlap error interrupt flag
<i>LPTIMER_INT_FLAG_I</i> <i>N0E</i>	LPTIMER_IN0 error interrupt flag
<i>LPTIMER_INT_FLAG_I</i> <i>N1E</i>	LPTIMER_IN1 error interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear LPTIMER interrupt flag */
```

```
lptimer_interrupt_flag_clear(LPTIMER, LPTIMER_INT_FLAG_CMPVM);
```

## 3.17. LPUART

The Low power Universal Asynchronous Receiver/Transmitter (LPUART) provides a flexible serial data exchange interface. The LPUART registers are listed in chapter [3.17.1](#), the LPUART firmware functions are introduced in chapter [3.17.2](#).

### 3.17.1. Descriptions of Peripheral registers

LPUART registers are listed in the table shown as below:

**Table 3-451. LPUART Registers**

Registers	Descriptions
LPUART_CTL0	Control register 0
LPUART_CTL1	Control register 1
LPUART_CTL2	Control register 2
LPUART_BAUD	Baud rate register
LPUART_CMD	Command register
LPUART_STAT	Status register
LPUART_INTC	Status clear register
LPUART_RDATA	Receive data register
LPUART_TDATA	Transmit data register

Registers	Descriptions
LPUART_CHC	Coherence control register

### 3.17.2. Descriptions of Peripheral functions

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

**Table 3-452. LPUART firmware function**

Function name	Function description
lpuart_deinit	reset LPUART
lpuart_baudrate_set	configure LPUART baud rate value
lpuart_parity_config	configure LPUART parity
lpuart_word_length_set	configure LPUART word length
lpuart_stop_bit_set	configure LPUART stop bit length
lpuart_enable	enable LPUART
lpuart_disable	disable LPUART
lpuart_transmit_config	configure LPUART transmitter
lpuart_receive_config	configure LPUART receiver
lpuart_data_first_config	data is transmitted/received with the LSB/MSB first
lpuart_invert_config	configure LPUART inverted
lpuart_overrun_enable	enable the LPUART overrun function
lpuart_overrun_disable	disable the LPUART overrun function
lpuart_data_transmit	LPUART transmit data function
lpuart_data_receive	LPUART receive data function
lpuart_command_enable	enable LPUART command
lpuart_address_config	configure address of the LPUART
lpuart_address_detection_mode_config	configure address detection mode
lpuart_mute_mode_enable	enable mute mode
lpuart_mute_mode_disable	disable mute mode
lpuart_mute_mode_wakeup_config	configure wakeup method in mute mode
lpuart_halfduplex_enable	enable half-duplex mode
lpuart_halfduplex_disable	disable half-duplex mode
lpuart_hardware_flow_rts_config	configure hardware flow control RTS
lpuart_hardware_flow_cts_config	configure hardware flow control CTS
lpuart_hardware_flow_coherence_config	configure hardware flow control coherence mode
lpuart_rs485_driver_enable	enable RS485 driver
lpuart_rs485_driver_disable	disable RS485 driver
lpuart_driver_asserttime_config	configure driver enable assertion time
lpuart_driver_deasserttime_config	configure driver enable de-assertion time
lpuart_depolarity_config	configure driver enable polarity mode
lpuart_dma_receive_config	configure LPUART DMA for reception

Function name	Function description
lpuart_dma_transmit_config	configure LPUART DMA for transmission
lpuart_reception_error_dma_disable	disable DMA on reception error
lpuart_reception_error_dma_enable	enable DMA on reception error
lpuart_wakeup_enable	enable LPUART to wakeup the mcu from deep-sleep mode
lpuart_wakeup_disable	disable LPUART to wakeup the mcu from deep-sleep mode
lpuart_wakeup_mode_config	configure the LPUART wakeup mode from deep-sleep mode
lpuart_flag_get	get flag in STAT/CHC register
lpuart_flag_clear	clear LPUART status
lpuart_interrupt_enable	enable LPUART interrupt
lpuart_interrupt_disable	disable LPUART interrupt
lpuart_interrupt_flag_get	get LPUART interrupt and flag status
lpuart_interrupt_flag_clear	clear LPUART interrupt flag

### Enum lpuart\_flag\_enum

**Table 3-453. Enum lpuart\_flag\_enum**

Member name	Function description
LPUART_FLAG_REA	receive enable acknowledge flag
LPUART_FLAG_TEA	transmit enable acknowledge flag
LPUART_FLAG_WU	wakeup from Deep-sleep mode flag
LPUART_FLAG_RWU	receiver wakeup from mute mode
LPUART_FLAG_AM	ADDR match flag
LPUART_FLAG_BSY	busy flag
LPUART_FLAG_CTS	CTS level
LPUART_FLAG_CTSF	CTS change flag
LPUART_FLAG_TBE	transmit data buffer empty
LPUART_FLAG_TC	transmission complete
LPUART_FLAG_RBNE	read data buffer not empty
LPUART_FLAG_IDLE	IDLE line detected flag
LPUART_FLAG_ORERR	overflow error
LPUART_FLAG_NERR	noise error flag
LPUART_FLAG_FERR	frame error flag
LPUART_FLAG_PERR	parity error flag
LPUART_FLAG_EPERR	early parity error flag

### Enum lpuart\_interrupt\_flag\_enum

**Table 3-454. Enum lpuart\_interrupt\_flag\_enum**

Member name	Function description
LPUART_INT_FLAG_AM	address match interrupt and flag
LPUART_INT_FLAG_PERR	parity error interrupt and flag
LPUART_INT_FLAG_TBE	transmitter buffer empty interrupt and flag

Member name	Function description
LPUART_INT_FLAG_TC	transmission complete interrupt and flag
LPUART_INT_FLAG_RBNE	read data buffer not empty interrupt and flag
LPUART_INT_FLAG_RBNE_ORE RR	read data buffer not empty interrupt and overrun error flag
LPUART_INT_FLAG_IDLE	IDLE line detected interrupt and flag
LPUART_INT_FLAG_WU	wakeup from deep-sleep mode interrupt and flag
LPUART_INT_FLAG_CTS	CTS interrupt and flag
LPUART_INT_FLAG_ERR_NERR	error interrupt and noise error flag
LPUART_INT_FLAG_ERR_ORER R	error interrupt and overrun error
LPUART_INT_FLAG_ERR_FERR	error interrupt and frame error flag

### Enum lpuart\_interrupt\_enum

**Table 3-455. Enum lpuart\_interrupt\_enum**

Member name	Function description
LPUART_INT_AM	address match interrupt
LPUART_INT_PERR	parity error interrupt
LPUART_INT_TBE	transmitter buffer empty interrupt
LPUART_INT_TC	transmission complete interrupt
LPUART_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt
LPUART_INT_IDLE	IDLE line detected interrupt
LPUART_INT_WU	wakeup from deep-sleep mode interrupt
LPUART_INT_CTS	CTS interrupt
LPUART_INT_ERR	error interrupt

### Enum lpuart\_invert\_enum

**Table 3-456. Enum lpuart\_invert\_enum**

Member name	Function description
LPUART_DINV_ENABLE	data bit level inversion
LPUART_DINV_DISABLE	data bit level not inversion
LPUART_TXPIN_ENABLE	TX pin level inversion
LPUART_TXPIN_DISABLE	TX pin level not inversion
LPUART_RXPIN_ENABLE	RX pin level inversion
LPUART_RXPIN_DISABLE	RX pin level not inversion
LPUART_SWAP_ENABLE	swap TX/RX pins
LPUART_SWAP_DISABLE	not swap TX/RX pins

### lpuart\_deinit

The description of lpuart\_deinit is shown as below:

Table 3-457. Function lpuart\_deinit

<b>Function name</b>	lpuart_deinit
<b>Function prototype</b>	void lpuart_deinit(uint32_t lpuart_periph);
<b>Function descriptions</b>	reset LPUART
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset LPUART0 */
lpuart_deinit(LPUART0);
```

### lpuart\_baudrate\_set

The description of lpuart\_baudrate\_set is shown as below:

Table 3-458. Function lpuart\_baudrate\_set

<b>Function name</b>	lpuart_baudrate_set
<b>Function prototype</b>	void lpuart_baudrate_set(uint32_t lpuart_periph, uint32_t baudval);
<b>Function descriptions</b>	configure LPUART baud rate value
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	
<b>baudval</b>	baud rate value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LPUART0 baud rate value */
lpuart_baudrate_set(LPUART0, 115200);
```

## lpuart\_parity\_config

The description of lpuart\_parity\_config is shown as below:

**Table 3-459. Function lpuart\_parity\_config**

<b>Function name</b>	lpuart_parity_config
<b>Function prototype</b>	void lpuart_parity_config(uint32_t lpuart_periph, uint32_t paritycfg);
<b>Function descriptions</b>	configure LPUART parity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	
<b>paritycfg</b>	configure LPUART parity
<i>LPUART_PM_NONE</i>	no parity
<i>LPUART_PM_ODD</i>	odd parity
<i>LPUART_PM_EVEN</i>	even parity
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LPUART0 parity */
```

```
lpuart_parity_config(LPUART0, LPUART_PM_EVEN);
```

## lpuart\_word\_length\_set

The description of lpuart\_word\_length\_set is shown as below:

**Table 3-460. Function lpuart\_word\_length\_set**

<b>Function name</b>	lpuart_word_length_set
<b>Function prototype</b>	void lpuart_word_length_set(uint32_t lpuart_periph, uint32_t wlen);
<b>Function descriptions</b>	configure LPUART word length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	
<b>wlen</b>	LPUART word length configure
<i>LPUART_WL_7BIT</i>	7 bits
<i>LPUART_WL_8BIT</i>	8 bits

<i>LPUART_WL_9BIT</i>	9 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LPUART0 word length */
```

```
lpuart_word_length_set(LPUART0, LPUART_WL_9BIT);
```

### lpuart\_stop\_bit\_set

The description of lpuart\_stop\_bit\_set is shown as below:

**Table 3-461. Function lpuart\_stop\_bit\_set**

<b>Function name</b>	lpuart_stop_bit_set
<b>Function prototype</b>	void lpuart_stop_bit_set(uint32_t lpuart_periph, uint32_t stblen);
<b>Function descriptions</b>	configure LPUART stop bit length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	
<b>stblen</b>	LPUART stop bit configure
<i>USART_STB_1BIT</i>	1 bit
<i>USART_STB_2BIT</i>	2 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LPUART0 stop bit length */
```

```
lpuart_stop_bit_set(LPUART0, LPUART_STB_2BIT);
```

### lpuart\_enable

The description of lpuart\_enable is shown as below:

**Table 3-462. Function lpuart\_enable**

<b>Function name</b>	lpuart_enable
<b>Function prototype</b>	void lpuart_enable(uint32_t lpuart_periph);



<b>Function descriptions</b>	enable LPUART
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable LPUART0 */
```

```
lpuart_enable(LPUART0);
```

### lpuart\_disable

The description of lpuart\_disable is shown as below:

**Table 3-463. Function lpuart\_disable**

<b>Function name</b>	lpuart_disable
<b>Function prototype</b>	void lpuart_disable(uint32_t lpuart_periph);
<b>Function descriptions</b>	disable LPUART
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable LPUART0 */
```

```
lpuart_disable(LPUART0);
```

### lpuart\_transmit\_config

The description of lpuart\_transmit\_config is shown as below:

**Table 3-464. Function lpuart\_transmit\_config**

<b>Function name</b>	lpuart_transmit_config
----------------------	------------------------

<b>Function prototype</b>	void lpuart_transmit_config(uint32_t lpuart_periph, uint32_t txconfig);
<b>Function descriptions</b>	configure LPUART transmitter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	
<b>txconfig</b>	enable or disable LPUART transmitter
<i>LPUART_TRANSMIT_ENABLE</i>	enable LPUART transmission
<i>LPUART_TRANSMIT_DISABLE</i>	disable LPUART transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LPUART0 transmitter */
```

```
lpuart_transmit_config(LPUART0, LPUART_TRANSMIT_ENABLE);
```

### lpuart\_receive\_config

The description of lpuart\_receive\_config is shown as below:

**Table 3-465. Function lpuart\_receive\_config**

<b>Function name</b>	lpuart_receive_config
<b>Function prototype</b>	void lpuart_receive_config(uint32_t lpuart_periph, uint32_t rxconfig);
<b>Function descriptions</b>	configure LPUART receiver
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	
<b>rxconfig</b>	enable or disable LPUART receiver
<i>LPUART_RECEIVE_ENABLE</i>	enable LPUART reception
<i>LPUART_RECEIVE_DISABLE</i>	disable LPUART reception
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* configure LPUART0 receiver */
```

```
lpuart_receive_config(LPUART0, LPUART_RECEIVE_ENABLE);
```

### lpuart\_data\_first\_config

The description of lpuart\_data\_first\_config is shown as below:

**Table 3-466. Function lpuart\_data\_first\_config**

Function name	lpuart_data_first_config
Function prototype	void lpuart_data_first_config(uint32_t lpuart_periph, uint32_t msbf);
Function descriptions	data is transmitted/received with the LSB/MSB first
Precondition	-
The called functions	-
Input parameter{in}	
lpuart_periph	LPUART peripheral
LPUARTx	(x=0,1) (LPUART1 only for GD32L235xx series)
Input parameter{in}	
msbf	LSB/MSB
LPUART_MSBF_LSB	LSB first
LPUART_MSBF_MSB	MSB first
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure LSB of data first */
```

```
lpuart_data_first_config(LPUART0, LPUART_MSBF_LSB);
```

### lpuart\_invert\_config

The description of lpuart\_invert\_config is shown as below:

**Table 3-467. Function lpuart\_invert\_config**

Function name	lpuart_invert_config
Function prototype	void lpuart_invert_config(uint32_t lpuart_periph, lpuart_invert_enum invertpara);
Function descriptions	configure LPUART inverted
Precondition	-

The called functions	-
Input parameter{in}	
lpuart_periph	LPUART peripheral
LPUARTx	(x=0,1) (LPUART1 only for GD32L235xx series)
Input parameter{in}	
invertpara	refer to <a href="#">Table 3-456. Enum lpuart_invert_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure LPUART0 inversion */
```

```
lpuart_invert_config(LPUART0, LPUART_DINV_ENABLE);
```

### lpuart\_overshoot\_enable

The description of lpuart\_overshoot\_enable is shown as below:

**Table 3-468. Function lpuart\_overshoot\_enable**

Function name	lpuart_overshoot_enable
Function prototype	void lpuart_overshoot_enable(uint32_t lpuart_periph);
Function descriptions	enable the LPUART overshoot
Precondition	-
The called functions	-
Input parameter{in}	
lpuart_periph	LPUART peripheral
LPUARTx	(x=0,1) (LPUART1 only for GD32L235xx series)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable LPUART0 overshoot */
```

```
lpuart_overshoot_enable(LPUART0);
```

### lpuart\_overshoot\_disable

The description of lpuart\_overshoot\_disable is shown as below:

**Table 3-469. Function lpuart\_overshoot\_disable**

Function name	lpuart_overshoot_disable
---------------	--------------------------

<b>Function prototype</b>	void lpuart_overrun_disable(uint32_t lpuart_periph);
<b>Function descriptions</b>	disable the LPUART overrun function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable LPUART0 overrun */
```

```
lpuart_overrun_disable(LPUART0);
```

### lpuart\_data\_transmit

The description of lpuart\_data\_transmit is shown as below:

**Table 3-470. Function lpuart\_data\_transmit**

<b>Function name</b>	lpuart_data_transmit
<b>Function prototype</b>	void lpuart_data_transmit(uint32_t lpuart_periph, uint32_t data);
<b>Function descriptions</b>	LPUART transmit data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	
<b>data</b>	data of transmission (0x00-0xFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* LPUART0 transmit data */
```

```
lpuart_data_transmit(LPUART0, 0xAA);
```

### lpuart\_data\_receive

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

Table 3-471. Function lpuart\_data\_receive

<b>Function name</b>	lpuart_data_receive
<b>Function prototype</b>	uint16_t lpuart_data_receive(uint32_t lpuart_periph);
<b>Function descriptions</b>	LPUART receive data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	data of received (0x0000-0x01FF)

Example:

```
/* LPUART0 receive data */
```

```
uint16_t temp;
```

```
temp = lpuart_data_receive(LPUART0);
```

### lpuart\_command\_enable

The description of lpuart\_command\_enable is shown as below:

Table 3-472. Function lpuart\_command\_enable

<b>Function name</b>	lpuart_command_enable
<b>Function prototype</b>	void lpuart_command_enable(uint32_t lpuart_periph, uint32_t cmdtype);
<b>Function descriptions</b>	enable LPUART command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	
<b>cmdtype</b>	command type
<i>LPUART_CMD_MMCM</i> <i>D</i>	mute mode command
<i>LPUART_CMD_RXFC</i> <i>MD</i>	receive data flush command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable LPUART0 command */
```

```
lpuart_command_enable(LPUART0, LPUART_CMD_MMCMMD);
```

### lpuart\_address\_config

The description of lpuart\_address\_config is shown as below:

**Table 3-473. Function lpuart\_address\_config**

<b>Function name</b>	lpuart_address_config
<b>Function prototype</b>	void lpuart_address_config(uint32_t lpuart_periph, uint8_t addr);
<b>Function descriptions</b>	configure address of the LPUART
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	
<b>addr</b>	address of LPUART (0-0xFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure address of the LPUART0 */
```

```
lpuart_address_config(LPUART0, 0x00);
```

### lpuart\_address\_detection\_mode\_config

The description of lpuart\_address\_detection\_mode\_config is shown as below:

**Table 3-474. Function lpuart\_address\_detection\_mode\_config**

<b>Function name</b>	lpuart_address_detection_mode_config
<b>Function prototype</b>	void lpuart_address_detection_mode_config(uint32_t lpuart_periph, uint32_t addmod);
<b>Function descriptions</b>	configure address detection mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	

<b>addmod</b>	address detection mode
<i>LPUART_ADDM_4BIT</i>	4-bit address detection
<i>LPUART_ADDM_FULL</i> <i>BIT</i>	full-bit address detection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*configure address detection mode */
```

```
lpuart_address_config(LPUART0, LPUART_ADDM_4BIT);
```

### lpuart\_mute\_mode\_enable

The description of lpuart\_mute\_mode\_enable is shown as below:

**Table 3-475. Function lpuart\_mute\_mode\_enable**

<b>Function name</b>	lpuart_mute_mode_enable
<b>Function prototype</b>	void lpuart_mute_mode_enable(uint32_t lpuart_periph);
<b>Function descriptions</b>	enable mute mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable LPUART0 receiver in mute mode */
```

```
lpuart_mute_mode_enable(LPUART0);
```

### lpuart\_mute\_mode\_disable

The description of lpuart\_mute\_mode\_disable is shown as below:

**Table 3-476. Function lpuart\_mute\_mode\_disable**

<b>Function name</b>	lpuart_mute_mode_disable
<b>Function prototype</b>	void lpuart_mute_mode_disable(uint32_t lpuart_periph);
<b>Function descriptions</b>	disable mute mode



<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable LPUART0 receiver in mute mode */
```

```
lpuart_mute_mode_disable(LPUART0);
```

### lpuart\_mute\_mode\_wakeup\_config

The description of lpuart\_mute\_mode\_wakeup\_config is shown as below:

**Table 3-477. Function lpuart\_mute\_mode\_wakeup\_config**

<b>Function name</b>	lpuart_mute_mode_wakeup_config
<b>Function prototype</b>	void lpuart_mute_mode_wakeup_config(uint32_t lpuart_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>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	
<b>wmethod</b>	two methods be used to enter or exit the mute mode
<i>LPUART_WM_IDLE</i>	idle line
<i>LPUART_WM_ADDR</i>	address mask
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LPUART0 wakeup method in mute mode */
```

```
lpuart_mute_mode_wakeup_config(LPUART0, LPUART_WM_IDLE);
```

## lpuart\_halfduplex\_enable

The description of lpuart\_halfduplex\_enable is shown as below:

**Table 3-478. Function lpuart\_halfduplex\_enable**

<b>Function name</b>	lpuart_halfduplex_enable
<b>Function prototype</b>	void lpuart_halfduplex_enable(uint32_t lpuart_periph);
<b>Function descriptions</b>	enable half-duplex mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable LPUART0 half duplex mode*/
```

```
lpuart_halfduplex_enable(LPUART0);
```

## lpuart\_halfduplex\_disable

The description of lpuart\_halfduplex\_disable is shown as below:

**Table 3-479. Function lpuart\_halfduplex\_disable**

<b>Function name</b>	lpuart_halfduplex_disable
<b>Function prototype</b>	void lpuart_halfduplex_disable(uint32_t lpuart_periph);
<b>Function descriptions</b>	disable half-duplex mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable LPUART0 half duplex mode*/
```

```
lpuart_halfduplex_disable(LPUART0);
```

## lpuart\_hardware\_flow\_rts\_config

The description of lpuart\_hardware\_flow\_rts\_config is shown as below:

**Table 3-480. Function lpuart\_hardware\_flow\_rts\_config**

<b>Function name</b>	lpuart_hardware_flow_rts_config
<b>Function prototype</b>	void lpuart_hardware_flow_rts_config(uint32_t lpuart_periph, uint32_t rtsconfig);
<b>Function descriptions</b>	configure hardware flow control RTS
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	
<b>rtsconfig</b>	enable or disable RTS
<i>LPUART_RTS_ENABLE</i>	enable RTS
<i>LPUART_RTS_DISABLE</i>	disable RTS
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LPUART0 hardware flow control RTS */
```

```
lpuart_hardware_flow_rts_config(LPUART0, LPUART_RTS_ENABLE);
```

## lpuart\_hardware\_flow\_cts\_config

The description of lpuart\_hardware\_flow\_cts\_config is shown as below:

**Table 3-481. Function lpuart\_hardware\_flow\_cts\_config**

<b>Function name</b>	lpuart_hardware_flow_cts_config
<b>Function prototype</b>	void lpuart_hardware_flow_cts_config(uint32_t lpuart_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>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	

<b>ctsconfig</b>	enable or disable CTS
<i>LPUART_CTS_ENABL</i> <i>E</i>	enable CTS
<i>LPUART_CTS_DISABL</i> <i>E</i>	disable CTS
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LPUART0 hardware flow control CTS */
```

```
lpuart_hardware_flow_cts_config(LPUART0, LPUART_CTS_ENABLE);
```

### lpuart\_hardware\_flow\_coherence\_config

The description of lpuart\_hardware\_flow\_coherence\_config is shown as below:

**Table 3-482. Function lpuart\_hardware\_flow\_coherence\_config**

<b>Function name</b>	lpuart_hardware_flow_coherence_config
<b>Function prototype</b>	void lpuart_hardware_flow_coherence_config(uint32_t lpuart_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>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>hcm</b>	
<b>hcm</b>	Hardware flow control coherence mode
<i>LPUART_HCM_NONE</i>	nRTS signal equals to the RBNE status register
<i>LPUART_HCM_EN</i>	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 */
```

```
lpuart_hardware_flow_coherence_config(LPUART0, LPUART_HCM_NONE);
```

## lpuart\_rs485\_driver\_enable

The description of lpuart\_rs485\_driver\_enable is shown as below:

**Table 3-483. Function lpuart\_rs485\_driver\_enable**

<b>Function name</b>	lpuart_rs485_driver_enable
<b>Function prototype</b>	void lpuart_rs485_driver_enable(uint32_t lpuart_periph);
<b>Function descriptions</b>	enable RS485 driver
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable LPUART0 RS485 driver */
```

```
lpuart_rs485_driver_enable(LPUART0);
```

## lpuart\_rs485\_driver\_disable

The description of lpuart\_rs485\_driver\_disable is shown as below:

**Table 3-484. Function lpuart\_rs485\_driver\_disable**

<b>Function name</b>	lpuart_rs485_driver_disable
<b>Function prototype</b>	void lpuart_rs485_driver_disable(uint32_t lpuart_periph);
<b>Function descriptions</b>	disable RS485 driver
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable LPUART0 RS485 driver */
```

```
lpuart_rs485_driver_disable(LPUART0);
```

## lpuart\_driver\_asserttime\_config

The description of lpuart\_driver\_asserttime\_config is shown as below:

**Table 3-485. Function lpuart\_driver\_asserttime\_config**

<b>Function name</b>	lpuart_driver_asserttime_config
<b>Function prototype</b>	void lpuart_driver_asserttime_config(uint32_t lpuart_periph, uint32_t deatime);
<b>Function descriptions</b>	configure driver enable assertion time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>deatime</b>	
	driver enable assertion time (0x00-0x0000001F)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set LPUART0 driver asserttime */
```

```
lpuart_driver_asserttime_config(LPUART0, 0x0000001F);
```

## lpuart\_driver\_deasserttime\_config

The description of lpuart\_driver\_deasserttime\_config is shown as below:

**Table 3-486. Function lpuart\_driver\_deasserttime\_config**

<b>Function name</b>	lpuart_driver_deasserttime_config
<b>Function prototype</b>	void lpuart_driver_deasserttime_config(uint32_t lpuart_periph, uint32_t lpuart_periph, uint32_t dedtime);
<b>Function descriptions</b>	configure driver enable de-assertion time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>dedtime</b>	
	driver enable de-assertion time (0x00-0x0000001F)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* set LPUART0 driver deasserttime */

lpuart_driver_deasserttime_config(LPUART0, 0x0000001F);
```

### lpuart\_depolarity\_config

The description of lpuart\_depolarity\_config is shown as below:

**Table 3-487. Function lpuart\_depolarity\_config**

<b>Function name</b>	lpuart_depolarity_config
<b>Function prototype</b>	void lpuart_depolarity_config(uint32_t lpuart_periph, uint32_t dep);
<b>Function descriptions</b>	configure driver enable polarity mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	
<b>dep</b>	DE signal
<i>LPUART_DEP_HIGH</i>	DE signal is active high
<i>LPUART_DEP_LOW</i>	DE signal is active low
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure driver enable polarity mode */

lpuart_driver_depolarity_config(LPUART0, LPUART_DEP_HIGH);
```

### lpuart\_dma\_receive\_config

The description of lpuart\_dma\_receive\_config is shown as below:

**Table 3-488. Function lpuart\_dma\_receive\_config**

<b>Function name</b>	lpuart_dma_receive_config
<b>Function prototype</b>	void lpuart_dma_receive_config(uint32_t lpuart_periph, uint32_t dmamcmd);
<b>Function descriptions</b>	configure LPUART DMA for reception
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	
<b>dmacmd</b>	enable or disable DMA for reception
<i>LPUART_DENR_ENABLE</i>	DMA enable for reception
<i>LPUART_DENR_DISABLE</i>	DMA disable for reception
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* LPUART0 DMA enable for reception */
```

```
lpuart_dma_receive_config(LPUART0, LPUART_DENR_ENABLE);
```

### lpuart\_dma\_transmit\_config

The description of lpuart\_dma\_transmit\_config is shown as below:

**Table 3-489. Function lpuart\_dma\_transmit\_config**

<b>Function name</b>	lpuart_dma_transmit_config
<b>Function prototype</b>	void lpuart_dma_transmit_config(uint32_t lpuart_periph, uint32_t dmacmd);
<b>Function descriptions</b>	configure LPUART DMA for transmission
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>dmacmd</b>	
<i>LPUART_DENT_ENABLE</i>	DMA enable for transmission
<i>LPUART_DENT_DISABLE</i>	DMA disable for transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* LPUART0 DMA enable for transmission */
```



```
lpuart_dma_transmit_config(LPUART0, LPUART_DENT_ENABLE);
```

### lpuart\_reception\_error\_dma\_disable

The description of lpuart\_reception\_error\_dma\_disable is shown as below:

**Table 3-490. Function lpuart\_reception\_error\_dma\_disable**

<b>Function name</b>	lpuart_reception_error_dma_disable
<b>Function prototype</b>	void lpuart_reception_error_dma_disable(uint32_t lpuart_periph);
<b>Function descriptions</b>	disable DMA on reception error
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA on reception error */
```

```
lpuart_reception_error_dma_disable(LPUART0);
```

### lpuart\_reception\_error\_dma\_enable

The description of lpuart\_reception\_error\_dma\_enable is shown as below:

**Table 3-491. Function lpuart\_reception\_error\_dma\_enable**

<b>Function name</b>	lpuart_reception_error_dma_enable
<b>Function prototype</b>	void lpuart_reception_error_dma_enable(uint32_t lpuart_periph);
<b>Function descriptions</b>	enable DMA on reception error
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMA on reception error */
```

```
lpuart_reception_error_dma_enable(LPUART0);
```

## lpuart\_wakeup\_enable

The description of lpuart\_wakeup\_enable is shown as below:

**Table 3-492. Function lpuart\_wakeup\_enable**

<b>Function name</b>	lpuart_wakeup_enable
<b>Function prototype</b>	void lpuart_wakeup_enable(uint32_t lpuart_periph);
<b>Function descriptions</b>	enable LPUART to wakeup the mcu from deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* LPUART0 wake up enable */
```

```
lpuart_wakeup_enable(LPUART0);
```

## lpuart\_wakeup\_disable

The description of lpuart\_wakeup\_disable is shown as below:

**Table 3-493. Function lpuart\_wakeup\_disable**

<b>Function name</b>	lpuart_wakeup_disable
<b>Function prototype</b>	void lpuart_wakeup_disable(uint32_t lpuart_periph);
<b>Function descriptions</b>	disable LPUART to wakeup the mcu from deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* LPUART0 wake up disable */
```

```
lpuart_wakeup_disable(LPUART0);
```

## lpuart\_wakeup\_mode\_config

The description of lpuart\_wakeup\_mode\_config is shown as below:

**Table 3-494. Function lpuart\_wakeup\_mode\_config**

<b>Function name</b>	lpuart_wakeup_mode_config
<b>Function prototype</b>	void lpuart_wakeup_mode_config(uint32_t lpuart_periph, uint32_t wum);
<b>Function descriptions</b>	configure the LPUART wakeup mode from deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	
<b>wum</b>	wakeup mode
<i>LPUART_WUM_ADDR</i>	WUF active on address match
<i>LPUART_WUM_STAR TB</i>	WUF active on start bit
<i>LPUART_WUM_RBNE</i>	WUF active on RBNE
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LPUART0 wake up mode */
```

```
lpuart_wakeup_mode_config(LPUART0, LPUART_WUM_ADDR);
```

## lpuart\_flag\_get

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

**Table 3-495. Function lpuart\_flag\_get**

<b>Function name</b>	lpuart_flag_get
<b>Function prototype</b>	FlagStatus lpuart_flag_get(uint32_t lpuart_periph, lpuart_flag_enum flag);
<b>Function descriptions</b>	get flag in STAT/CHC register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	

<b>flag</b>	LPUART flags, refer to <a href="#">Table 3-453. Enum lpuart_flag_enum</a> only one among these parameters can be selected
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get flag LPUART0 state */
```

```
FlagStatus status;
```

```
status = lpuart_flag_get(LPUART0, LPUART_FLAG_TBE);
```

### lpuart\_flag\_clear

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

**Table 3-496. Function lpuart\_flag\_clear**

<b>Function name</b>	lpuart_flag_clear
<b>Function prototype</b>	void lpuart_flag_clear(uint32_t lpuart_periph, lpuart_flag_enum flag);
<b>Function descriptions</b>	clear LPUART status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	
<b>flag</b>	LPUART flags, refer to <a href="#">Table 3-453. Enum lpuart_flag_enum</a> only one among these parameters can be selected
<i>LPUART_FLAG_PERR</i>	parity error flag
<i>LPUART_FLAG_FERR</i>	frame error flag
<i>LPUART_FLAG_NERR</i>	noise detected flag
<i>LPUART_FLAG_ORERR</i>	overrun error flag
<i>LPUART_FLAG_IDLE</i>	idle line detected flag
<i>LPUART_FLAG_TC</i>	transmission complete flag
<i>LPUART_FLAG_CTSF</i>	CTS change flag
<i>LPUART_FLAG_AM</i>	address match flag
<i>LPUART_FLAG_WU</i>	wakeup from deep-sleep mode flag
<i>LPUART_FLAG_EPERR</i>	early parity error flag
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* clear LPUART0 flag */
```

```
lpuart_flag_clear(LPUART0, LPUART_FLAG_TC);
```

## lpuart\_interrupt\_enable

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

**Table 3-497. Function lpuart\_interrupt\_enable**

<b>Function name</b>	lpuart_interrupt_enable
<b>Function prototype</b>	void lpuart_interrupt_enable(uint32_t lpuart_periph, lpuart_interrupt_enum interrupt);
<b>Function descriptions</b>	enable LPUART interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt type, refer to <a href="#">Table 3-455. Enum lpuart_interrupt_enum</a> only one among these parameters can be selected
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable LPUART0 TBE interrupt */
```

```
lpuart_interrupt_enable(LPUART0, LPUART_INT_TBE);
```

## lpuart\_interrupt\_disable

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

**Table 3-498. Function lpuart\_interrupt\_disable**

<b>Function name</b>	lpuart_interrupt_disable
<b>Function prototype</b>	void lpuart_interrupt_disable(uint32_t lpuart_periph, lpuart_interrupt_enum interrupt);
<b>Function descriptions</b>	disable LPUART interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
Input parameter{in}	
<b>interrupt</b>	interrupt type, refer to <a href="#">Table 3-455. Enum lpuart_interrupt_enum</a> only one among these parameters can be selected
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable LPUART0 TBE interrupt */
```

```
lpuart_interrupt_disable(LPUART0, LPUART_INT_TBE);
```

### lpuart\_interrupt\_flag\_get

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

**Table 3-499. Function lpuart\_interrupt\_flag\_get**

<b>Function name</b>	lpuart_interrupt_flag_get
<b>Function prototype</b>	FlagStatus lpuart_interrupt_flag_get(uint32_t lpuart_periph, lpuart_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	get LPUART interrupt and flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
Input parameter{in}	
<b>int_flag</b>	LPUART interrupt flag, refer to <a href="#">Table 3-454. Enum lpuart_interrupt_flag_enum</a> , only one among these parameters can be selected
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the LPUART0 interrupt flag status */
```

```
FlagStatus status;
```

```
status = lpuart_interrupt_flag_get(LPUART0, LPUART_INT_FLAG_RBNE);
```

## lpuart\_interrupt\_flag\_clear

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

**Table 3-500. Function lpuart\_interrupt\_flag\_clear**

<b>Function name</b>	lpuart_interrupt_flag_clear
<b>Function prototype</b>	void lpuart_interrupt_flag_clear(uint32_t lpuart_periph, lpuart_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	clear LPUART interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_periph</b>	LPUART peripheral
<i>LPUARTx</i>	(x=0,1) (LPUART1 only for GD32L235xx series)
<b>Input parameter{in}</b>	
<b>int_flag</b>	LPUART interrupt flag, refer to <a href="#">Table 3-454. Enum lpuart_interrupt_flag_enum</a> , only one among these parameters can be selected
<i>LPUART_INT_FLAG_PERR</i>	parity error flag
<i>LPUART_INT_FLAG_ERR_FERR</i>	frame error flag
<i>LPUART_INT_FLAG_ERR_NERR</i>	noise detected flag
<i>LPUART_INT_FLAG_RBNE_ORERR</i>	read data buffer not empty interrupt and overrun error flag
<i>LPUART_INT_FLAG_ERR_ORERR</i>	error interrupt and overrun error
<i>LPUART_INT_FLAG_IDLE</i>	idle line detected flag
<i>LPUART_INT_FLAG_TC</i>	transmission complete flag
<i>LPUART_INT_FLAG_CTS</i>	CTS change flag
<i>LPUART_INT_FLAG_ADDR_M</i>	address match flag
<i>LPUART_INT_FLAG_WU</i>	wakeup from deep-sleep mode flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the LPUART0 interrupt flag */

lpuart_interrupt_flag_clear(LPUART0, LPUART_INT_FLAG_TC);
```

## 3.18. MISC

MISC is a software package that provide the interfaces for NVIC and SysTick. The NVIC and SysTick registers are listed in chapter [3.18.1](#), the MISC firmware functions are introduced in chapter [3.18.2](#).

### 3.18.1. Descriptions of Peripheral registers

**Table 3-501. 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
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 3-502. 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



### 3.18.2. Descriptions of Peripheral functions

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

**Table 3-503. MISC firmware function**

Function name	Function description
<code>nvic_irq_enable</code>	enable NVIC interrupt request
<code>nvic_irq_disable</code>	disable NVIC interrupt request
<code>nvic_system_reset</code>	initiates a system reset request to reset the MCU
<code>nvic_vector_table_set</code>	set the NVIC vector table base address
<code>system_lowpower_set</code>	set the state of the low power mode
<code>system_lowpower_reset</code>	reset the state of the low power mode
<code>systick_clksource_set</code>	set the systick clock source

### Enum IRQn\_Type

**Table 3-504. IRQn\_Type for GD32L233xx devices**

Member name	Function description
<code>WWDGT_IRQn</code>	window watchDog timer interrupt
<code>LVD_IRQn</code>	LVD through EXTI line detect interrupt
<code>TAMPER_STAMP_IRQn</code>	RTC Tamper and Timestamp from EXTI interrupt
<code>RTC_WKUP_IRQn</code>	RTC Wakeup interrupt
<code>FMC_IRQn</code>	FMC interrupt
<code>RCU_CTC_IRQn</code>	RCU and CTC interrupt
<code>EXTI0_IRQn</code>	EXTI line 0 interrupts
<code>EXTI1_IRQn</code>	EXTI line 1 interrupts
<code>EXTI2_IRQn</code>	EXTI line 2 interrupts
<code>EXTI3_IRQn</code>	EXTI line 3 interrupts
<code>EXTI4_IRQn</code>	EXTI line 4 interrupts
<code>DMA_Channel0_IRQn</code>	DMA channel 0 interrupt
<code>DMA_Channel1_IRQn</code>	DMA channel 1 interrupts
<code>DMA_Channel2_IRQn</code>	DMA channel 2 interrupts
<code>DMA_Channel3_IRQn</code>	DMA channel 3 interrupts
<code>DMA_Channel4_IRQn</code>	DMA channel 4 interrupts
<code>DMA_Channel5_IRQn</code>	DMA channel 5 interrupts
<code>DMA_Channel6_IRQn</code>	DMA channel 6 interrupts
<code>ADC_IRQn</code>	ADC interrupts
<code>USBD_HP_IRQn</code>	USBD High Priority
<code>USBD_LP_IRQn</code>	USBD Low Priority
<code>TIMER1_IRQn</code>	TIMER1 interrupt
<code>TIMER2_IRQn</code>	TIMER2 interrupt
<code>TIMER8_IRQn</code>	TIMER8 interrupt
<code>TIMER11_IRQn</code>	TIMER11 interrupt

Member name	Function description
TIMER5_IRQn	TIMER5 interrupt
TIMER6_IRQn	TIMER6 interrupt
USART0_IRQn	USART0 interrupt
USART1_IRQn	USART1 interrupt
UART3_IRQn	UART3 interrupt
UART4_IRQn	UART4 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
DAC_IRQn	DAC interrupt
I2C2_EV_IRQn	I2C2 event interrupt
I2C2_ER_IRQn	I2C2 error interrupt
RTC_Alarm_IRQn	RTC Alarm interrupt
USBD_WKUP_IRQn	USBD Wakeup interrupt
EXTI5_9_IRQn	EXTI line 5 to 9 interrupts
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
I2C2_WKUP_IRQn	I2C2 Wakeup interrupt
USART0_WKUP_IRQn	USART0 Wakeup interrupt
LPUART_IRQn	LPUART global interrupt
CAU_IRQn	CAU interrupt
TRNG_IRQn	TRNG interrupt
SLCD_IRQn	SLCD interrupt
USART1_WKUP_IRQn	USART1 Wakeup interrupt
I2C1_WKUP_IRQn	I2C1 Wakeup interrupt
LPUART_WKUP_IRQn	LPUART Wakeup interrupt
LPTIMER_IRQn	LPTIMER interrupt

**Table 3-505. IRQn\_Type for GD32L235xx devices**

Member name	Function description
WWDGT_IRQn	window watchDog timer interrupt
LVD_IRQn	LVD through EXTI line detect interrupt
TAMPER_STAMP_IRQn	RTC Tamper and Timestamp from EXTI interrupt
RTC_WKUP_IRQn	RTC Wakeup interrupt
FMC_IRQn	FMC interrupt
RCU_CTC_IRQn	RCU and CTC interrupt

Member name	Function description
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
DMA_Channel3_IRQn	DMA channel 3 interrupts
DMA_Channel4_IRQn	DMA channel 4 interrupts
DMA_Channel5_IRQn	DMA channel 5 interrupts
DMA_Channel6_IRQn	DMA channel 6 interrupts
ADC_IRQn	ADC interrupts
USBD_HP_CAN_TX_IRQn	USBD High Priority or CAN TX
USBD_LP_CAN_RX0_IRQn	USBD Low Priority or CAN RX0
TIMER1_IRQn	TIMER1 interrupt
TIMER2_IRQn	TIMER2 interrupt
TIMER8_IRQn	TIMER8 interrupt
TIMER11_IRQn	TIMER11 interrupt
TIMER5_IRQn	TIMER5 interrupt
TIMER6_IRQn	TIMER6 interrupt
USART0_IRQn	USART0 interrupt
USART1_IRQn	USART1 interrupt
UART3_IRQn	UART3 interrupt
UART4_IRQn	UART4 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
DAC_IRQn	DAC interrupt
I2C2_EV_IRQn	I2C2 event interrupt
I2C2_ER_IRQn	I2C2 error interrupt
RTC_Alarm_IRQn	RTC Alarm interrupt
USBD_WKUP_IRQn	USBD Wakeup interrupt
EXTI5_9_IRQn	EXTI line 5 to 9 interrupts
TIMER0_TRG_CMT_UP_BRK_IRQn	TIMER0 trigger and Channel commutation or update or break interrupt
TIMER0_Channel_IRQn	TIMER0 capture compare interrupt
TIMER14_IRQn	TIMER14 interrupt

Member name	Function description
EXTI10_15_IRQn	EXTI line 10 to 15 interrupts
TIMER40_IRQn	TIMER40 interrupt
CAN_RX1_IRQn	CAN RX1 interrupt
CAN_EWMC_IRQn	CAN EWMC interrupt
DMAMUX_IRQn	DMAMUX interrupt
CMP0_IRQn	Comparator 0 interrupt
CMP1_IRQn	Comparator 1 interrupt
I2C0_WKUP_IRQn	I2C0 Wakeup interrupt
I2C2_WKUP_IRQn	I2C2 Wakeup interrupt
USART0_WKUP_IRQn	USART0 Wakeup interrupt
LPUART0_IRQn	LPUART0 global interrupt
CAU_IRQn	CAU interrupt
TRNG_IRQn	TRNG interrupt
SLCD_IRQn	SLCD interrupt
USART1_WKUP_IRQn	USART1 Wakeup interrupt
I2C1_WKUP_IRQn	I2C1 Wakeup interrupt
LPUART0_WKUP_IRQn	LPUART0 Wakeup interrupt
LPTIMER0_IRQn	LPTIMER0 interrupt
LPUART1_WKUP_IRQn	LPUART1 Wakeup interrupt
LPTIMER1_IRQn	LPTIMER1 interrupt
LPUART1_IRQn	LPUART1 global interrupt

## nvic\_irq\_enable

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

**Table 3-506. Function nvic\_irq\_enable**

Function name	nvic_irq_enable
Function prototype	void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority);
Function descriptions	enable NVIC interrupt request
Precondition	-
The called functions	NVIC_SetPriority、NVIC_EnableIRQ
Input parameter{in}	
nvic_irq	NVIC interrupt, for GD32L233xx devices which refer to enum <a href="#">Table 3-504. IRQn Type for GD32L233xx devices</a> or for GD32L235xx devices <a href="#">Table 3-505. IRQn Type for GD32L235xx devices</a>
Input parameter{in}	
nvic_irq_pre_priority	the pre-emption priority needed to set
Output parameter{out}	
-	-
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 3-507. 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>	
nvic_irq	NVIC interrupt, for GD32L233xx devices which refer to enum <a href="#">Table 3-504. IRQn Type for GD32L233xx devices</a> or for GD32L235xx devices <a href="#">Table 3-505. IRQn Type for GD32L235xx devices</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable window watchDog timer interrupt */
nvic_irq_disable(WWDGT_IRQn);
```

### nvic\_system\_reset

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

**Table 3-508. 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();
```

### nvic\_vector\_table\_set

The description of nvic\_vector\_table\_set is shown as below:

**Table 3-509. 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
<i>H</i>	
<b>Input parameter{in}</b>	
<b>offset</b>	vector table offset (vector table start address= base address+offset)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set vector table address = NVIC_VECTTAB_FLASH + 0x200 */
nvic_vector_table_set(NVIC_VECTTAB_FLASH, 0x200);
```

### system\_lowpower\_set

The description of system\_lowpower\_set is shown as below:

**Table 3-510. 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

<code>SCB_LPM_SLEEP_EXIT_ISR</code>	if chose this para, the system always enter low power mode by exiting from ISR
<code>SCB_LPM_DEEPSLEEP_P</code>	if chose this para, the system will enter the DEEPSLEEP mode
<code>SCB_LPM_WAKE_BY_ALL_INT</code>	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 3-511. 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
<code>SCB_LPM_SLEEP_EXIT_ISR</code>	if chose this para, the system will exit low power mode by exiting from ISR
<code>SCB_LPM_DEEPSLEEP_P</code>	if chose this para, the system will enter the SLEEP mode
<code>SCB_LPM_WAKE_BY_ALL_INT</code>	if chose this para, the lowpower mode only can be woke up by the enable interrupts
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* the system will exit low power mode by exiting from ISR */
system_lowpower_reset(SCB_LPM_SLEEP_EXIT_ISR);
```

## systick\_clksource\_set

The description of systick\_clksource\_set is shown as below:

**Table 3-512. Function systick\_clksource\_set**

<b>Function name</b>	systick_clksource_set
<b>Function prototype</b>	void systick_clksource_set(uint32_t systick_clksource);
<b>Function descriptions</b>	set the systick clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>systick_clksource</b>	the systick clock source needed to choose
SYSTICK_CLKSOURC E_HCLK	systick clock source is from HCLK
SYSTICK_CLKSOURC E_HCLK_DIV8	systick clock source is from HCLK/8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* systick clock source is HCLK/8 */
```

```
systick_clksource_set(SYSTICK_CLKSOURCE_HCLK_DIV8);
```

## 3.19. PMU

According to the Power management unit (PMU), provides ten types of power saving modes, including Run, Run1, Run2, Sleep, Sleep1, Sleep2, Deep-sleep, Deep-sleep 1, Deep-sleep 2 and Standby mode. The PMU registers are listed in chapter [3.19.1](#), the PMU firmware functions are introduced in chapter [3.19.2](#).

### 3.19.1. Descriptions of Peripheral registers

PMU registers are listed in the table shown as below:

**Table 3-513. PMU Registers**

Registers	Descriptions
PMU_CTL0	PMU control 0 register
PMU_CS	PMU control and status register
PMU_CTL1	PMU control 1 register
PMU_STAT	PMU status register
PMU_PAR	PMU parameter register



### 3.19.2. Descriptions of Peripheral functions

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

**Table 3-514. PMU firmware function**

Function name	Function description
pmu_deinit	reset PMU
pmu_lvd_select	select low voltage detector threshold
pmu_lvd_disable	disable PMU lvd
pmu_ldo_output_select	select LDO output voltage
pmu_vc_enable	enable VBAT battery charging
pmu_vc_disable	disable VBAT battery charging
pmu_vcr_select	select PMU VBAT battery charging resistor
pmu_low_power_enable	enable low power in Run/Sleep mode
pmu_low_power_disable	disable low power in Run/Sleep mode
pmu_to_sleepmode	PMU work in sleep mode
pmu_to_deepsleepmode	PMU work at Deep-sleep/Deep-sleep 1/Deep-sleep 2 mode
pmu_to_standbymode	pmu work at standby mode
pmu_wakeup_pin_enable	enable PMU wakeup pin
pmu_wakeup_pin_disable	disable PMU wakeup pin
pmu_backup_write_enable	enable backup domain write
pmu_backup_write_disable	disable backup domain write
pmu_sram_power_config	configure power state of SRAM1
pmu_core1_power_config	configure power state of COREOFF1 domain (only for GD32L233xx series)
pmu_eflash_sleep_power_config	configure power state of eflash domain when in Run / Run1 / Run2 / Run3 mode (only for GD32L235xx series)
pmu_eflash_deepsleep_power_config	configure power state of eflash domain when in Deep-sleep / Deep-sleep 1 / Deep-sleep 2 mode (only for GD32L235xx series)
pmu_deepsleep2_retention_enable	have retention register in Deep-sleep 2
pmu_deepsleep2_retention_disable	no retention register in Deep-sleep 2
pmu_deepsleep2_sram_power_config	configure SRAM1 power state when enter Deep-sleep 2
pmu_deepsleep_wait_time_config	configure IRC16M counter before enter Deep-sleep mode
pmu_wakeuptime_core1_software_enable	use software value signal when wake up COREOFF1 domain (only for GD32L233xx series)
pmu_wakeuptime_core1_software_disable	use hardware ack signal when wake up COREOFF1 domain (only for GD32L233xx series)
pmu_wakeuptime_eflash_config	configure eflash wake up counter (only for GD32L235xx series)
pmu_wakeuptime_sram_config	configure SRAM1 wakeup time
pmu_wakeuptime_sram_software_enable	use software value signal when wake up SRAM1 (only for GD32L233xx series)

Function name	Function description
pmu_wakeuptime_sram_software_disable	use hardware ack signal when wake up SRAM1 (only for GD32L233xx series)
pmu_wakeuptime_deepsleep2_software_enable	use software value signal when wake up Deep-sleep 2
pmu_wakeuptime_deepsleep2_software_disable	use hardware ack signal when wake up Deep-sleep 2
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 3-515. 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\_lvd\_select

The description of pmu\_lvd\_select is shown as below:

**Table 3-516. Function pmu\_lvd\_select**

Function name	pmu_lvd_select
Function prototype	void pmu_lvd_select(uint32_t lvd_t_n);
Function descriptions	select low voltage detector threshold
Precondition	-
The called functions	-
Input parameter{in}	
lvd_t_n	voltage threshold value
PMU_LVDT_0	voltage threshold is 2.1V

<i>PMU_LVDT_1</i>	voltage threshold is 2.3V
<i>PMU_LVDT_2</i>	voltage threshold is 2.4V
<i>PMU_LVDT_3</i>	voltage threshold is 2.6V
<i>PMU_LVDT_4</i>	voltage threshold is 2.7V
<i>PMU_LVDT_5</i>	voltage threshold is 2.9V
<i>PMU_LVDT_6</i>	voltage threshold is 3.0V
<i>PMU_LVDT_7</i>	input analog voltage on PB7 (compared with 0.8V)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select low voltage detector threshold as 3.0V */
```

```
pmu_lvd_select(PMU_LVDT_6);
```

### pmu\_lvd\_disable

The description of pmu\_lvd\_disable is shown as below:

**Table 3-517. Function pmu\_lvd\_disable**

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

Example:

```
/* disable PMU lvd */
```

```
pmu_lvd_disable();
```

### pmu\_ldo\_output\_select

The description of pmu\_ldo\_output\_select is shown as below:

**Table 3-518. Function pmu\_ldo\_output\_select**

<b>Function name</b>	pmu_ldo_output_select
----------------------	-----------------------

<b>Function prototype</b>	void pmu_ldo_output_select(uint32_t ldo_output);
<b>Function descriptions</b>	select LDO output voltage
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ldo_output</b>	output voltage mode
<i>PMU_LDOVS_LOW</i>	LDO output voltage low mode, only for GD32L233xx series
<i>PMU_LDOVS_HIGH</i>	LDO output voltage high mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select output low voltage mode */
pmu_ldo_output_select(PMU_LDOVS_LOW);
```

### pmu\_vc\_enable

The description of pmu\_vc\_enable is shown as below:

**Table 3-519. Function pmu\_vc\_enable**

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

Example:

```
/* enable VBAT battery charging */
pmu_vc_enable();
```

### pmu\_vc\_disable

The description of pmu\_vc\_disable is shown as below:

Table 3-520. Function pmu\_vc\_disable

Function name	pmu_vc_disable
Function prototype	void pmu_vc_disable(void);
Function descriptions	disable VBAT battery charging
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable VBAT battery charging */
pmu_vc_disable();
```

### pmu\_vcr\_select

The description of pmu\_vcr\_select is shown as below:

Table 3-521. Function pmu\_vcr\_select

Function name	pmu_vcr_select
Function prototype	void pmu_vcr_select(uint32_t resistor);
Function descriptions	select PMU VBAT battery charging resistor
Precondition	-
The called functions	-
Input parameter{in}	
resistor	VBAT battery charging resistor
PMU_VCRSEL_5K	5 kOhms resistor is selected for charging VBAT battery
PMU_VCRSEL_1P5K	1.5 kOhms resistor is selected for charging VBAT battery
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select PMU VBAT battery charging resistor */
pmu_vcr_select(PMU_VCRSEL_5K);
```

### pmu\_low\_power\_enable

The description of pmu\_low\_power\_enable is shown as below:

**Table 3-522. Function pmu\_low\_power\_enable**

<b>Function name</b>	pmu_low_power_enable
<b>Function prototype</b>	void pmu_low_power_enable(void);
<b>Function descriptions</b>	enable low power in Run/Sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable low power in Run/Sleep mode */
pmu_low_power_enable();
```

### pmu\_low\_power\_disable

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

**Table 3-523. Function pmu\_low\_power\_disable**

<b>Function name</b>	pmu_low_power_disable
<b>Function prototype</b>	void pmu_low_power_disable(void);
<b>Function descriptions</b>	disable low power in Run/Sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable low power in Run/Sleep mode */
pmu_low_power_disable();
```

### pmu\_to\_sleepmode

The description of pmu\_to\_sleepmode is shown as below:

Table 3-524. Function pmu\_to\_sleepmode

<b>Function name</b>	pmu_to_sleepmode
<b>Function prototype</b>	void pmu_to_sleepmode(uint32_t lowdrive, uint8_t sleepmodecmd);
<b>Function descriptions</b>	PMU work at sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lowdrive</b>	NPLDO work mode
<i>PMU_LDNPDSP_NOR MALDRIVE</i>	low-driver mode disable
<i>PMU_LDNPDSP_LOW DRIVE</i>	low-driver mode enable
<b>Input parameter{in}</b>	
<b>sleepmodecmd</b>	command to enter sleep mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work at sleep mode */
```

```
pmu_to_sleepmode(PMU_LDNP_LOWDRIVE, WFI_CMD);
```

### pmu\_to\_deepsleepmode

The description of pmu\_to\_deepsleepmode is shown as below:

Table 3-525. Function pmu\_to\_deepsleepmode

<b>Function name</b>	pmu_to_deepsleepmode
<b>Function prototype</b>	void pmu_to_deepsleepmode(uint32_t lowdrive, uint8_t deepsleepmodecmd, uint8_t deepsleepmode);
<b>Function descriptions</b>	PMU work at Deep-sleep/Deep-sleep 1/Deep-sleep 2 mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lowdrive</b>	NPLDO work mode
<i>PMU_LDNPDSP_NOR MALDRIVE</i>	low-driver mode disable
<i>PMU_LDNPDSP_LOW DRIVE</i>	low-driver mode enable
<b>Input parameter{in}</b>	

<b>deepsleepmodecmd</b>	command to enter Deep-sleep/Deep-sleep 1/Deep-sleep 2 mode
<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
<i>PMU_DEEPSLEEP2</i>	Deep-sleep mode 2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work at Deep-sleep mode */
```

```
pmu_to_deepsleepmode(PMU_LDNPDSP_NORMALDRIVE, WFI_CMD, PMU_DEEPSLEEP);
```

### pmu\_to\_standbymode

The description of pmu\_to\_standbymode is shown as below:

**Table 3-526. 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 at standby mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work at standby mode */
```

```
pmu_to_standby(WFI_CMD);
```

### pmu\_wakeup\_pin\_enable

The description of pmu\_wakeup\_pin\_enable is shown as below:



Table 3-527. Function pmu\_wakeup\_pin\_enable

<b>Function name</b>	pmu_wakeup_pin_enable
<b>Function prototype</b>	void pmu_wakeup_pin_enable(uint32_t wakeup_pin);
<b>Function descriptions</b>	enable wakeup pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_pin</b>	Wakeup pin
<i>PMU_WAKEUP_PIN0</i>	WKUP Pin 0 (PA0)
<i>PMU_WAKEUP_PIN1</i>	WKUP Pin 1 (PC13)
<i>PMU_WAKEUP_PIN2</i>	WKUP Pin 2 (PA2)
<i>PMU_WAKEUP_PIN3</i>	WKUP Pin 3 (PB2)
<i>PMU_WAKEUP_PIN4</i>	WKUP Pin 4 (PC6)
<i>PMU_WAKEUP_PIN5</i>	WKUP Pin 5 (PB5) (only for GD32L235xx series)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable wakeup pin0 */
```

```
pmu_wakeup_pin_enable(PMU_WAKEUP_PIN0);
```

### pmu\_wakeup\_pin\_disable

The description of pmu\_wakeup\_pin\_disable is shown as below:

Table 3-528. Function pmu\_wakeup\_pin\_disable

<b>Function name</b>	pmu_wakeup_pin_disable
<b>Function prototype</b>	void pmu_wakeup_pin_disable(uint32_t wakeup_pin);
<b>Function descriptions</b>	disable wakeup pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_pin</b>	Wakeup pin
<i>PMU_WAKEUP_PIN0</i>	WKUP Pin 0 (PA0)
<i>PMU_WAKEUP_PIN1</i>	WKUP Pin 1 (PC13)
<i>PMU_WAKEUP_PIN2</i>	WKUP Pin 2 (PA2)
<i>PMU_WAKEUP_PIN3</i>	WKUP Pin 3 (PB2)
<i>PMU_WAKEUP_PIN4</i>	WKUP Pin 4 (PC6)
<i>PMU_WAKEUP_PIN5</i>	WKUP Pin 5 (PB5) (only for GD32L235xx series)
<b>Output parameter{out}</b>	
-	-

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 3-529. Function pmu\_backup\_write\_enable**

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

Example:

```
/* enable backup domain write */
```

```
pmu_backup_write_enable();
```

### pmu\_backup\_write\_disable

The description of pmu\_backup\_write\_disable is shown as below:

**Table 3-530. 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\_sram\_power\_config

The description of pmu\_sram\_power\_config is shown as below:

**Table 3-531. Function pmu\_sram\_power\_config**

<b>Function name</b>	pmu_sram_power_config
<b>Function prototype</b>	void pmu_sram_power_config(uint32_t state);
<b>Function descriptions</b>	configure power state of SRAM1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>state</b>	power state of SRAM1
<i>PMU_SRAM1_SLEEP</i>	SRAM1 go to power-off
<i>PMU_SRAM1_WAKE</i>	SRAM1 wakeup
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure power state of SRAM1 */
pmu_sram_power_config(PMU_SRAM1_WAKE);
```

### pmu\_core1\_power\_config (only for GD32L233xx series)

The description of pmu\_core1\_power\_config is shown as below:

**Table 3-532. Function pmu\_core1\_power\_config**

<b>Function name</b>	pmu_core1_power_config
<b>Function prototype</b>	void pmu_core1_power_config(uint32_t state);
<b>Function descriptions</b>	configure power state of COREOFF1 domain
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>state</b>	power state of COREOFF1 domain
<i>PMU_CORE1_SLEEP</i>	COREOFF1 domain go to power-off
<i>PMU_CORE1_WAKE</i>	COREOFF1 domain wakeup

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure power state of COREOFF1 domain */
pmu_core1_power_config(PMU_SRAM1_WAKE);
```

### pmu\_deepsleep2\_retention\_enable (only for GD32L233xx series)

The description of pmu\_deepsleep2\_retention\_enable is shown as below:

**Table 3-533. Function pmu\_deepsleep2\_retention\_enable**

Function name	pmu_deepsleep2_retention_enable
Function prototype	void pmu_deepsleep2_retention_enable(void);
Function descriptions	have retention register in Deep-sleep 2
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* have retention register in Deep-sleep 2 */
pmu_deepsleep2_retention_enable();
```

### pmu\_deepsleep2\_retention\_disable (only for GD32L233xx series)

The description of pmu\_deepsleep2\_retention\_disable is shown as below:

**Table 3-534. Function pmu\_deepsleep2\_retention\_disable**

Function name	pmu_deepsleep2_retention_disable
Function prototype	void pmu_deepsleep2_retention_disable(void);
Function descriptions	no retention register in Deep-sleep 2
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* no retention register in Deep-sleep 2 */
```

```
pmu_deepsleep2_retention_disable();
```

### pmu\_eflash\_sleep\_power\_config (only for GD32L235xx series)

The description of pmu\_eflash\_sleep\_power\_config is shown as below:

**Table 3-535. Function pmu\_eflash\_sleep\_power\_config**

<b>Function name</b>	pmu_eflash_sleep_power_config
<b>Function prototype</b>	void pmu_eflash_sleep_power_config(uint32_t state);
<b>Function descriptions</b>	configure power state of eflash domain when in Run mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>state</b>	power state of eflash domain
PMU_EFLASH_SLEEP_ON	eflash domain power on in Run mode
PMU_EFLASH_SLEEP_OFF	eflash domain power off in Run mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure power state of eflash domain when in Run mode */
```

```
pmu_eflash_sleep_power_config(PMU_EFLASH_SLEEP_ON);
```

### pmu\_eflash\_deepsleep\_power\_config (only for GD32L235xx series)

The description of pmu\_eflash\_deepsleep\_power\_config is shown as below:

**Table 3-536. 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(uint32_t state);
<b>Function descriptions</b>	configure power state of eflash domain when in Deep-sleep / Deep-sleep 1 / Deep-sleep 2 mode
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>state</b>	power state of eflash domain
<i>PMU_EFLASH_DSPSL_EEP_ON</i>	eflash power on when in Deep-sleep / Deep-sleep 1 / Deep-sleep 2 mode
<i>PMU_EFLASH_DSPSL_EEP_OFF</i>	eflash power off when in Deep-sleep / Deep-sleep 1 / Deep-sleep 2 mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure power state of eflash domain when in Deep-sleep / Deep-sleep 1 / Deep-sleep 2 mode */
```

```
pmu_eflash_deepsleep_power_config(PMU_EFLASH_DSPSL_EEP_ON);
```

### pmu\_deepsleep2\_sram\_power\_config

The description of pmu\_deepsleep2\_sram\_power\_config is shown as below:

**Table 3-537. Function pmu\_deepsleep2\_sram\_power\_config**

<b>Function name</b>	pmu_deepsleep2_sram_power_config
<b>Function prototype</b>	void pmu_deepsleep2_sram_power_config(uint32_t state);
<b>Function descriptions</b>	configure SRAM1 power state when enter Deep-sleep 2
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>state</b>	power state of SRAM1
<i>PMU_SRAM1_POWER_OFF</i>	SRAM1 power-off
<i>PMU_SRAM1_POWER_REMAIN</i>	SRAM1 power same as Run / Run1 / Run2 mode for GD32L233xx series and Run mode for GD32L235xx series
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SRAM1 power state when enter Deep-sleep 2 */
```

```
pmu_deepsleep2_sram_power_config(PMU_SRAM1_POWER_OFF);
```

## pmu\_deepsleep\_wait\_time\_config

The description of pmu\_deepsleep\_wait\_time\_config is shown as below:

**Table 3-538. 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 IRC16M counter before enter Deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wait_time</b>	IRC16M counter before enter Deep-sleep mode
<i>uint32_t</i>	0x0~0x1F
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure IRC16M counter before enter Deep-sleep mode */
```

```
pmu_deepsleep_wait_time_config(1);
```

## pmu\_wakeuptime\_core1\_software\_enable (only for GD32L233xx series)

The description of pmu\_wakeuptime\_core1\_software\_enable is shown as below:

**Table 3-539. Function pmu\_wakeuptime\_core1\_software\_enable**

<b>Function name</b>	pmu_wakeuptime_core1_software_enable
<b>Function prototype</b>	void pmu_wakeuptime_core1_software_enable(uint32_t wakeup_time);
<b>Function descriptions</b>	use software value signal when wake up COREOFF1 domain
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_time</b>	wakeup time of power switch of COREOFF1 domain
<i>uint32_t</i>	0x0~0xFF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* use software value signal when wake up COREOFF1 domain */
```

```
pmu_wakeuptime_core1_software_enable(1);
```

### pmu\_wakeuptime\_core1\_software\_disable (only for GD32L233xx series)

The description of pmu\_wakeuptime\_core1\_software\_disable is shown as below:

**Table 3-540. Function pmu\_wakeuptime\_core1\_software\_disable**

<b>Function name</b>	pmu_wakeuptime_core1_software_disable
<b>Function prototype</b>	void pmu_wakeuptime_core1_software_disable(void);
<b>Function descriptions</b>	use hardware ack signal when wake up COREOFF1 domain
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* use hardware ack signal when wake up COREOFF1 domain */
pmu_wakeuptime_core1_software_disable();
```

### pmu\_wakeuptime\_eflash\_config (only for GD32L235xx series)

The description of pmu\_wakeuptime\_eflash\_config is shown as below:

**Table 3-541. Function pmu\_wakeuptime\_eflash\_config**

<b>Function name</b>	pmu_wakeuptime_eflash_config
<b>Function prototype</b>	void pmu_wakeuptime_eflash_config(uint32_t wait_time);
<b>Function descriptions</b>	configure eflash wake up counter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wait_time</b>	IRC16M counter, 0x0~0xFF
<b>uint32_t</b>	0x0~0xFF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure eflash wake up counter */
pmu_wakeuptime_eflash_config(2);
```



## pmu\_wakeuptime\_sram\_config

The description of pmu\_wakeuptime\_sram\_config is shown as below:

**Table 3-542. Function pmu\_wakeuptime\_sram\_config**

<b>Function name</b>	pmu_wakeuptime_sram_config
<b>Function prototype</b>	void pmu_wakeuptime_sram_config(uint32_t wakeup_time);
<b>Function descriptions</b>	configure SRAM1 wakeup time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_time</b>	wakeup time of power switch of SRAM1
<i>uint32_t</i>	0x0~0xFF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SRAM1 wakeup time */
```

```
pmu_wakeuptime_sram_config(2);
```

## pmu\_wakeuptime\_sram\_software\_enable (only for GD32L233xx series)

The description of pmu\_wakeuptime\_sram\_software\_enable is shown as below:

**Table 3-543. Function pmu\_wakeuptime\_sram\_software\_enable**

<b>Function name</b>	pmu_wakeuptime_sram_software_enable
<b>Function prototype</b>	void pmu_wakeuptime_sram_software_enable(void);
<b>Function descriptions</b>	use software value signal when wake up SRAM1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* use software value signal when wake up SRAM1 */
```

```
pmu_wakeuptime_sram_software_enable();
```

### pmu\_wakeuptime\_sram\_software\_disable (only for GD32L233xx series)

The description of pmu\_wakeuptime\_sram\_software\_disable is shown as below:

**Table 3-544. Function pmu\_wakeuptime\_sram\_software\_disable**

<b>Function name</b>	pmu_wakeuptime_sram_software_disable
<b>Function prototype</b>	void pmu_wakeuptime_sram_software_disable(void);
<b>Function descriptions</b>	use hardware ack signal when wake up SRAM1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* use hardware ack signal when wake up SRAM1 */
pmu_wakeuptime_sram_software_disable();
```

### pmu\_wakeuptime\_deepsleep2\_software\_enable

The description of pmu\_wakeuptime\_deepsleep2\_software\_enable is shown as below:

**Table 3-545. Function pmu\_wakeuptime\_deepsleep2\_software\_enable**

<b>Function name</b>	pmu_wakeuptime_deepsleep2_software_enable
<b>Function prototype</b>	void pmu_wakeuptime_deepsleep2_software_enable(uint32_t wakeup_time);
<b>Function descriptions</b>	use software value signal when wake up Deep-sleep 2
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_time</b>	wakeup time of power switch of COREOFF0 domain
<i>uint32_t</i>	0x0~0xFF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* use software value signal when wake up Deep-sleep 2 */
pmu_wakeuptime_deepsleep2_software_enable(1);
```

## pmu\_wakeuptime\_deepsleep2\_software\_disable

The description of pmu\_wakeuptime\_deepsleep2\_software\_disable is shown as below:

**Table 3-546. Function pmu\_wakeuptime\_deepsleep2\_software\_disable**

<b>Function name</b>	pmu_wakeuptime_deepsleep2_software_disable
<b>Function prototype</b>	void pmu_wakeuptime_deepsleep2_software_disable(void);
<b>Function descriptions</b>	use hardware ack signal when wake up Deep-sleep 2
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* use hardware ack signal when wake up Deep-sleep 2 */
```

```
pmu_wakeuptime_deepsleep2_software_disable();
```

## pmu\_flag\_get

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

**Table 3-547. 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_LVD</i>	lvd flag
<i>PMU_FLAG_LDOVSRF</i>	LDO voltage select ready flag
<i>PMU_FLAG_NPRDY</i>	normal-power LDO ready flag
<i>PMU_FLAG_LPRDY</i>	low-power LDO ready flag
<i>PMU_FLAG_SRAM1_SLEEP</i>	SRAM1 is in sleep state flag
<i>PMU_FLAG_SRAM1_ACTIVE</i>	SRAM1 is in active state flag
<i>PMU_FLAG_EFLASHP</i>	eflash domain is in sleep state flag (only for GD32L235xx series)

<i>S_SLEEP</i>	
<i>PMU_FLAG_EFLASHP</i> <i>S_ACTIVE</i>	eflash domain is in active state flag (only for GD32L235xx series)
<i>PMU_FLAG_CORE1_S</i> <i>LEEP</i>	COREOFF1 domain is in sleep state flag (only for GD32L233xx series)
<i>PMU_FLAG_CORE1_A</i> <i>CTIVE</i>	COREOFF1 domain is in active state flag (only for GD32L233xx series)
<i>PMU_FLAG_DEEPSLE</i> <i>EP_2</i>	Deep-sleep 2 mode status flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

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

```
FlagStatus status;
```

```
status = pmu_flag_get(PMU_FLAG_WAKEUP);
```

### pmu\_flag\_clear

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

**Table 3-548. 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
<i>PMU_FLAG_WAKEUP</i>	wakeup flag
<i>PMU_FLAG_STANDBY</i>	standby flag
<i>PMU_FLAG_DEEPSLE</i> <i>EP_2</i>	Deep-sleep 2 mode status flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear flag bit */
```

```
pmu_flag_clear(PMU_FLAG_STANDBY);
```

## 3.20. 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 [3.20.1](#), the RCU firmware functions are introduced in chapter [3.20.2](#).

### 3.20.1. Descriptions of Peripheral registers

**Table 3-549. RCU Registers**

Registers	Descriptions
RCU_CTL	control register
RCU_CFG0	clock configuration register 0
RCU_INT	clock interrupt register
RCU_APB2RST	APB2 reset register
RCU_APB1RST	APB1 reset register
RCU_AHBEN	AHB enable register
RCU_APB2EN	APB2 enable register
RCU_APB1EN	APB1 enable register
RCU_BDCTL	backup domain control register
RCU_RSTSCK	reset source/clock register
RCU_AHBRST	AHB reset register
RCU_CFG1	clock configuration register 1
RCU_CFG2	clock configuration register 2
RCU_ANB2EN	AHB2 enable register
RCU_AHB2RST	AHB2 reset register
RCU_VKEY	voltage key register
RCU_LPB	low-power bandgap mode register

### 3.20.2. Descriptions of Peripheral functions

**Table 3-550. RCU firmware function**

Function name	Function description
rcu_deinit	deinitialize the RCU
rcu_periph_clock_enable	enable the peripherals clock
rcu_periph_clock_disable	disable the peripherals clock
rcu_periph_clock_sleep_enable	enable the peripherals clock when in sleep mode
rcu_periph_clock_sleep_disable	disable the peripherals clock when in sleep mode
rcu_periph_reset_enable	enable the peripherals reset
rcu_periph_reset_disable	disable the peripheral reset

Function name	Function description
rcu_bkp_reset_enable	enable the BKP domain reset
rcu_bkp_reset_disable	disable the BKP domain reset
rcu_system_clock_source_config	configure the system clock source
rcu_system_clock_source_get	get the system clock source
rcu_ahb_clock_config	configure the AHB clock prescaler selection
rcu_apb1_clock_config	configure the APB1 clock prescaler selection
rcu_apb2_clock_config	configure the APB2 clock prescaler selection
rcu_adc_clock_config	configure the ADC clock source and prescaler selection
rcu_ckout_config	configure the CK_OUT clock source and divider
rcu_pll_config	configure the main PLL clock
rcu_usart_clock_config	configure the usart clock
rcu_i2c_clock_config	configure the I2Cx(x=0,1,2) clock source selection
rcu_lptimer_clock_config	configure the LPTIMER clock source selection
rcu_lpuart_clock_config	configure the LPUART clock source selection
rcu_irc16mdiv_clock_config	configure the IRC16MDIV clock selection
rcu_usbd_clock_config	configure the USB D clock source selection
rcu_rtc_clock_config	configure the RTC/SLCD clock source selection
rcu_pll_source_ck_prediv_config	configure PLL source clocks pre-divider
rcu_lxtal_drive_capability_config	configure the LXTAL drive capability
rcu_lp_bandgap_config	configure low power bandgap mode selection
rcu_oscstabil_wait	wait for oscillator stabilization flags is SET or oscillator startup is timeout
rcu_oscstabil_on	turn on the oscillator
rcu_oscstabil_off	turn off the oscillator
rcu_oscstabil_bypass_mode_enable	enable the oscillator bypass mode
rcu_oscstabil_bypass_mode_disable	disable the oscillator bypass mode
rcu_irc16m_adjust_value_set	set the IRC16M adjust value
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_voltage_key_unlock	unlock the voltage key
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 all 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 3-551. Enum rcu\_periph\_enum**

enum name	Function description
RCU_DMA	DMA clock
RCU_CAU	CAU clock
RCU_TRNG	TRNG clock
RCU_CRC	CRC clock
RCU_GPIOA	GPIOA clock
RCU_GPIOB	GPIOB clock
RCU_GPIOC	GPIOC clock
RCU_GPIOD	GPIOD clock
RCU_GPIOF	GPIOF clock
RCU_SYSCFG	SYSCFG clock
RCU_CMP	CMP clock
RCU_ADC	ADC clock
RCU_TIMER8	TIMER8 clock
RCU_SPI0	SPI0 clock
RCU_USART0	USART0 clock
RCU_DBGMCU	DBGMCU clock
RCU_TIMER1	TIMER1 clock
RCU_TIMER2	TIMER2 clock
RCU_TIMER5	TIMER5 clock
RCU_TIMER6	TIMER6 clock
RCU_TIMER11	TIMER11 clock
RCU_LPTIMER	LPTIMER clock (only for GD32L233xx series)
RCU_LPTIMER0	LPTIMER0 clock (only for GD32L235xx series)
RCU_LPTIMER1	LPTIMER1 clock (only for GD32L235xx series)
RCU_SLCD	SLCD clock
RCU_WWDGT	WWDGT clock
RCU_SPI1	SPI1 clock
RCU_USART1	USART1 clock
RCU_LPUART	LPUART clock (only for GD32L233xx series)
RCU_LPUART0	LPUART0 clock (only for GD32L235xx series)
RCU_LPUART1	LPUART1 clock (only for GD32L235xx series)
RCU_CAN	CAN clock (only for GD32L235xx series)
RCU_UART3	UART3 clock
RCU_UART4	UART4 clock
RCU_I2C0	I2C0 clock
RCU_I2C1	I2C1 clock
RCU_USBD	USBD clock
RCU_I2C2	I2C2 clock

enum name	Function description
RCU_PMU	PMU clock
RCU_DAC	DAC clock
RCU_CTC	CTC clock
RCU_BKP	BKP clock
RCU_RTC	RTC clock

### Enum rcu\_periph\_sleep\_enum

**Table 3-552. Enum rcu\_periph\_sleep\_enum**

enum name	Function description
RCU_SRAM0_SLP	SRAM0 clock
RCU_FMC_SLP	FMC clock
RCU_SRAM1_SLP	SRAM1 clock

### Enum rcu\_periph\_reset\_enum

**Table 3-553. Enum rcu\_periph\_reset\_enum**

enum name	Function description
RCU_CAURST	SRAM0 clock
RCU_TRNGRST	TRNG reset
RCU_CRCRST	CRC reset
RCU_GPIOARST	GPIOA reset
RCU_GPIOBRST	GPIOB reset
RCU_GPIOCRST	GPIOC reset
RCU_GPIODRST	GPIOD reset
RCU_GPIOFRST	GPIOF reset
RCU_SYSCFGRST	SYSCFG reset
RCU_CMPRST	CMP reset
RCU_ADCRST	ADC reset
RCU_TIMER8RST	TIMER8 reset
RCU_SPI0RST	SPI0 reset
RCU_USART0RST	USART0 reset
RCU_TIMER1RST	TIMER1 reset
RCU_TIMER2RST	TIMER2 reset
RCU_TIMER5RST	TIMER5 reset
RCU_TIMER6RST	TIMER6 reset
RCU_TIMER11RST	TIMER11 reset
RCU_LPTIMERRST	LPTIMER reset (only for GD32L233xx series)
RCU_LPTIMER0RST	LPTIMER0 reset (only for GD32L235xx series)
RCU_LPTIMER1RST	LPTIMER1 reset (only for GD32L235xx series)
RCU_SLCDRST	SLCD reset
RCU_WWDGTRST	WWDGT reset



enum name	Function description
RCU_SPI1RST	SPI1 reset
RCU_USART1RST	USART1 reset
RCU_LPUARTRST	LPUART reset (only for GD32L233xx series)
RCU_LPUART0RST	LPUART0 reset (only for GD32L235xx series)
RCU_LPUART1RST	LPUART1 reset (only for GD32L235xx series)
RCU_CANRST	CAN reset (only for GD32L235xx series)
RCU_UART3RST	UART3 reset
RCU_UART4RST	UART4 reset
RCU_I2C0RST	I2C0 reset
RCU_I2C1RST	I2C1 reset
RCU_USBD RST	USB D reset
RCU_I2C2RST	I2C2 reset
RCU_PMURST	PMU reset
RCU_DACRST	DAC reset
RCU_CTCRST	CTC reset

## Enum rcu\_flag\_enum

**Table 3-554. Enum rcu\_flag\_enum**

enum name	Function description
RCU_FLAG_IRC32KST B	IRC32K stabilization flags
RCU_FLAG_LXTALST B	LXTAL stabilization flags
RCU_FLAG_IRC16MS TB	IRC16M stabilization flags
RCU_FLAG_HXTALST B	HXTAL stabilization flags
RCU_FLAG_IRC48MS TB	IRC48M stabilization flags
RCU_FLAG_PLLSTB	PLL stabilization flags
RCU_FLAG_V11RST	V11 reset flags
RCU_FLAG_EPRST	EPR reset flags
RCU_FLAG_PORRST	power reset flags
RCU_FLAG_SWRST	SW reset flags
RCU_FLAG_FWDGTR ST	FWDGT reset flags
RCU_FLAG_WWDGTR ST	WWDGT reset flags
RCU_FLAG_LPRST	LP reset flags

## Enum rcu\_int\_flag\_enum

Table 3-555. Enum rcu\_int\_flag\_enum

enum name	Function description
RCU_INT_FLAG_IRC3 2KSTB	IRC32K stabilization interrupt flag
RCU_INT_FLAG_LXTA LSTB	LXTAL stabilization interrupt flag
RCU_INT_FLAG_IRC1 6MSTB	IRC16M stabilization interrupt flag
RCU_INT_FLAG_HXTA LSTB	HXTAL stabilization interrupt flag
RCU_INT_FLAG_PLLS TB	PLL 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_CKM	CKM interrupt flag
RCU_INT_FLAG_IRC3 2KSTB	IRC32K stabilization interrupt flag

## Enum rcu\_int\_flag\_clear\_enum

Table 3-556. Enum rcu\_int\_flag\_clear\_enum

enum name	Function description
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_IRC1 6MSTB_CLR	IRC16M stabilization interrupt flags clear
RCU_INT_FLAG_HXTA LSTB_CLR	HXTAL stabilization interrupt flags clear
RCU_INT_FLAG_PLLS TB_CLR	PLL stabilization interrupt flags clear
RCU_INT_FLAG_IRC4 8MSTB_CLR	IRC48M stabilization interrupt flags clear
RCU_INT_FLAG_LXTA LCKM_CLR	LXTAL clock stuck interrupt flag clear
RCU_INT_FLAG_CKM _CLR	CKM interrupt flags clear
RCU_INT_FLAG_IRC3	IRC32K stabilization interrupt flags clear

enum name	Function description
2KSTB_CLR	

### Enum rcu\_int\_enum

Table 3-557. Enum rcu\_int\_enum

enum name	Function description
RCU_INT_IRC32KSTB	IRC32K stabilization interrupt
RCU_INT_LXTALSTB	LXTAL stabilization interrupt
RCU_INT_IRC16MSTB	IRC16M stabilization interrupt
RCU_INT_HXTALSTB	HXTAL stabilization interrupt
RCU_INT_PLLSTB	PLL stabilization interrupt
RCU_INT_IRC48MSTB	IRC48M stabilization interrupt
RCU_INT_LCKM	LXTAL clock stuck interrupt

### Enum rcu\_osci\_type\_enum

Table 3-558. Enum rcu\_osci\_type\_enum

enum name	Function description
RCU_HXTAL	HXTAL
RCU_LXTAL	LXTAL
RCU_IRC16M	IRC16M
RCU_IRC48M	IRC48M
RCU_IRC32K	IRC32K
RCU_PLL_CK	PLL

### Enum rcu\_clock\_freq\_enum

Table 3-559. Enum rcu\_clock\_freq\_enum

enum name	Function description
CK_SYS	system clock
CK_AHB	AHB clock
CK_APB1	APB1 clock
CK_APB2	APB2 clock
CK_ADC	ADC clock
CK_USART0	USART0 clock
CK_I2C0	I2C0 clock
CK_I2C1	I2C1 clock
CK_I2C2	I2C2 clock
CK_LPUART	LPUART clock (only for GD32L233xx series)
CK_LPUART0	LPUART0 clock (only for GD32L235xx series)
CK_LPUART1	LPUART1 clock (only for GD32L235xx series)
CK_USART1	USART1 clock
CK_LPTIMER	LPTIMER clock (only for GD32L233xx series)

enum name	Function description
CK_LPTIMER0	LPTIMER clock (only for GD32L235xx series)
CK_LPTIMER1	LPTIMER clock (only for GD32L235xx series)

### Enum usart\_idx\_enum

Table 3-560. Enum usart\_idx\_enum

enum name	Function description
IDX_USART0	idnex of USART0
IDX_USART1	idnex of USART1

### Enum lptimer\_idx\_enum (only for GD32L235xx series)

Table 3-561. Enum lptimer\_idx\_enum

enum name	Function description
IDX_LPTIMER0	idnex of LPTIMER0
IDX_LPTIMER1	idnex of LPTIMER1

### Enum lpuart\_idx\_enum (only for GD32L235xx series)

Table 3-562. Enum lpuart\_idx\_enum

enum name	Function description
IDX_LPUART0	idnex of LPUART0
IDX_LPUART1	idnex of LPUART1

### Enum i2c\_idx\_enum

Table 3-563. Enum i2c\_idx\_enum

enum name	Function description
IDX_I2C0	idnex of I2C0
IDX_I2C1	idnex of I2C1
IDX_I2C2	idnex of I2C2

### rcu\_deinit

The description of rcu\_deinit is shown as below:

Table 3-564. Function rcu\_deinit

Function name	rcu_deinit
Function prototype	void rcu_deinit(void);
Function descriptions	deinitialize the RCU, reset the value of all RCU registers into initial values
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset RCU */
rcu_deinit();
```

### rcu\_periph\_clock\_enable

The description of rcu\_periph\_clock\_enable is shown as below:

**Table 3-565. 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 3-551. Enum rcu_periph_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the USART0 clock */
rcu_periph_clock_enable(RCU_USART0);
```

### rcu\_periph\_clock\_disable

The description of rcu\_periph\_clock\_disable is shown as below:

**Table 3-566. 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 3-551. Enum rcu_periph_enum</a>
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable the USART0 clock */
```

```
rcu_periph_clock_disable(RCU_USART0);
```

### rcu\_periph\_clock\_sleep\_enable

The description of rcu\_periph\_clock\_sleep\_enable is shown as below:

**Table 3-567. Function rcu\_periph\_clock\_sleep\_enable**

<b>Function name</b>	rcu_periph_clock_sleep_enable
<b>Function prototype</b>	void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);
<b>Function descriptions</b>	enable the peripherals clock when in sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-552. Enum rcu_periph_sleep_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the FMC clock when in sleep mode */
```

```
rcu_periph_clock_sleep_enable(RCU_FMC_SLP);
```

### rcu\_periph\_clock\_sleep\_disable

The description of rcu\_periph\_clock\_sleep\_disable is shown as below:

**Table 3-568. Function rcu\_periph\_clock\_sleep\_disable**

<b>Function name</b>	rcu_periph_clock_sleep_disable
<b>Function prototype</b>	void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph);
<b>Function descriptions</b>	disable the peripherals clock when in sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-552. Enum rcu_periph_sleep_enum</a>
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* disable the FMC clock when in sleep mode */
```

```
rcu_periph_clock_sleep_disable(RCU_FMC_SLP);
```

### rcu\_periph\_reset\_enable

The description of rcu\_periph\_reset\_enable is shown as below:

**Table 3-569. Function rcu\_periph\_reset\_enable**

Function name	rcu_periph_reset_enable
Function prototype	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);
Function descriptions	enable the peripherals reset
Precondition	-
The called functions	-
Input parameter{in}	
periph_reset	RCU peripherals reset, refer to <a href="#">Table 3-553. Enum rcu_periph_reset_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 reset */
```

```
rcu_periph_reset_enable(RCU_SPI0RST);
```

### rcu\_periph\_reset\_disable

The description of rcu\_periph\_reset\_disable is shown as below:

**Table 3-570. 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 the peripheral reset
Precondition	-
The called functions	-
Input parameter{in}	
periph_reset	RCU peripherals reset, refer to <a href="#">Table 3-553. Enum rcu_periph_reset_enum</a>
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable SPI0 reset */
```

```
rcu_periph_reset_disable(RCU_SPI0RST);
```

### rcu\_bkp\_reset\_enable

The description of rcu\_bkp\_reset\_enable is shown as below:

**Table 3-571. Function rcu\_bkp\_reset\_enable**

Function name	rcu_bkp_reset_enable
Function prototype	void rcu_bkp_reset_enable(void);
Function descriptions	enable the BKP domain reset
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the BKP domain */
```

```
rcu_bkp_reset_enable();
```

### rcu\_bkp\_reset\_disable

The description of rcu\_bkp\_reset\_disable is shown as below:

**Table 3-572. Function rcu\_bkp\_reset\_disable**

Function name	rcu_bkp_reset_disable
Function prototype	void rcu_bkp_reset_disable(void);
Function descriptions	disable the BKP domain reset
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-



Return value	
-	-

Example:

```
/* disable the BKP domain reset */
```

```
rcu_bkp_reset_disable();
```

### rcu\_system\_clock\_source\_config

The description of rcu\_system\_clock\_source\_config is shown as below:

**Table 3-573. 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}	
ck_sys	system clock source select
RCU_CKSYSSRC_IRC16M	select CK_IRC16M as the CK_SYS source
RCU_CKSYSSRC_HXTAL	select CK_HXTAL as the CK_SYS source
RCU_CKSYSSRC_PLL	select CK_PLL as the CK_SYS source
RCU_CKSYSSRC_IRC48M	select CK_IRC48M as the CK_SYS source
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the CK_HXTAL as the CK_SYS source */
```

```
rcu_system_clock_source_config(RCU_CKSYSSRC_HXTAL);
```

### rcu\_system\_clock\_source\_get

The description of rcu\_system\_clock\_source\_get is shown as below:

**Table 3-574. Function rcu\_system\_clock\_source\_get**

Function name	rcu_system_clock_source_get
Function prototype	uint32_t rcu_system_clock_source_get(void);
Function descriptions	get the system clock source
Precondition	-

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	RCU_SCSS_IRC16M/RCU_SCSS_HXTAL/RCU_SCSS_PLL/RCU_SCSS_IRC48M

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 3-575. Function rcu\_ahb\_clock\_config**

Function name	rcu_ahb_clock_config
Function prototype	void rcu_ahb_clock_config(uint32_t ck_ahb);
Function descriptions	configure the AHB clock prescaler selection
Precondition	-
The called functions	-
Input parameter{in}	
ck_ahb	AHB clock prescaler selection
RCU_AHB_CKSYS_DIVx	select CK_SYS / x, (x=1, 2, 4, 8, 16, 64, 128, 256, 512)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure CK_SYS/128 */

rcu_ahb_clock_config(RCU_AHB_CKSYS_DIV128);
```

### rcu\_apb1\_clock\_config

The description of rcu\_apb1\_clock\_config is shown as below:

**Table 3-576. Function rcu\_apb1\_clock\_config**

Function name	rcu_apb1_clock_config
---------------	-----------------------

<b>Function prototype</b>	void rcu_apb1_clock_config(uint32_t ck_apb1);
<b>Function descriptions</b>	configure the APB1 clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_apb1</b>	APB1 clock prescaler selection
<i>RCU_APB1_CKAHB_D</i> <i>IVx</i>	select (CK_AHB / x) as CK_APB1 (x=1,2,4,8,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure CK_AHB/16 as CK_APB1 */
```

```
rcu_apb1_clock_config(RCU_APB1_CKAHB_DIV16);
```

### rcu\_apb2\_clock\_config

The description of rcu\_apb2\_clock\_config is shown as below:

**Table 3-577. Function rcu\_apb2\_clock\_config**

<b>Function name</b>	rcu_apb2_clock_config
<b>Function prototype</b>	void rcu_apb2_clock_config(uint32_t ck_apb2);
<b>Function descriptions</b>	configure the APB2 clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_apb2</b>	APB2 clock prescaler selection
<i>RCU_APB2_CKAHB_D</i> <i>IVx</i>	select (CK_AHB / x) as CK_APB2 clock (x=1,2,4,8,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure CK_AHB/8 as CK_APB2 */
```

```
rcu_apb2_clock_config(RCU_APB2_CKAHB_DIV8);
```

### rcu\_adc\_clock\_config

The description of rcu\_adc\_clock\_config is shown as below:

Table 3-578. Function rcu\_adc\_clock\_config

<b>Function name</b>	rcu_adc_clock_config
<b>Function prototype</b>	void rcu_adc_clock_config(rcu_adc_clock_enum ck_adc);
<b>Function descriptions</b>	configure the ADC clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_adc</b>	ADC clock prescaler selection
<i>RCU_ADCCCK_IRC16M</i>	select CK_IRC16M as CK_ADC
<i>RCU_ADCCCK_APB2_D</i> <i>IV2</i>	select CK_APB2/2 as CK_ADC
<i>RCU_ADCCCK_APB2_D</i> <i>IV4</i>	select CK_APB2/4 as CK_ADC
<i>RCU_ADCCCK_APB2_D</i> <i>IV6</i>	select CK_APB2/6 as CK_ADC
<i>RCU_ADCCCK_APB2_D</i> <i>IV8</i>	select CK_APB2/8 as CK_ADC
<i>RCU_ADCCCK_APB2_D</i> <i>IV10</i>	select CK_APB2/10 as CK_ADC
<i>RCU_ADCCCK_APB2_D</i> <i>IV12</i>	select CK_APB2/12 as CK_ADC
<i>RCU_ADCCCK_APB2_D</i> <i>IV14</i>	select CK_APB2/14 as CK_ADC
<i>RCU_ADCCCK_APB2_D</i> <i>IV16</i>	select CK_APB2/16 as CK_ADC
<i>RCU_ADCCCK_AHB_DI</i> <i>V3</i>	select CK_AHB/3 as CK_ADC
<i>RCU_ADCCCK_AHB_DI</i> <i>V5</i>	select CK_AHB/5 as CK_ADC
<i>RCU_ADCCCK_AHB_DI</i> <i>V7</i>	select CK_AHB/7 as CK_ADC
<i>RCU_ADCCCK_AHB_DI</i> <i>V9</i>	select CK_AHB/9 as CK_ADC
<i>RCU_ADCCCK_AHB_DI</i> <i>V11</i>	select CK_AHB/11 as CK_ADC
<i>RCU_ADCCCK_AHB_DI</i> <i>V13</i>	select CK_AHB/13 as CK_ADC
<i>RCU_ADCCCK_AHB_DI</i> <i>V15</i>	select CK_AHB/15 as CK_ADC
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the ADC prescaler factor */
rcu_adc_clock_config(RCU_ADCCCK_APB2_DIV2);
```

### rcu\_ckout\_config

The description of rcu\_ckout\_config is shown as below:

**Table 3-579. Function rcu\_ckout\_config**

Function name	rcu_ckout_config
Function prototype	void rcu_ckout_config(uint32_t ckout_src, uint32_t ckout_div);
Function descriptions	configure the CK_OUT clock source and divoision factor
Precondition	-
The called functions	-
Input parameter{in}	
ckout_src	CK_OUT clock source selection
RCU_CKOUTSRC_NO NE	no clock selected
RCU_CKOUTSRC_IRC 48M	select high speed 48M internal oscillator clock
RCU_CKOUTSRC_IRC 32K	select high speed 32K internal oscillator clock
RCU_CKOUTSRC_LX TAL	select LXTAL clock
RCU_CKOUTSRC_CK SYS	select system clock CK_SYS
RCU_CKOUTSRC_IRC 16M	select high speed 16M internal oscillator clock
RCU_CKOUTSRC_HX TAL	select HXTAL clock
RCU_CKOUTSRC_CK PLL_DIV1	select CK_PLL clock
RCU_CKOUTSRC_CK PLL_DIV2	Select (CK_PLL / 2) clock
Input parameter{in}	
ckout_div	CK_OUT divider
RCU_CKOUT_DIVx	CK_OUT is divided by x(x=1,2,4,8,16,32,64,128)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the HXTAL as CK_OUT clock source */
```

```
rcu_ckout_config(RCU_CKOUTSRC_HXTAL, RCU_CKOUT_DIV1);
```

### rcu\_pll\_config

The description of rcu\_pll\_config is shown as below:

**Table 3-580. Function rcu\_pll\_config**

<b>Function name</b>	rcu_pll_config
<b>Function prototype</b>	void rcu_pll_config(uint32_t pll_src, uint32_t pll_mul);
<b>Function descriptions</b>	configure the main PLL clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pll_src</b>	PLL clock source selection
<i>RCU_PLLSRC_IRC16M</i>	IRC16M clock is selected as source clock of PLL
<i>RCU_PLLSRC_HXTAL</i>	HXTAL is selected as source clock of PLL
<i>RCU_PLLSRC_IRC48M</i>	select CK_IRC48M as PLL source clock
<b>Input parameter{in}</b>	
<b>pll_mul</b>	PLL clock multiplication factor
<i>RCU_PLL_MULx</i>	PLL source clock * x (x = 4..127)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL */
```

```
rcu_pll_config(RCU_PLLSRC_HXTAL, RCU_PLL_MUL10);
```

### rcu\_usart\_clock\_config

The description of rcu\_usart\_clock\_config is shown as below:

**Table 3-581. 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 USARTx(x=0,1) clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>usart_idx</b>	idnex of USART, refer to <a href="#">Table 3-560. Enum usart_idx_enum</a>
<i>IDX_USARTx</i>	idnex of USARTx(x=0,1)
<b>ck_usart</b>	USART clock source selection
<i>RCU_USARTSRC_CKAPB</i>	CK_USART select CK_APB1/ CK_APB2
<i>RCU_USARTSRC_CKSYS</i>	CK_USART select CK_SYS
<i>RCU_USARTSRC_LXTAL</i>	CK_USART select CK_LXTAL
<i>RCU_USARTSRC_IRC16MDIV</i>	CK_USART select CK_IRC16MDIV
<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 3-582. 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=0,1,2) clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_idx</b>	idnex of I2C, refer to <a href="#">Table 3-563. Enum i2c_idx_enum</a>
<i>IDX_I2Cx</i>	idnex of I2Cx(x=0,1,2)
<b>ck_i2c</b>	I2C clock source selection
<i>RCU_I2CSRC_CKAPB1</i>	CK_I2C select CK_APB1
<i>RCU_I2CSRC_CKSYS</i>	CK_I2C select CK_SYS
<i>RCU_I2CSRC_IRC16MDIV</i>	CK_I2C select IRC16MDIV
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the CKAPB1 as I2C0 clock */
```

```
rcu_usart_clock_config(IDX_I2C0, RCU_I2CSRC_CKAPB1);
```

### rcu\_lptimer\_clock\_config (only for GD32L233xx series)

The description of rcu\_lptimer\_clock\_config is shown as below:

**Table 3-583. Function rcu\_lptimer\_clock\_config**

Function name	rcu_lptimer_clock_config
Function prototype	void rcu_lptimer_clock_config(uint32_t ck_lptimer);
Function descriptions	configure the LPTIMER clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
ck_lptimer	LPTIMER clock source selection
RCU_LPTIMERSRC_CKAPB2	CK_LPTIMER select CK_APB2
RCU_LPTIMERSRC_IRC32K	CK_LPTIMER select CK_IRC32K
RCU_LPTIMERSRC_LXTAL	CK_LPTIMER select CK_LXTAL
RCU_LPTIMERSRC_IRC16MDIV	CK_LPTIMER select CK_IRC16MDIV
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the LPTIMER clock source selection */
```

```
rcu_lptimer_clock_config(RCU_LPTIMERSRC_CKAPB2);
```

### rcu\_lptimer\_clock\_config (only for GD32L235xx series)

The description of rcu\_lptimer\_clock\_config is shown as below:

**Table 3-584. Function rcu\_lptimer\_clock\_config**

Function name	rcu_lptimer_clock_config
Function prototype	void rcu_lptimer_clock_config(lptimer_idx_enum lptimer_idx, uint32_t ck_lptimer);
Function descriptions	configure the LPTIMER clock source selection
Precondition	-



<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lptimer_idx</b>	idnex of LPTIMER, refer to <a href="#">Table 3-561. Enum lptimer_idx_enum</a>
<i>IDX_LPTIMERx</i>	idnex of LPTIMERx(x=0,1)
<b>ck_lptimer</b>	LPTIMER clock source selection
<i>RCU_LPTIMERSRC_CKAPB1</i>	CK_LPTIMER select CK_APB1
<i>RCU_LPTIMERSRC_IRC32K</i>	CK_LPTIMER select CK_IRC32K
<i>RCU_LPTIMERSRC_LXTAL</i>	CK_LPTIMER select CK_LXTAL
<i>RCU_LPTIMERSRC_IRC16MDIV</i>	CK_LPTIMER select CK_IRC16MDIV
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the LPTIMER clock source selection */
```

```
rcu_lptimer_clock_config(IDX_LPTIMER0, RCU_LPTIMERSRC_CKAPB1);
```

### rcu\_lpuart\_clock\_config (only for GD32L233xx series)

The description of rcu\_lpuart\_clock\_config is shown as below:

**Table 3-585. Function rcu\_lpuart\_clock\_config**

<b>Function name</b>	rcu_lpuart_clock_config
<b>Function prototype</b>	void rcu_lpuart_clock_config(uint32_t ck_lpuart);
<b>Function descriptions</b>	configure the LPUART clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_lpuart</b>	LPUART clock source selection
<i>RCU_LPUARTSRC_CKAPB1</i>	LPUART select CK_APB1
<i>RCU_LPUARTSRC_CKSYS</i>	LPUART select CK_SYS
<i>RCU_LPUARTSRC_LXTAL</i>	LPUART select CK_LXTAL
<i>RCU_LPUARTSRC_IRC16MDIV</i>	LPUART select CK_IRC16MDIV
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* configure the lpuart clock source selection */
```

```
rcu_lpuart_clock_config(RCU_LPUARTSRC_CKAPB1);
```

### rcu\_lpuart\_clock\_config (only for GD32L235xx series)

The description of rcu\_lpuart\_clock\_config is shown as below:

**Table 3-586. Function rcu\_lpuart\_clock\_config**

<b>Function name</b>	rcu_lpuart_clock_config
<b>Function prototype</b>	void rcu_lpuart_clock_config(lpuart_idx_enum lpuart_idx, uint32_t ck_lpuart);
<b>Function descriptions</b>	configure the LPUART clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lpuart_idx</b>	idnex of LPUART, refer to <a href="#">Table 3-562. Enum lpuart_idx_enum</a>
<i>IDX_LPUARTx</i>	idnex of LPUARTx(x=0,1)
<b>ck_lpuart</b>	LPUART clock source selection
<i>RCU_LPUARTSRC_CKAPB1</i>	LPUART select CK_APB1
<i>RCU_LPUARTSRC_CKSYS</i>	LPUART select CK_SYS
<i>RCU_LPUARTSRC_CKLXTAL</i>	LPUART select CK_LXTAL
<i>RCU_LPUARTSRC_CKIRC16MDIV</i>	LPUART select CK_IRC16MDIV
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the lpuart clock source selection */
```

```
rcu_lpuart_clock_config(IDX_LPUART0, RCU_LPUARTSRC_CKAPB1);
```

### rcu\_irc16mdiv\_clock\_config

The description of rcu\_irc16mdiv\_clock\_config is shown as below:

Table 3-587. Function rcu\_irc16mdiv\_clock\_config

<b>Function name</b>	rcu_irc16mdiv_clock_config
<b>Function prototype</b>	void rcu_irc16mdiv_clock_config(uint32_t ck_irc16mdiv);
<b>Function descriptions</b>	configure the IRC16MDIV clock selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_irc16mdiv</b>	IRC16MDIV clock selection
<i>RCU_IRC16MDIV_NO NE</i>	CK_IRC16MDIV select CK_IRC16M
<i>RCU_IRC16MDIV_2</i>	CK_IRC16MDIV select CK_IRC16M divided by 2
<i>RCU_IRC16MDIV_4</i>	CK_IRC16MDIV select CK_IRC16M divided by 4
<i>RCU_IRC16MDIV_8</i>	CK_IRC16MDIV select CK_IRC16M divided by 8
<i>RCU_IRC16MDIV_16</i>	CK_IRC16MDIV select CK_IRC16M divided by 16
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the IRC16MDIV clock selection */
rcu_irc16mdiv_clock_config(RCU_IRC16MDIV_2);
```

### rcu\_usbd\_clock\_config

The description of rcu\_usbd\_clock\_config is shown as below:

Table 3-588. Function rcu\_usbd\_clock\_config

<b>Function name</b>	rcu_usbd_clock_config
<b>Function prototype</b>	void rcu_usbd_clock_config(uint32_t ck_usbd);
<b>Function descriptions</b>	configure the USB D clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_usbd</b>	USB D clock source selection
<i>RCU_USBDSRC_IRC4 8M</i>	select CK_IRC48M as USB D source clock
<i>RCU_USBDSRC_PLL</i>	select CK_PLL as USB D source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the USB clock source selection */
```

```
rcu_usbd_clock_config(RCU_USBDSRC_IRC48M);
```

### rcu\_rtc\_clock\_config

The description of rcu\_rtc\_clock\_config is shown as below:

**Table 3-589. 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/SLCD clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_clock_source</b>	RTC/SLCD clock source selection
<i>RCU_RTCSRC_NONE</i>	no clock selected
<i>RCU_RTCSRC_LXTAL</i>	select CK_LXTAL as RTC/SLCD source clock
<i>RCU_RTCSRC_IRC40K</i>	select CK_IRC40K as RTC/SLCD source clock
<i>RCU_RTCSRC_HXTAL_DIV_32</i>	select (CK_HXTAL / 32) as RTC/SLCD 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\_pll\_source\_ck\_prediv\_config

The description of rcu\_pll\_source\_ck\_prediv\_config is shown as below:

**Table 3-590. Function rcu\_pll\_source\_ck\_prediv\_config**

<b>Function name</b>	rcu_pll_source_ck_prediv_config
<b>Function prototype</b>	void rcu_pll_source_ck_prediv_config(uint32_t pllsource_ck_prediv);
<b>Function descriptions</b>	configure PLL source clocks pre-divider
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pllsource_ck_prediv</b>	PLL source clocks divider used as input of PLL
<i>RCU_PLL_PREDVx</i>	PLL source clocks divided x used as input of PLL (x=1..16)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure PLL source clocks pre-divider */
```

```
rcu_pll_source_ck_prediv_config(RCU_PLL_PREDV2);
```

### rcu\_lxtal\_drive\_capability\_config

The description of rcu\_lxtal\_drive\_capability\_config is shown as below:

**Table 3-591. Function rcu\_lxtal\_drive\_capability\_config**

Function name	rcu_lxtal_drive_capability_config
Function prototype	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);
Function descriptions	configure the LXTAL drive capability
Precondition	-
The called functions	-
Input parameter{in}	
lxtal_dricap	drive capability of LXTAL
RCU_LXTAL_LOWDRI	lower driving capability
RCU_LXTAL_MED_LOWDRI	medium low driving capability
RCU_LXTAL_MED_HIGHDRI	medium high driving capability
RCU_LXTAL_HIGHDRI	higher driving capability
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the LXTAL lower driving capability */
```

```
rcu_lxtal_drive_capability_config(RCU_LXTAL_LOWDRI);
```

### rcu\_lp\_bandgap\_config

The description of rcu\_lp\_bandgap\_config is shown as below:

**Table 3-592. Function rcu\_lp\_bandgap\_config**

Function name	rcu_lp_bandgap_config
Function prototype	void rcu_lp_bandgap_config(uint32_t lp_bandgap_clock);

<b>Function descriptions</b>	configure low power bandgap mode selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lp_bandgap_clock</b>	low power bandgap clock
<i>RCU_LPBM_32CLK</i>	The length of holding phase is 3.2ms, 32 clock cycles
<i>RCU_LPBM_64CLK</i>	The length of holding phase is 6.4ms, 64 clock cycles
<i>RCU_LPBM_128CLK</i>	The length of holding phase is 12.8ms, 128 clock cycles
<i>RCU_LPBM_256CLK</i>	The length of holding phase is 25.6ms, 256 clock cycles
<i>RCU_LPBM_512CLK</i>	The length of holding phase is 51.2ms, 512 clock cycles
<i>RCU_LPBM_1024CLK</i>	The length of holding phase is 102.4ms, 1024 clock cycles
<i>RCU_LPBM_2048CLK</i>	The length of holding phase is 204.8ms, 2048 clock cycles
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure low power bandgap mode selection */
rcu_lp_bandgap_config(RCU_LPBM_64CLK);
```

## rcu\_osci\_stab\_wait

The description of rcu\_osci\_stab\_wait is shown as below:

**Table 3-593. Function rcu\_osci\_stab\_wait**

<b>Function name</b>	rcu_osci_stab_wait
<b>Function prototype</b>	ErrStatus rcu_osci_stab_wait(rcu_osci_type_enum osci);
<b>Function descriptions</b>	wait for oscillator stabilization flags is SET or oscillator startup is timeout
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-558. Enum rcu_osci_type_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* wait for oscillator stabilization flag */
if(SUCCESS == rcu_osci_stab_wait(RCU_HXTAL)){
}
```

## rcu\_osc\_i\_on

The description of rcu\_osc\_i\_on is shown as below:

**Table 3-594. Function rcu\_osc\_i\_on**

<b>Function name</b>	rcu_osc_i_on
<b>Function prototype</b>	void rcu_osc_i_on(rcu_osc_i_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 3-558. Enum rcu_osc_i_type_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn on the high speed crystal oscillator */
```

```
rcu_osc_i_on(RCU_HXTAL);
```

## rcu\_osc\_i\_off

The description of rcu\_osc\_i\_off is shown as below:

**Table 3-595. Function rcu\_osc\_i\_off**

<b>Function name</b>	rcu_osc_i_off
<b>Function prototype</b>	void rcu_osc_i_off(rcu_osc_i_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 3-558. Enum rcu_osc_i_type_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn off the high speed crystal oscillator */
```

```
rcu_osc_i_off(RCU_HXTAL);
```

## rcu\_osci\_bypass\_mode\_enable

The description of rcu\_osci\_bypass\_mode\_enable is shown as below:

**Table 3-596. Function rcu\_osci\_bypass\_mode\_enable**

<b>Function name</b>	rcu_osci_bypass_mode_enable
<b>Function prototype</b>	void rcu_osci_bypass_mode_enable(rcu_osci_type_enum osci);
<b>Function descriptions</b>	enable the oscillator bypass mode
<b>Precondition</b>	HXTALEN or LXTALEN must be reset before it
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-558. Enum rcu_osci_type_enum</a>
RCU_HXTAL	high speed crystal oscillator(HXTAL)
RCU_LXTAL	low speed crystal oscillator(LXTAL)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_enable(RCU_HXTAL);
```

## rcu\_osci\_bypass\_mode\_disable

The description of rcu\_osci\_bypass\_mode\_disable is shown as below:

**Table 3-597. Function rcu\_osci\_bypass\_mode\_disable**

<b>Function name</b>	rcu_osci_bypass_mode_disable
<b>Function prototype</b>	void rcu_osci_bypass_mode_disable(rcu_osci_type_enum osci);
<b>Function descriptions</b>	disable the oscillator bypass mode
<b>Precondition</b>	HXTALEN or LXTALEN must be reset before it
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-558. Enum rcu_osci_type_enum</a>
RCU_HXTAL	high speed crystal oscillator(HXTAL)
RCU_LXTAL	low speed crystal oscillator(LXTAL)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the high speed crystal oscillator bypass mode */
```



```
rcu_osc_bypass_mode_disable(RCU_HXTAL);
```

## rcu\_irc16m\_adjust\_value\_set

The description of rcu\_irc16m\_adjust\_value\_set is shown as below:

**Table 3-598. Function rcu\_irc16m\_adjust\_value\_set**

<b>Function name</b>	rcu_irc16m_adjust_value_set
<b>Function prototype</b>	void rcu_irc16m_adjust_value_set(uint32_t irc16m_adjval);
<b>Function descriptions</b>	set the IRC16M adjust value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>irc16m_adjval</b>	IRC8M adjust value, must be between 0 and 0x1F
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the IRC16M adjust value */
rcu_irc16m_adjust_value_set(0x10);
```

## rcu\_hxtal\_clock\_monitor\_enable

The description of rcu\_hxtal\_clock\_monitor\_enable is shown as below:

**Table 3-599. 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 3-600. 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 3-601. 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 3-602. 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\_voltage\_key\_unlock

The description of rcu\_voltage\_key\_unlock is shown as below:

**Table 3-603. Function rcu\_voltage\_key\_unlock**

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

Example:

```
/* unlock the voltage key */
rcu_voltage_key_unlock();
```

## rcu\_clock\_freq\_get

The description of rcu\_clock\_freq\_get is shown as below:

**Table 3-604. Function rcu\_clock\_freq\_get**

<b>Function name</b>	rcu_clock_freq_get
<b>Function prototype</b>	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);
<b>Function descriptions</b>	get the system clock, bus clock and peripheral clock frequency
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>clock</b>	the clock frequency which to get, refer to <a href="#">Table 3-559. Enum rcu_clock_freq_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	clock frequency of system, AHB, APB1, APB2, ADC or USRAT0/1, LPUART, LPTIMER, LPTIMERx (x=0,1) (GD32L235xx series), CK_LPUARTx (x=0,1) (GD32L235xx series)

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 3-605. 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 3-554. Enum rcu_flag_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the clock stabilization flag */
```

```
if(RESET != rcu_flag_get(RCU_FLAG_LXTALSTB)){  
}  
}
```

### rcu\_all\_reset\_flag\_clear

The description of rcu\_all\_reset\_flag\_clear is shown as below:

**Table 3-606. Function rcu\_all\_reset\_flag\_clear**

<b>Function name</b>	rcu_all_reset_flag_clear
<b>Function prototype</b>	void rcu_all_reset_flag_clear(void);
<b>Function descriptions</b>	clear all the reset flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear all the reset flag */
```

```
rcu_all_reset_flag_clear();
```

### rcu\_interrupt\_flag\_get

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

**Table 3-607. Function rcu\_interrupt\_flag\_get**

<b>Function name</b>	rcu_interrupt_flag_get
<b>Function prototype</b>	FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag);
<b>Function descriptions</b>	get the clock stabilization interrupt and ckm flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt and ckm flags, refer to <a href="#">Table 3-555. Enum rcu_int_flag_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```

/* get the clock stabilization interrupt flag */

if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){

}

```

### rcu\_interrupt\_flag\_clear

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

**Table 3-608. Function rcu\_interrupt\_flag\_clear**

<b>Function name</b>	rcu_interrupt_flag_clear
<b>Function prototype</b>	void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag_clear)
<b>Function descriptions</b>	clear the interrupt flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag_clear</b>	clock stabilization and stuck interrupt flags clear, refer to <a href="#">Table 3-556. Enum rcu_int_flag_clear_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* clear the interrupt HXTAL stabilization interrupt flag */

rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);

```

### rcu\_interrupt\_enable

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

**Table 3-609. 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 3-557. Enum rcu_int_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the HXTAL stabilization interrupt */
```

```
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

## rcu\_interrupt\_disable

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

**Table 3-610. 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>	-
<b>Input parameter{in}</b>	
<b>stab_int</b>	clock stabilization interrupt, refer to <a href="#">Table 3-557. Enum rcu_int_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the HXTAL stabilization interrupt */
```

```
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

## 3.21. 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 [3.21.1](#), the FWDGT firmware functions are introduced in chapter [3.21.2](#).

### 3.21.1. Descriptions of Peripheral registers

RTC registers are listed in the table shown as below:

**Table 3-611. 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_WUT	RTC wakeup timer register

Registers	Descriptions
RTC_ALRM0TD	RTC alarm 0 time and date register
RTC_ALRM1TD	RTC alarm 1 time and date register
RTC_WPK	RTC write protection key register
RTC_SS	RTC sub second register
RTC_SHIFTCTL	RTC shift function control register
RTC_TTS	RTC time of timestamp register
RTC_DTS	RTC date of timestamp register
RTC_SSTS	RTC sub second of timestamp register
RTC_HRFC	RTC high resolution frequency compensation register
RTC_TAMP	RTC tamper register
RTC_ALRM0SS	RTC alarm 0 sub second register
RTC_ALRM1SS	RTC alarm 1 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
RTC_BKP4	RTC backup 4 register

### 3.21.2. Descriptions of Peripheral functions

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

**Table 3-612. RTC firmware function**

Function name	Function description
rtc_deinit	reset most of the RTC registers
rtc_init	initialize RTC registers
rtc_init_mode_enter	enter RTC init mode
rtc_init_mode_exit	exit RTC init mode
rtc_register_sync_wait	wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated
rtc_current_time_get	get current time and date
rtc_subsecond_get	get current subsecond value
rtc_alarm_config	configure RTC alarm
rtc_alarm_subsecond_config	configure subsecond of RTC alarm
rtc_alarm_get	get RTC alarm
rtc_alarm_subsecond_get	get RTC alarm subsecond
rtc_alarm_enable	enable RTC alarm
rtc_alarm_disable	disable RTC alarm
rtc_timestamp_enable	enable RTC time-stamp
rtc_timestamp_disable	disable RTC time-stamp
rtc_timestamp_get	get RTC timestamp time and date



Function name	Function description
rtc_timestamp_internalevent_config	configure RTC time-stamp internal event
rtc_timestamp_subsecond_get	get RTC time-stamp subsecond
rtc_tamper_enable	enable RTC tamper
rtc_tamper_disable	disable RTC tamper
rtc_tamper_mask	set specified RTC tamper mask function
rtc_tamper_without_bkp_reset	tamperx event does not erase the RTC_BKP registers
rtc_output_pin_select	select the RTC output pin
rtc_alarm_output_config	configure RTC alarm output source
rtc_calibration_output_config	configure RTC calibration output source
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_bypass_shadow_enable	enable RTC bypass shadow registers function
rtc_bypass_shadow_disable	disable RTC bypass shadow registers function
rtc_refclock_detection_enable	enable RTC reference clock detection function
rtc_refclock_detection_disable	disable RTC reference clock detection function
rtc_wakeup_enable	enable RTC wakeup timer
rtc_wakeup_disable	disable RTC wakeup timer
rtc_wakeup_clock_set	set auto wakeup timer clock
rtc_wakeup_timer_set	set auto wakeup timer value
rtc_wakeup_timer_get	get auto wakeup timer value
rtc_smooth_calibration_config	configure RTC smooth calibration
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
rtc_lxtal_stab_reset_enable	enable LXTAL stabilization reset
rtc_lxtal_stab_reset_disable	disable LXTAL stabilization reset

## Structure rtc\_parameter\_struct

**Table 3-613. Structure 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 3-614. Structure 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 3-615. Structure 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

### Structure rtc\_tamper\_struct

**Table 3-616. Structure rtc\_tamper\_struct**

Member name	Function description
tamper_source	RTC tamper source
tamper_trigger	RTC tamper trigger
tamper_filter	RTC tamper consecutive samples needed during a voltage level detection
tamper_sample_frequency	RTC tamper sampling frequency during a voltage level detection
tamper_precharge_enable	RTC tamper precharge feature during a voltage level detection
tamper_precharge_time	RTC tamper precharge duration if precharge feature is enabled
tamper_with_timestamp	RTC tamper time-stamp feature

## rtc\_deinit

The description of rtc\_deinit is shown as below:

**Table 3-617. 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\_init

The description of rtc\_init is shown as below:

**Table 3-618. 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 3-613. Structure 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\_enter

The description of rtc\_init\_mode\_enter is shown as below:

**Table 3-619. 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\_mode\_exit

The description of rtc\_init\_mode\_exit is shown as below:

**Table 3-620. 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\_register\_sync\_wait

The description of rtc\_register\_sync\_wait is shown as below:

**Table 3-621. 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>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/*wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the
shadow registers are updated*/
ErrStatus error_status = rtc_register_sync_wait();
```

### rtc\_current\_time\_get

The description of rtc\_current\_time\_get is shown as below:

Table 3-622. 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 3-613. Structure 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 3-623. 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\_alarm\_config

The description of rtc\_alarm\_config is shown as below:

**Table 3-624. Function rtc\_alarm\_config**

<b>Function name</b>	rtc_alarm_config
<b>Function prototype</b>	void rtc_alarm_config(uint8_t rtc_alarm, rtc_alarm_struct *rtc_alarm_time);
<b>Function descriptions</b>	configure RTC alarm
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_alarm</b>	RTC_ALARM0 or RTC_ALARM1
<b>Input parameter{in}</b>	
<b>rtc_alarm_time</b>	pointer to a rtc_alarm_struct structure which contains parameters for RTC alarm configuration, the structure members can refer to members of the structure <a href="#">Table 3-614. Structure rtc_alarm_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*rtc_alarm_config*/
```

```
rtc_alarm_struct rtc_alarm_time;
```

```
rtc_alarm_config(RTC_ALARM0,&rtc_alarm_time);
```

## rtc\_alarm\_subsecond\_config

The description of rtc\_alarm\_subsecond\_config is shown as below:

**Table 3-625. Function rtc\_alarm\_subsecond\_config**

<b>Function name</b>	rtc_alarm_subsecond_config
<b>Function prototype</b>	void rtc_alarm_subsecond_config(uint8_t rtc_alarm, uint32_t mask_subsecond, uint32_t subsecond)
<b>Function descriptions</b>	configure subsecond of RTC alarm
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_alarm</b>	RTC_ALARM0 or RTC_ALARM1
<b>Input parameter{in}</b>	
<b>mask_subsecond</b>	alarm subsecond mask
<b>RTC_MASKSSC_0_14</b>	mask alarm subsecond configuration
<b>RTC_MASKSSC_1_14</b>	mask RTC_ALRM0SS_SSC[14:1], and RTC_ALRM0SS_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_ALARM0,RTC_MASKSSC_9_14, 0x7FFF);
```

### **rtc\_alarm\_enable**

The description of `rtc_alarm_enable` is shown as below:



Table 3-626. Function rtc\_alarm\_enable

Function name	rtc_alarm_enable
Function prototype	void rtc_alarm_enable(uint8_t rtc_alarm);
Function descriptions	enable RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	RTC_ALARM0 or RTC_ALARM1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*enable RTC alarm*/

rtc_alarm_enable(RTC_ALARM0);
```

### rtc\_alarm\_disable

The description of rtc\_alarm\_disable is shown as below:

Table 3-627. Function rtc\_alarm\_disable

Function name	rtc_alarm_disable
Function prototype	ErrStatus rtc_alarm_disable(uint8_t rtc_alarm);
Function descriptions	disable RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	RTC_ALARM0 or RTC_ALARM1
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/*disable RTC alarm*/

ErrStatus error_status = rtc_alarm_disable(RTC_ALARM0);
```

### rtc\_alarm\_get

The description of rtc\_alarm\_get is shown as below:

Table 3-628. Function rtc\_alarm\_get

Function name	rtc_alarm_get
Function prototype	void rtc_alarm_get(uint8_t rtc_alarm, rtc_alarm_struct *rtc_alarm_time);
Function descriptions	get RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	RTC_ALARM0 or RTC_ALARM1
Output parameter{out}	
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 3-614. Structure rtc_alarm_struct</a>
Return value	
-	-

Example:

```
/*disable RTC alarm*/
```

```
rtc_alarm_struct rtc_alarm_time;
```

```
rtc_alarm_get(RTC_ALARM0,&rtc_alarm_time);
```

### rtc\_alarm\_subsecond\_get

The description of rtc\_alarm\_subsecond\_get is shown as below:

Table 3-629. Function rtc\_alarm\_subsecond\_get

Function name	rtc_alarm_subsecond_get
Function prototype	uint32_t rtc_alarm_subsecond_get(uint8_t rtc_alarm);
Function descriptions	get RTC alarm subsecond
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	RTC_ALARM0 or RTC_ALARM1
Output parameter{out}	
-	-
Return value	
uint32_t	RTC alarm subsecond value(0x0-0x3FFF)

Example:

```
/*get RTC alarm subsecond*/
```

```
uint32_t subsecond = rtc_alarm_subsecond_get(RTC_ALARM0);
```

## rtc\_timestamp\_enable

The description of rtc\_timestamp\_enable is shown as below:

**Table 3-630. 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 3-631. 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\_internalevent\_config

The description of `rtc_timestamp_internalevent_config` is shown as below:

**Table 3-632. Function `rtc_timestamp_internalevent_config`**

<b>Function name</b>	<code>rtc_timestamp_internalevent_config</code>
<b>Function prototype</b>	<code>void rtc_timestamp_internalevent_config(uint32_t mode)</code>
<b>Function descriptions</b>	configure RTC time-stamp internal event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mode</b>	specify which internal or external event to be detected
<code>RTC_ITSEN_DISABLE</code>	disable RTC time-stamp internal event
<code>RTC_ITSEN_ENABLE</code>	enable RTC time-stamp internal event
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable RTC time-stamp internal event */
```

```
rtc_timestamp_internalevent_config(RTC_ITSEN_DISABLE);
```

## rtc\_timestamp\_get

The description of `rtc_timestamp_get` is shown as below:

**Table 3-633. Function `rtc_timestamp_get`**

<b>Function name</b>	<code>rtc_timestamp_get</code>
<b>Function prototype</b>	<code>void rtc_timestamp_get(rtc_timestamp_struct* rtc_timestamp);</code>
<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>	
<code>rtc_timestamp</code>	Pointer to a <code>rtc_timestamp_struct</code> structure which contains parameters for RTC time-stamp configuration, the structure members can refer to members of the structure <a href="#">Table 3-616. Structure <code>rtc_tamper_struct</code></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 3-634. 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\_tamper\_enable

The description of rtc\_tamper\_enable is shown as below:

**Table 3-635. Function rtc\_timestamp\_enable**

<b>Function name</b>	rtc_tamper_enable
<b>Function prototype</b>	void rtc_tamper_enable(rtc_tamper_struct* rtc_tamper);
<b>Function descriptions</b>	enable RTC tamper
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
rtc_tamper	pointer to a rtc_tamper_struct structure which contains parameters for RTC tamper configuration, the structure members can refer to members of the structure <a href="#">Table 3-616. Structure rtc_tamper_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* enable RTC tamper */

rtc_tamper_struct rtc_tamper

rtc_tamper_enable(& rtc_tamper);
```

### rtc\_tamper\_disable

The description of rtc\_tamper\_disable is shown as below:

**Table 3-636. Function rtc\_tamper\_disable**

<b>Function name</b>	rtc_tamper_disable
<b>Function prototype</b>	void rtc_tamper_disable(uint32_t source);
<b>Function descriptions</b>	disable RTC tamper
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	specify which tamper source to be disabled
<i>RTC_TAMPER0</i>	RTC tamper0
<i>RTC_TAMPER1</i>	RTC tamper1
<i>RTC_TAMPER2</i>	RTC tamper2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable RTC tamper0 */

rtc_tamper_disable(RTC_TAMPER0);
```

### rtc\_tamper\_mask

The description of rtc\_tamper\_mask is shown as below:

**Table 3-637. Function rtc\_tamper\_mask**

<b>Function name</b>	rtc_tamper_mask
<b>Function prototype</b>	void rtc_tamper_mask(uint32_t source);
<b>Function descriptions</b>	set specified RTC tamper mask
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	specify which tamper will be masked

<i>RTC_TAMPMASK_NO NE</i>	both tamper 0~2 would not be masked
<i>RTC_TAMPMASK_TP0</i>	tamper0 will be masked, both TPIE and TP0IE would be fouced to reset
<i>RTC_TAMPMASK_TP1</i>	Tamper1 will be masked, both TPIE and TP1IE would be fouced to reset
<i>RTC_TAMPMASK_TP2</i>	Tamper2 will be masked, both TPIE and TP2IE would be fouced to reset
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* both tamper 0~3 would not be masked */
```

```
rtc_tamper_mask(RTC_TAMPMASK_NONE);
```

### rtc\_tamper\_without\_bkp\_reset

The description of `rtc_tamper_without_bkp_reset` is shown as below:

**Table 3-638. Function `rtc_tamper_without_bkp_reset`**

<b>Function name</b>	<code>rtc_tamper_without_bkp_reset</code>
<b>Function prototype</b>	<code>void rtc_tamper_without_bkp_reset(uint32_t ne_source);</code>
<b>Function descriptions</b>	tamperx event does not erase the RTC_BKP registers
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ne_source</b>	specify which tamper source will not trigger the RTC_BKP registers reset
<i>RTC_TAMPXNOERAS E_NONE</i>	both tamper0 and tamper1 event will trigger RTC_BKP registers reset
<i>RTC_TAMPXNOERAS E_TP0</i>	tamper0 event will not trigger RTC_BKP registers reset
<i>RTC_TAMPXNOERAS E_TP1</i>	tamper1 event will not trigger RTC_BKP registers reset
<i>RTC_TAMPXNOERAS E_TP2</i>	tamper2 event will not trigger RTC_BKP registers reset
<i>RTC_TAMPXNOERAS E_TP0_TP1</i>	Neither tamper0 nor tamepr1 event will trigger RTC_BKP registers reset
<i>RTC_TAMPXNOERAS E_TP_ALL</i>	tamper 0~3 event all will not trigger RTC_BKP registers reset
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* tamper 0~3 event all will not trigger RTC_BKP registers reset */
```

```
rtc_tamper_without_bkp_reset(RTC_TAMPXNOERASE_TP_ALL);
```

## rtc\_output\_pin\_select

The description of rtc\_output\_pin\_select is shown as below:

**Table 3-639. Function rtc\_output\_pin\_select**

<b>Function name</b>	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\_alarm\_output\_config

The description of rtc\_alarm\_output\_config is shown as below:

**Table 3-640. Function rtc\_alarm\_output\_config**

<b>Function name</b>	rtc_alarm_output_config
<b>Function prototype</b>	void rtc_alarm_output_config(uint32_t source, uint32_t mode);
<b>Function descriptions</b>	configure rtc alarm output source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	specify signal to output
<i>RTC_ALARM0_HIGH</i>	when the alarm0 flag is set, the output pin is high
<i>RTC_ALARM0_LOW</i>	when the alarm0 flag is set, the output pin is low
<i>RTC_ALARM1_HIGH</i>	when the alarm1 flag is set, the output pin is high
<i>RTC_ALARM1_LOW</i>	when the alarm1 flag is set, the output pin is low
<i>RTC_WAKEUP_HIGH</i>	when the wakeup flag is set, the output pin is high



<i>RTC_WAKEUP_LOW</i>	when the wakeup flag is set, the output pin is low
<b>Input parameter{in}</b>	
<b>mode</b>	specify the output pin mode when output alarm signal or auto wakeup signal
<i>RTC_ALARM_OUTPU</i> <i>T_OD</i>	open drain mode
<i>RTC_ALARM_OUTPU</i> <i>T_PP</i>	push pull mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure rtc alarm0 output source */

rtc_alarm_output_config(RTC_ALARM0_LOW, RTC_ALARM_OUTPUT_PP);
```

### rtc\_calibration\_output\_config

The description of rtc\_calibration\_output\_config is shown as below:

**Table 3-641. Function 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>	
<b>source</b>	specify signal to output
<i>RTC_CALIBRATION_5</i> <i>12HZ</i>	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 512Hz signal
<i>RTC_CALIBRATION_1</i> <i>HZ</i>	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 1Hz signal
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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\_hour\_adjust

The description of rtc\_hour\_adjust is shown as below:

**Table 3-642. Function rtc\_hour\_adjust**

<b>Function name</b>	rtc_hour_adjust
<b>Function prototype</b>	void rtc_hour_adjust(uint32_t operation);
<b>Function descriptions</b>	adjust the daylight saving time by adding or subtracting one hour from the current time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>operation</b>	hour adjustment operation
<i>RTC_CTL_A1H</i>	add one hour
<i>RTC_CTL_S1H</i>	subtract one hour
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* adjust the daylight saving time by adding one hour from the current time */
```

```
rtc_hour_adjust(RTC_CTL_A1H);
```

## rtc\_second\_adjust

The description of rtc\_second\_adjust is shown as below:

**Table 3-643. Function rtc\_second\_adjust**

<b>Function name</b>	rtc_second_adjust
<b>Function prototype</b>	ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);
<b>Function descriptions</b>	adjust RTC second or subsecond value of current time
<b>Precondition</b>	-
<b>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>	
-	-

Return value	
ErrStatus	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\_bypass\_shadow\_enable

The description of rtc\_bypass\_shadow\_enable is shown as below:

**Table 3-644. Function rtc\_bypass\_shadow\_enable**

Function name	rtc_bypass_shadow_enable
Function prototype	void rtc_bypass_shadow_enable(void);
Function descriptions	enable RTC bypass shadow registers function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable RTC bypass shadow registers function*/
```

```
rtc_bypass_shadow_enable();
```

### rtc\_bypass\_shadow\_disable

The description of rtc\_bypass\_shadow\_disable is shown as below:

**Table 3-645. Function rtc\_bypass\_shadow\_disable**

Function name	rtc_bypass_shadow_disable
Function prototype	void rtc_bypass_shadow_disable(void);
Function descriptions	disable RTC bypass shadow registers function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* disable RTC bypass shadow registers function*/
```

```
rtc_bypass_shadow_disable();
```

### rtc\_refclock\_detection\_enable

The description of rtc\_refclock\_detection\_enable shown as below:

**Table 3-646. Function 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 3-647. Function 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\_wakeup\_enable

The description of rtc\_refclock\_detection\_disable is shown as below:

**Table 3-648. Function rtc\_wakeup\_enable**

<b>Function name</b>	rtc_wakeup_enable
<b>Function prototype</b>	void rtc_wakeup_enable(void);
<b>Function descriptions</b>	enable RTC auto wakeup function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable RTC auto wakeup function */

rtc_wakeup_enable();
```

### rtc\_wakeup\_disable

The description of rtc\_wakeup\_disable is shown as below:

**Table 3-649. Function rtc\_wakeup\_disable**

<b>Function name</b>	rtc_wakeup_disable
<b>Function prototype</b>	ErrStatus rtc_wakeup_disable(void);
<b>Function descriptions</b>	disable RTC auto wakeup function
<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:

```
/* disable RTC auto wakeup function */
```

```
ErrStatus error_status = rtc_wakeup_disable();
```

### rtc\_wakeup\_clock\_set

The description of rtc\_wakeup\_clock\_set is shown as below:

**Table 3-650. Function rtc\_wakeup\_clock\_set**

Function name	rtc_wakeup_clock_set
Function prototype	ErrStatus rtc_wakeup_clock_set(uint8_t wakeup_clock);
Function descriptions	set RTC auto wakeup timer clock
Precondition	-
The called functions	-
Input parameter{in}	
wakeup_clock	wakeup timer clock is RTC clock divided factor
WAKEUP_RTCK_DIV 16	RTC auto wakeup timer clock is RTC clock divided by 16
WAKEUP_RTCK_DIV 8	RTC auto wakeup timer clock is RTC clock divided by 8
WAKEUP_RTCK_DIV 4	RTC auto wakeup timer clock is RTC clock divided by 4
WAKEUP_RTCK_DIV 2	RTC auto wakeup timer clock is RTC clock divided by 2
WAKEUP_CKSPRE	RTC auto wakeup timer clock is ckspre
WAKEUP_CKSPRE_2 EXP16	RTC auto wakeup timer clock is ckspre and wakeup timer add 2exp16
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* RTC auto wakeup timer clock is ckspre */
```

```
ErrStatus error_status = rtc_wakeup_clock_set(WAKEUP_RTCK_DIV8);
```

### rtc\_wakeup\_timer\_set

The description of rtc\_wakeup\_timer\_set is shown as below:

**Table 3-651. Function rtc\_wakeup\_timer\_set**

Function name	rtc_wakeup_timer_set
Function prototype	ErrStatus rtc_wakeup_timer_set(uint16_t wakeup_timer);
Function descriptions	set wakeup timer value
Precondition	-

The called functions	-
Input parameter{in}	
wakeup_timer	wakeup timer value
uint16_t	0x0000-0xffff
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* set wakeup timer value */
```

```
ErrStatus error_status = rtc_wakeup_timer_set(0XFFEE);
```

### rtc\_wakeup\_timer\_get

The description of rtc\_wakeup\_timer\_set is shown as below:

**Table 3-652. Function rtc\_wakeup\_timer\_get**

Function name	rtc_wakeup_timer_get
Function prototype	uint16_t rtc_wakeup_timer_get(void);
Function descriptions	set wakeup timer value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	0-0xFFFF

Example:

```
/* get wakeup timer value */
```

```
uint32_t wakeup_time = rtc_wakeup_timer_get();
```

### rtc\_smooth\_calibration\_config

The description of rtc\_smooth\_calibration\_config is shown as below:

**Table 3-653. Function rtc\_smooth\_calibration\_config**

Function name	rtc_smooth_calibration_config
Function prototype	ErrStatus rtc_smooth_calibration_config(uint32_t window, uint32_t plus, uint32_t minus);
Function descriptions	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\_interrupt\_enable

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

**Table 3-654. 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



<i>RTC_INT_ALARM1</i>	Alarm1 interrupt
<i>RTC_INT_TAMP_ALL</i>	tamp interrupt
<i>RTC_INT_TAMP0</i>	Tamper0 detection interrupt
<i>RTC_INT_TAMP1</i>	Tamper1 detection interrupt
<i>RTC_INT_TAMP2</i>	Tamper2 detection interrupt
<i>RTC_INT_WAKEUP</i>	wakeup timer interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable specified RTC interrupt*/
```

```
rtc_interrupt_enable(RTC_INT_TAMP0);
```

### rtc\_interrupt\_disable

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

**Table 3-655. Function rtc\_interrupt\_disable**

<b>Function name</b>	rtc_interrupt_disable
<b>Function prototype</b>	void rtc_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disble specified RTC interrupt
<b>Precondition</b>	-
<b>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
<i>RTC_INT_ALARM1</i>	Alarm1 interrupt
<i>RTC_INT_TAMP_ALL</i>	All tamp interrupt
<i>RTC_INT_TAMP0</i>	Tamper0 detection interrupt
<i>RTC_INT_TAMP1</i>	Tamper1 detection interrupt
<i>RTC_INT_TAMP2</i>	Tamper2 detection interrupt
<i>RTC_INT_WAKEUP</i>	wakeup timer interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disble specified RTC interrupt */
```

```
rtc_interrupt_disable(RTC_INT_TAMP0);
```

## rtc\_flag\_get

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

**Table 3-656. 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_TP2</i>	RTC tamper 2 detected flag
<i>RTC_FLAG_TP1</i>	RTC tamper 1 detected flag
<i>RTC_FLAG_TP0</i>	RTC tamper 0 detected 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_ALARM1</i>	Alarm1 event flag
<i>RTC_FLAG_WT</i>	wakeup timer occurs 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
<i>RTC_FLAG_ALARM1</i> <i>W</i>	Alarm1 written available flag
<i>RTC_FLAG_WTW</i>	wakeup timer can be written flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
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 3-657. 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_TP2</i>	RTC tamper 2 detected flag
<i>RTC_FLAG_TP1</i>	RTC tamper 1 detected flag
<i>RTC_FLAG_TP0</i>	RTC tamper 0 detected flag
<i>RTC_FLAG_TSOVR</i>	time-stamp overflow event flag
<i>RTC_FLAG_TS</i>	time-stamp event flag
<i>RTC_FLAG_WT</i>	wakeup timer occurs flag
<i>RTC_FLAG_ALARM0</i>	Alarm0 event flag
<i>RTC_FLAG_ALARM1</i>	Alarm1 event flag
<i>RTC_FLAG_RSYN</i>	time and date registers synchronized event flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear time-stamp event flag */
```

```
rtc_flag_clear(RTC_FLAG_TS);
```

### rtc\_lxtal\_stab\_reset\_enable

The description of rtc\_lxtal\_stab\_reset\_enable is shown as below:

Table 3-658. Function rtc\_lxtal\_stab\_reset\_enable

<b>Function name</b>	rtc_lxtal_stab_reset_enable
<b>Function prototype</b>	void rtc_lxtal_stab_reset_enable (void);
<b>Function descriptions</b>	enable LXTAL stabilization reset, this workaround is only for Rev.Code E
<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*/
```

```
rtc_lxtal_stab_reset_enable ();
```

### rtc\_lxtal\_stab\_reset\_disable

The description of rtc\_lxtal\_stab\_reset\_disable shown as below:

**Table 3-659. Function rtc\_lxtal\_stab\_reset\_disable**

<b>Function name</b>	rtc_lxtal_stab_reset_disable
<b>Function prototype</b>	void rtc_lxtal_stab_reset_disable (void);
<b>Function descriptions</b>	disable LXTAL stabilization reset, this workaround is only for Rev.Code E
<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 */
```

```
rtc_lxtal_stab_reset_disable( );
```

## 3.22. SLCD

The SLCD controller directly drives LCD displays by creating the AC segment and commonvoltage signals automatically. The SLCD registers are listed in chapter [3.22.1](#), the SLCD firmware functions are introduced in chapter [3.22.2](#).

### 3.22.1. Descriptions of Peripheral registers

SLCD registers are listed in the table shown as below:

**Table 3-660. SLCD Registers**

Registers	Descriptions
SLCD_CTL	SLCD control register
SLCD_CFG	SLCD configuration register
SLCD_STAT	SLCD status register
SLCD_STATC	SLCD status flag clear register

Registers	Descriptions
SLCD_DATA0	SLCD display data register 0
SLCD_DATA1	SLCD display data register 1
SLCD_DATA2	SLCD display data register 2
SLCD_DATA3	SLCD display data register 3
SLCD_DATA4	SLCD display data register 4
SLCD_DATA5	SLCD display data register 5
SLCD_DATA6	SLCD display data register 6
SLCD_DATA7	SLCD display data register 7

### 3.22.2. Descriptions of Peripheral functions

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

**Table 3-661. SLCD firmware function**

Function name	Function description
slcd_deinit	reset SLCD interface
slcd_enable	enable SLCD interface
slcd_disable	disable SLCD interface
slcd_init	initialize SLCD interface
slcd_enhance_mode_enable	enable SLCD enhance mode (only for GD32L233xx series)
slcd_enhance_mode_disable	disable SLCD enhance mode (only for GD32L233xx series)
slcd_weak_driving_resistance_select	select SLCD weak driving resistance (only for GD32L235xx series)
slcd_bias_voltage_select	select SLCD bias voltage
slcd_duty_select	select SLCD duty
slcd_clock_config	configure SLCD input clock
slcd_blink_mode_config	configure SLCD blink mode
slcd_contrast_ratio_config	configure SLCD contrast ratio
slcd_dead_time_config	configure SLCD dead time duration
slcd_pulse_on_duration_config	configure SLCD pulse on duration (only for GD32L233xx series)
slcd_com_seg_remap	select SLCD common/segment pad
slcd_voltage_source_select	select SLCD voltage source
slcd_high_drive_config	enable or disable permanent high drive
slcd_data_register_write	write SLCD display data registers
slcd_data_update_request	update SLCD data request
slcd_flag_get	get SLCD status flag
slcd_flag_clear	clear SLCD flag
slcd_interrupt_enable	enable SLCD interrupt
slcd_interrupt_disable	disable SLCD interrupt
slcd_interrupt_flag_get	get SLCD interrupt flag
slcd_interrupt_flag_clear	clear SLCD interrupt flag

## Enum slcd\_data\_register\_enum

**Table 3-662. Enum slcd\_data\_register\_enum**

Member name	Function description
SLCD_DATA_REG0	SLCD display data register 0
SLCD_DATA_REG1	SLCD display data register 1
SLCD_DATA_REG2	SLCD display data register 2
SLCD_DATA_REG3	SLCD display data register 3
SLCD_DATA_REG4	SLCD display data register 4
SLCD_DATA_REG5	SLCD display data register 5
SLCD_DATA_REG6	SLCD display data register 6
SLCD_DATA_REG7	SLCD display data register 7

## slcd\_deinit

The description of slcd\_deinit is shown as below:

**Table 3-663. Function slcd\_deinit**

Function name	slcd_deinit
Function prototype	void slcd_deinit(void);
Function descriptions	SLCD reset interface
Precondition	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the SLCD */
```

```
slcd_deinit();
```

## slcd\_enable

The description of slcd\_enable is shown as below:

**Table 3-664. Function slcd\_enable**

Function name	slcd_enable
Function prototype	void slcd_enable(void);
Function descriptions	enable SLCD interface
Precondition	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SLCD */
slcd_enable();
```

### slcd\_disable

The description of slcd\_disable is shown as below:

**Table 3-665. Function slcd\_disable**

Function name	slcd_disable
Function prototype	void slcd_disable(void);
Function descriptions	disable SLCD interface
Precondition	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SLCD */
slcd_disable();
```

### slcd\_init

The description of slcd\_init is shown as below:

**Table 3-666. Function slcd\_init**

Function name	slcd_init
Function prototype	void slcd_init(uint32_t prescaler, uint32_t divider, uint32_t duty, uint32_t bias);
Function descriptions	initialize SLCD interface
Precondition	-
Input parameter{in}	
prescaler	the SLCD prescaler
SLCD_PRESCALER_1	$f_{PSC} = f_{in\_clk}$
SLCD_PRESCALER_2	$f_{PSC} = f_{in\_clk}/2$

SLCD_PRESCALER_4	$f_{PSC} = f_{in\_clk}/4$
SLCD_PRESCALER_8	$f_{PSC} = f_{in\_clk}/8$
SLCD_PRESCALER_1 6	$f_{PSC} = f_{in\_clk}/4$
SLCD_PRESCALER_3 2	$f_{PSC} = f_{in\_clk}/32$
SLCD_PRESCALER_6 4	$f_{PSC} = f_{in\_clk}/64$
SLCD_PRESCALER_1 28	$f_{PSC} = f_{in\_clk}/128$
SLCD_PRESCALER_2 56	$f_{PSC} = f_{in\_clk}/256$
SLCD_PRESCALER_5 12	$f_{PSC} = f_{in\_clk}/512$
SLCD_PRESCALER_1 024	$f_{PSC} = f_{in\_clk}/1024$
SLCD_PRESCALER_2 048	$f_{PSC} = f_{in\_clk}/2048$
SLCD_PRESCALER_4 096	$f_{PSC} = f_{in\_clk}/4096$
SLCD_PRESCALER_8 192	$f_{PSC} = f_{in\_clk}/8192$
SLCD_PRESCALER_1 6384	$f_{PSC} = f_{in\_clk}/16384$
SLCD_PRESCALER_3 2768	$f_{PSC} = f_{in\_clk}/32768$
<b>Input parameter{in}</b>	
<b>divider</b>	the SLCD divider
SLCD_DIVIDER_16	$f_{SLCD} = f_{PSC}/16$
SLCD_DIVIDER_17	$f_{SLCD} = f_{PSC}/17$
SLCD_DIVIDER_18	$f_{SLCD} = f_{PSC}/18$
SLCD_DIVIDER_19	$f_{SLCD} = f_{PSC}/19$
SLCD_DIVIDER_20	$f_{SLCD} = f_{PSC}/20$
SLCD_DIVIDER_21	$f_{SLCD} = f_{PSC}/21$
SLCD_DIVIDER_22	$f_{SLCD} = f_{PSC}/22$
SLCD_DIVIDER_23	$f_{SLCD} = f_{PSC}/23$
SLCD_DIVIDER_24	$f_{SLCD} = f_{PSC}/24$
SLCD_DIVIDER_25	$f_{SLCD} = f_{PSC}/25$
SLCD_DIVIDER_26	$f_{SLCD} = f_{PSC}/26$
SLCD_DIVIDER_27	$f_{SLCD} = f_{PSC}/27$
SLCD_DIVIDER_28	$f_{SLCD} = f_{PSC}/28$
SLCD_DIVIDER_29	$f_{SLCD} = f_{PSC}/29$



SLCD_DIVIDER_30	f <sub>SLCD</sub> = f <sub>PSC</sub> /30
SLCD_DIVIDER_31	f <sub>SLCD</sub> = f <sub>PSC</sub> /31
<b>Input parameter{in}</b>	
<b>duty</b>	the SLCD duty
SLCD_DUTY_STATIC	static duty
SLCD_DUTY_1_2	1/2 duty
SLCD_DUTY_1_3	1/3 duty
SLCD_DUTY_1_4	1/4 duty
SLCD_DUTY_1_6	1/6 duty
SLCD_DUTY_1_8	1/8 duty
<b>Input parameter{in}</b>	
<b>bias</b>	the SLCD voltage bias
SLCD_BIAS_1_4	1/4 voltage bias
SLCD_BIAS_1_2	1/2 voltage bias
SLCD_BIAS_1_3	1/3 voltage bias
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize SLCD interface */
```

```
slcd_init(SLCD_PRESCALER_16,          SLCD_DIVIDER_20,          SLCD_DUTY_1_4,
SLCD_BIAS_1_4);
```

### slcd\_enhance\_mode\_enable (only for GD32L233xx series)

The description of slcd\_enhance\_mode\_enable is shown as below:

**Table 3-667. Function slcd\_enhance\_mode\_enable**

<b>Function name</b>	slcd_enhance_mode_enable
<b>Function prototype</b>	void slcd_enhance_mode_enable(void);
<b>Function descriptions</b>	enable SLCD enhance mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SLCD enhance mode */
```

```
slcd_enhance_mode_enable();
```

### slcd\_enhance\_mode\_disable (only for GD32L233xx series)

The description of slcd\_enhance\_mode\_disable is shown as below:

**Table 3-668. Function slcd\_enhance\_mode\_disable**

<b>Function name</b>	slcd_enhance_mode_disable
<b>Function prototype</b>	void slcd_enhance_mode_disable(void);
<b>Function descriptions</b>	disable SLCD enhance mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SLCD enhance mode */
slcd_enhance_mode_disable();
```

### slcd\_weak\_driving\_resistance\_select (only for GD32L235xx series)

The description of slcd\_weak\_driving\_resistance\_select is shown as below:

**Table 3-669. Function slcd\_weak\_driving\_resistance\_select**

<b>Function name</b>	slcd_weak_driving_resistance_select
<b>Function prototype</b>	void slcd_weak_driving_resistance_select(uint32_t resistance);
<b>Function descriptions</b>	select SLCD weak driving resistance
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>resistance</b>	weak driving resistance
SLCD_RSEL_6M	weak driving resistance 6M
SLCD_RSEL_4M	weak driving resistance 4M
SLCD_RSEL_2M	weak driving resistance 2M
SLCD_RSEL_1M	weak driving resistance 1M
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select weak driving resistance */
```

```
slcd_weak_driving_resistance_select(SLCD_RSEL_1M);
```

### slcd\_bias\_voltage\_select

The description of slcd\_bias\_voltage\_select is shown as below:

**Table 3-670. Function slcd\_bias\_voltage\_select**

<b>Function name</b>	slcd_bias_voltage_select
<b>Function prototype</b>	void slcd_bias_voltage_select(uint32_t bias_voltage);
<b>Function descriptions</b>	select SLCD bias voltage
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>bias_voltage</b>	the SLCD voltage bias
SLCD_BIAS_1_4	1/4 voltage bias
SLCD_BIAS_1_2	1/2 voltage bias
SLCD_BIAS_1_3	1/3 voltage bias
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the SLCD 1/4 bias voltage */
```

```
slcd_bias_voltage_select(SLCD_BIAS_1_4);
```

### slcd\_duty\_select

The description of slcd\_duty\_select is shown as below:

**Table 3-671. Function slcd\_duty\_select**

<b>Function name</b>	slcd_duty_select
<b>Function prototype</b>	void slcd_duty_select(uint32_t duty);
<b>Function descriptions</b>	select SLCD duty
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>duty</b>	duty select
SLCD_DUTY_STATIC	static duty
SLCD_DUTY_1_2	1/2 duty
SLCD_DUTY_1_3	1/3 duty
SLCD_DUTY_1_4	1/4 duty
SLCD_DUTY_1_6	1/6 duty
SLCD_DUTY_1_8	1/8 duty
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* select SLCD duty */
slcd_duty_select(SLCD_DUTY_1_2);
```

### slcd\_clock\_config

The description of slcd\_clock\_config is shown as below:

**Table 3-672. Function slcd\_clock\_config**

Function name	slcd_clock_config
Function prototype	void slcd_clock_config(uint32_t prescaler,uint32_t divider);
Function descriptions	configure SLCD input clock
Precondition	-
Input parameter{in}	
prescaler	the prescaler factor
SLCD_PRESCALER_1	$f_{PSC} = f_{in\_clk}$
SLCD_PRESCALER_2	$f_{PSC} = f_{in\_clk}/2$
SLCD_PRESCALER_4	$f_{PSC} = f_{in\_clk}/4$
SLCD_PRESCALER_8	$f_{PSC} = f_{in\_clk}/8$
SLCD_PRESCALER_16	$f_{PSC} = f_{in\_clk}/16$
SLCD_PRESCALER_32	$f_{PSC} = f_{in\_clk}/32$
SLCD_PRESCALER_64	$f_{PSC} = f_{in\_clk}/64$
SLCD_PRESCALER_128	$f_{PSC} = f_{in\_clk}/128$
SLCD_PRESCALER_256	$f_{PSC} = f_{in\_clk}/256$
SLCD_PRESCALER_512	$f_{PSC} = f_{in\_clk}/512$
SLCD_PRESCALER_1024	$f_{PSC} = f_{in\_clk}/1024$
SLCD_PRESCALER_2048	$f_{PSC} = f_{in\_clk}/2048$
SLCD_PRESCALER_4096	$f_{PSC} = f_{in\_clk}/4096$
SLCD_PRESCALER_8192	$f_{PSC} = f_{in\_clk}/8192$
SLCD_PRESCALER_16384	$f_{PSC} = f_{in\_clk}/16384$

6384	
SLCD_PRESCALER_3	$f_{PSC} = f_{in\_clk}/32768$
2768	
<b>Input parameter{in}</b>	
<b>divider</b>	the divider factor
SLCD_DIVIDER_16	$f_{SLCD} = f_{PSC}/16$
SLCD_DIVIDER_17	$f_{SLCD} = f_{PSC}/17$
SLCD_DIVIDER_18	$f_{SLCD} = f_{PSC}/18$
SLCD_DIVIDER_19	$f_{SLCD} = f_{PSC}/19$
SLCD_DIVIDER_20	$f_{SLCD} = f_{PSC}/20$
SLCD_DIVIDER_21	$f_{SLCD} = f_{PSC}/21$
SLCD_DIVIDER_22	$f_{SLCD} = f_{PSC}/22$
SLCD_DIVIDER_23	$f_{SLCD} = f_{PSC}/23$
SLCD_DIVIDER_24	$f_{SLCD} = f_{PSC}/24$
SLCD_DIVIDER_25	$f_{SLCD} = f_{PSC}/25$
SLCD_DIVIDER_26	$f_{SLCD} = f_{PSC}/26$
SLCD_DIVIDER_27	$f_{SLCD} = f_{PSC}/27$
SLCD_DIVIDER_28	$f_{SLCD} = f_{PSC}/28$
SLCD_DIVIDER_29	$f_{SLCD} = f_{PSC}/29$
SLCD_DIVIDER_30	$f_{SLCD} = f_{PSC}/30$
SLCD_DIVIDER_31	$f_{SLCD} = f_{PSC}/31$
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the prescaler and the divider of SLCD clock */
slcd_clock_config(SLCD_PRESCALER_4, SLCD_DIVIDER_19);
```

### slcd\_blink\_mode\_config

The description of slcd\_blink\_mode\_config is shown as below:

**Table 3-673. Function slcd\_blink\_mode\_config**

<b>Function name</b>	slcd_blink_mode_config
<b>Function prototype</b>	void slcd_blink_mode_config(uint32_t mode,uint32_t blink_divider);
<b>Function descriptions</b>	configure SLCD blink mode
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>mode</b>	blink mode
SLCD_BLINKMODE_0 FF	blink disabled

<i>SLCD_BLINKMODE_SEG0_COM0</i>	blink enabled on SEG[0], COM[0]
<i>SLCD_BLINKMODE_SEG0_ALLCOM</i>	blink enabled on SEG[0], all COM
<i>SLCD_BLINKMODE_ALLSEG_ALLCOM</i>	blink enabled on all SEG and all COM
<b>Input parameter{in}</b>	
<b>divider</b>	the divider factor
<i>SLCD_BLINK_FREQUENCY_DIV8</i>	blink frequency = $f_{SLCD}/8$
<i>SLCD_BLINK_FREQUENCY_DIV16</i>	blink frequency = $f_{SLCD}/16$
<i>SLCD_BLINK_FREQUENCY_DIV32</i>	blink frequency = $f_{SLCD}/32$
<i>SLCD_BLINK_FREQUENCY_DIV64</i>	blink frequency = $f_{SLCD}/64$
<i>SLCD_BLINK_FREQUENCY_DIV128</i>	blink frequency = $f_{SLCD}/128$
<i>SLCD_BLINK_FREQUENCY_DIV256</i>	blink frequency = $f_{SLCD}/256$
<i>SLCD_BLINK_FREQUENCY_DIV512</i>	blink frequency = $f_{SLCD}/512$
<i>SLCD_BLINK_FREQUENCY_DIV1024</i>	blink frequency = $f_{SLCD}/1024$
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SLCD blink mode */
```

```
slcd_blink_mode_config(SLCD_BLINKMODE_SEG0_COM0,  
SLCD_BLINK_FREQUENCY_DIV8);
```

### slcd\_contrast\_ratio\_config

The description of slcd\_contrast\_ratio\_config is shown as below:

**Table 3-674. Function slcd\_contrast\_ratio\_config**

<b>Function name</b>	slcd_contrast_ratio_config
<b>Function prototype</b>	void slcd_contrast_ratio_config(uint32_t contrast_ratio);
<b>Function descriptions</b>	configure SLCD contrast ratio
<b>Precondition</b>	-

Input parameter{in}	
<b>contrast_ratio</b>	specify the VSLCD voltage
SLCD_CONTRAST_LE VEL_0	maximum SLCD Voltage = 2.65V
SLCD_CONTRAST_LE VEL_1	maximum SLCD Voltage = 2.80V
SLCD_CONTRAST_LE VEL_2	maximum SLCD Voltage = 2.92V
SLCD_CONTRAST_LE VEL_3	maximum SLCD Voltage = 3.08V
SLCD_CONTRAST_LE VEL_4	maximum SLCD Voltage = 3.23V
SLCD_CONTRAST_LE VEL_5	maximum SLCD Voltage = 3.37V
SLCD_CONTRAST_LE VEL_6	maximum SLCD Voltage = 3.52V
SLCD_CONTRAST_LE VEL_7	maximum SLCD Voltage = 3.67V
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SLCD contrast ratio */
```

```
slcd_contrast_ratio_config(SLCD_CONTRAST_LEVEL_0);
```

### slcd\_dead\_time\_config

The description of slcd\_dead\_time\_config is shown as below:

**Table 3-675. Function slcd\_dead\_time\_config**

<b>Function name</b>	slcd_dead_time_config
<b>Function prototype</b>	void slcd_dead_time_config(uint32_t dead_time);
<b>Function descriptions</b>	configure SLCD dead time duration
<b>Precondition</b>	-
Input parameter{in}	
<b>dead_time</b>	the length of the dead time between frames
SLCD_DEADTIME_PE RIOD_0	no dead time
SLCD_DEADTIME_PE RIOD_1	1 phase inserted between couple of frame
SLCD_DEADTIME_PE	2 phase inserted between couple of frame

<i>RIOD_2</i>	
<i>SLCD_DEADTIME_PE</i> <i>RIOD_3</i>	3 phase inserted between couple of frame
<i>SLCD_DEADTIME_PE</i> <i>RIOD_4</i>	4 phase inserted between couple of frame
<i>SLCD_DEADTIME_PE</i> <i>RIOD_5</i>	5 phase inserted between couple of frame
<i>SLCD_DEADTIME_PE</i> <i>RIOD_6</i>	6 phase inserted between couple of frame
<i>SLCD_DEADTIME_PE</i> <i>RIOD_7</i>	7 phase inserted between couple of frame
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SLCD dead time duration */
```

```
slcd_dead_time_config(SLCD_DEADTIME_PERIOD_1);
```

### slcd\_pulse\_on\_duration\_config (only for GD32L233xx series)

The description of slcd\_pulse\_on\_duration\_config is shown as below:

**Table 3-676. Function slcd\_pulse\_on\_duration\_config**

<b>Function name</b>	slcd_pulse_on_duration_config
<b>Function prototype</b>	void slcd_pulse_on_duration_config(uint32_t duration);
<b>Function descriptions</b>	configure SLCD pulse on duration
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>duration</b>	the pulse duration in terms of PSC pulses
<i>SLCD_PULSEON_DURATION_0</i>	pulse on duration = 0
<i>SLCD_PULSEON_DURATION_1</i>	pulse on duration = $1/f_{psc}$
<i>SLCD_PULSEON_DURATION_2</i>	pulse on duration = $2/f_{psc}$
<i>SLCD_PULSEON_DURATION_3</i>	pulse on duration = $3/f_{psc}$
<i>SLCD_PULSEON_DURATION_4</i>	pulse on duration = $4/f_{psc}$
<i>SLCD_PULSEON_DURATION_5</i>	pulse on duration = $5/f_{psc}$



<i>SLCD_PULSEON_DURATION_6</i>	pulse on duration = $6/f_{psc}$
<i>SLCD_PULSEON_DURATION_7</i>	pulse on duration = $7/f_{psc}$
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SLCD pulse on duration */
```

```
slcd_pulse_on_duration_config(SLCD_PULSEON_DURATION_7);
```

### slcd\_com\_seg\_remap

The description of slcd\_com\_seg\_remap is shown as below:

**Table 3-677. Function slcd\_com\_seg\_remap**

<b>Function name</b>	slcd_com_seg_remap
<b>Function prototype</b>	void slcd_com_seg_remap(ControlStatus newvalue);
<b>Function descriptions</b>	select SLCD common/segment pad
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>NewValue</b>	ENABLE or DISABLE
<i>ENABLE</i>	LCD_COM[7:4] pad select LCD_SEG[31:28]
<i>DISABLE</i>	LCD_COM[7:4] pad select LCD_COM[7:4]
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select SLCD common/segment pad */
```

```
slcd_com_seg_remap(ENABLE);
```

### slcd\_voltage\_source\_select

The description of slcd\_voltage\_source\_select is shown as below:

**Table 3-678. Function slcd\_voltage\_source\_select**

<b>Function name</b>	slcd_voltage_source_select
<b>Function prototype</b>	void slcd_voltage_source_select(uint8_t voltage_source);
<b>Function descriptions</b>	select SLCD voltage source

<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>voltage_source</b>	the SLCD voltage source
<i>SLCD_VOLTAGE_INTERNAL</i>	internal source
<i>SLCD_VOLTAGE_EXTERNAL</i>	external source (VSLCD pin)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select SLCD voltage source */
slcd_voltage_source_select(SLCD_VOLTAGE_EXTERNAL);
```

### slcd\_high\_drive\_config

The description of slcd\_high\_drive\_config is shown as below:

**Table 3-679. Function slcd\_high\_drive\_config**

<b>Function name</b>	slcd_high_drive_config
<b>Function prototype</b>	void slcd_high_drive_config(ControlStatus newvalue);
<b>Function descriptions</b>	enable or disable permanent high drive
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>NewValue</b>	ENABLE or DISABLE
<i>ENABLE</i>	Permanent high drive enabled
<i>DISABLE</i>	Permanent high drive disabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable permanent high drive */
slcd_high_drive_config(ENABLE);
```

### slcd\_data\_register\_write

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

Table 3-680. Function slcd\_data\_register\_write

Function name	slcd_data_register_write
Function prototype	void slcd_data_register_write(slcd_data_register_enum register_number, uint32_t data);
Function descriptions	write SLCD data register
Precondition	-
Input parameter{in}	
register_number	refer to <a href="#">Table 3-662. Enum slcd_data_register_enum.</a>
SLCD_DATA_REGx(x=0, 1, ..., 7)	SLCD_DATAx register
Input parameter{in}	
data	the data write to the register
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write SLCD data register */
slcd_data_register_write(SLCD_DATA_REG0, 0x0000FFFF);
```

### slcd\_data\_update\_request

The description of slcd\_data\_update\_request is shown as below:

Table 3-681. Function slcd\_data\_update\_request

Function name	slcd_data_update_request
Function prototype	void slcd_data_update_request(void);
Function descriptions	update SLCD data request
Precondition	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* update SLCD data request */
slcd_data_update_request();
```

## slcd\_flag\_get

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

**Table 3-682. Function slcd\_flag\_get**

<b>Function name</b>	slcd_flag_get
<b>Function prototype</b>	FlagStatus slcd_flag_get(uint8_t flag);
<b>Function descriptions</b>	get the SLCD flag
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	status flag
SLCD_FLAG_ON	SLCD controller on flag
SLCD_FLAG_SOF	start of frame flag
SLCD_FLAG_UPR	SLCD data update request flag
SLCD_FLAG_UPD	update SLCD data done flag
SLCD_FLAG_VRDY	SLCD voltage ready flag (only for GD32L233xx series)
SLCD_FLAG_SYN	SLCD CFG register synchronization flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the SLCD status flag */
```

```
slcd_flag_get(SLCD_FLAG_ON);
```

## slcd\_flag\_clear

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

**Table 3-683. Function slcd\_flag\_clear**

<b>Function name</b>	slcd_flag_clear
<b>Function prototype</b>	void slcd_flag_clear(uint8_t flag);
<b>Function descriptions</b>	Clear the SLCD status flag
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	status flag
SLCD_FLAG_SOF	start of frame flag
SLCD_FLAG_UPD	update SLCD data done flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the SLCD status flag */

slcd_flag_clear(SLCD_FLAG_SOF);
```

### slcd\_interrupt\_enable

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

**Table 3-684. Function slcd\_interrupt\_enable**

<b>Function name</b>	slcd_interrupt_enable
<b>Function prototype</b>	void slcd_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable the SLCD interrupt
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt source
SLCD_INT_SOF	start of frame interrupt
SLCD_INT_UPD	SLCD update done interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the start of frame interrupt */

slcd_interrupt_enable(SLCD_INT_SOF);
```

### slcd\_interrupt\_disable

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

**Table 3-685. Function slcd\_interrupt\_disable**

<b>Function name</b>	slcd_interrupt_disable
<b>Function prototype</b>	void slcd_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable the SLCD interrupt
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt source
SLCD_INT_SOF	start of frame interrupt
SLCD_INT_UPD	SLCD update done interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the start of frame interrupt */
```

```
slcd_interrupt_disable(SLCD_INT_SOF);
```

### slcd\_interrupt\_flag\_get

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

**Table 3-686. Function slcd\_interrupt\_flag\_get**

<b>Function name</b>	slcd_interrupt_flag_get
<b>Function prototype</b>	FlagStatus slcd_interrupt_flag_get(uint8_t interrupt);
<b>Function descriptions</b>	get the SLCD interrupt flag
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt source
SLCD_INT_FLAG_SOF	start of frame interrupt
SLCD_INT_FLAG_UPD	SLCD update done interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the SLCD interrupt flag */
```

```
slcd_interrupt_flag_get(SLCD_INT_FLAG_SOF);
```

### slcd\_interrupt\_flag\_clear

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

**Table 3-687. Function slcd\_interrupt\_flag\_clear**

<b>Function name</b>	slcd_interrupt_flag_clear
<b>Function prototype</b>	FlagStatus slcd_interrupt_flag_clear(uint8_t interrupt);
<b>Function descriptions</b>	clear the SLCD interrupt flag
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt source
SLCD_INT_FLAG_SOF	start of frame interrupt
SLCD_INT_FLAG_UPD	SLCD update done interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the SLCD interrupt flag */

slcd_interrupt_flag_clear(SLCD_INT_FLAG_SOF);
```

## 3.23. 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 [3.23.1](#), the SPI/I2S firmware functions are introduced in chapter [3.23.2](#).

### 3.23.1. Descriptions of Peripheral registers

SPI/I2S registers are listed in the table shown as below:

**Table 3-688. 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

### 3.23.2. Descriptions of Peripheral functions

SPI/I2S firmware functions are listed in the table shown as below:

**Table 3-689. 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

Function name	Function description
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
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 3-690. spi\_parameter\_struct

Member name	Function description
device_mode	SPI master or slave



	(SPI_MASTER, SPI_SLAVE)
trans_mode	SPI transtype (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	SPI frame size (SPI_FRAME_SIZE_16BIT, SPI_FRAME_SIZE_8BIT)
nss	SPI NSS control by hardware or software (SPI_NSS_SOFT, SPI_NSS_HARD)
endian	SPI big endian or little endian (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)
clock_polarity_phase	SPI clock phase and polarity (SPI_CK_PL_LOW_PH_1EDGE, SPI_CK_PL_HIGH_PH_1EDGE, SPI_CK_PL_LOW_PH_2EDGE, SPI_CK_PL_HIGH_PH_2EDGE)
prescale	SPI prescale factor (SPI_PSC_n (n=2,4,8,16,32,64,128,256))

### spi\_i2s\_deinit

The description of spi\_i2s\_deinit is shown as below:

**Table 3-691. 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 3-692. Function spi\_struct\_para\_init

Function name	spi_struct_para_init
Function prototype	void spi_struct_para_init(spi_parameter_struct* spi_struct);
Function descriptions	initialize the parameters of SPI struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
spi_struct	SPI parameter struct, the structure members can refer to members of the structure <a href="#">Table 3-690. spi_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

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 3-693. 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 3-690. 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 3-694. 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>	-
Input parameter{in}	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 */

spi_enable(SPI0);
```

## spi\_disable

The description of spi\_disable is shown as below:

**Table 3-695. 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 3-696. 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=1)</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_SLAVERX</i>	I2S slave receive mode
<i>I2S_MODE_MASTERTX</i>	I2S master transmit mode
<i>I2S_MODE_MASTERRX</i>	I2S master receive mode

Input parameter{in}	
<b>standard</b>	I2S standard
<i>I2S_STD_PHILLIPS</i>	I2S phillips standard
<i>I2S_STD_MSB</i>	I2S MSB standard
<i>I2S_STD_LSB</i>	I2S LSB standard
<i>I2S_STD_PCMSHORT</i>	I2S PCM short standard
<i>I2S_STD_PCMLONG</i>	I2S PCM long standard
Input parameter{in}	
<b>ckpl</b>	I2S idle state clock polarity
<i>I2S_CKPL_LOW</i>	I2S clock polarity low level
<i>I2S_CKPL_HIGH</i>	I2S clock polarity high level
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize I2S1 */
```

```
i2s_init(SPI1, I2S_MODE_MASTERTX, I2S_STD_PHILLIPS, I2S_CKPL_LOW);
```

### i2s\_psc\_config

The description of i2s\_psc\_config is shown as below:

**Table 3-697. 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
Input parameter{in}	
<b>spi_periph</b>	I2S peripheral
<i>SPIx(x=1)</i>	I2S peripheral selection
Input parameter{in}	
<b>audiosample</b>	I2S audio sample rate
<i>I2S_AUDIOSAMPLE_8K</i>	audio sample rate is 8KHz
<i>I2S_AUDIOSAMPLE_11K</i>	audio sample rate is 11KHz
<i>I2S_AUDIOSAMPLE_16K</i>	audio sample rate is 16KHz
<i>I2S_AUDIOSAMPLE_22K</i>	audio sample rate is 22KHz

2K	
I2S_AUDIOSAMPLE_3	audio sample rate is 32KHz
2K	
I2S_AUDIOSAMPLE_4	audio sample rate is 44KHz
4K	
I2S_AUDIOSAMPLE_4	audio sample rate is 48KHz
8K	
I2S_AUDIOSAMPLE_9	audio sample rate is 96KHz
6K	
I2S_AUDIOSAMPLE_1	audio sample rate is 192KHz
92K	
<b>Input parameter{in}</b>	
<b>frameformat</b>	I2S data length and channel length
I2S_FRAMEFORMAT_DT16B_CH16B	I2S data length is 16 bit and channel length is 16 bit
I2S_FRAMEFORMAT_DT16B_CH32B	I2S data length is 16 bit and channel length is 32 bit
I2S_FRAMEFORMAT_DT24B_CH32B	I2S data length is 24 bit and channel length is 32 bit
I2S_FRAMEFORMAT_DT32B_CH32B	I2S data length is 32 bit and channel length is 32 bit
<b>Input parameter{in}</b>	
<b>mckout</b>	I2S master clock output
I2S_MCKOUT_ENABL	I2S master clock output enable
E	
I2S_MCKOUT_DISABL	I2S master clock output disable
E	
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure I2S1 prescaler */
```

```
i2s_psc_config(SPI1, I2S_AUDIOSAMPLE_44K, I2S_FRAMEFORMAT_DT16B_CH16B,
I2S_MCKOUT_DISABLE);
```

### i2s\_enable

The description of i2s\_enable is shown as below:

**Table 3-698. 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
<b>SPIx(x=1)</b>	I2S peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2S1*/
i2s_enable(SPI1);
```

### i2s\_disable

The description of i2s\_disable is shown as below:

**Table 3-699. 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=1)</b>	I2S peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2S1*/
i2s_disable(SPI1);
```

### spi\_nss\_output\_enable

The description of spi\_nss\_output\_enable is shown as below:

Table 3-700. Function spi\_nss\_output\_enable

Function name	spi_nss_output_enable
Function prototype	void spi_nss_output_enable(uint32_t spi_periph);
Function descriptions	enable SPI NSS output 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:

```
/* 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 3-701. Function spi\_nss\_output\_disable

Function name	spi_nss_output_disable
Function prototype	void spi_nss_output_disable(uint32_t spi_periph);
Function descriptions	disable SPI NSS output 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:

```
/* 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 3-702. Function spi\_nss\_internal\_high

Function name	spi_nss_internal_high
Function prototype	void spi_nss_internal_high(uint32_t spi_periph);
Function descriptions	SPI NSS pin high level in software 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:

```
/* 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 3-703. Function spi\_nss\_internal\_low

Function name	spi_nss_internal_low
Function prototype	void spi_nss_internal_low(uint32_t spi_periph);
Function descriptions	SPI NSS pin low level in software 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:

```
/* 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 3-704. Function spi\_dma\_enable

Function name	spi_dma_enable
Function prototype	void spi_dma_enable(uint32_t spi_periph, uint8_t dma);
Function descriptions	enable SPI DMA function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx(x=0,1)	SPI peripheral selection
Input parameter{in}	
dma	SPI DMA mode
SPI_DMA_TRANSMIT	SPI transmit data use DMA
SPI_DMA_RECEIVE	SPI receive data use DMA
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 transmit data DMA function */
spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);
```

### spi\_dma\_disable

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

Table 3-705. Function spi\_dma\_disable

Function name	spi_dma_disable
Function prototype	void spi_dma_disable(uint32_t spi_periph, uint8_t dma);
Function descriptions	disable SPI DMA function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx(x=0,1)	SPI peripheral selection
Input parameter{in}	
dma	SPI DMA mode
SPI_DMA_TRANSMIT	SPI transmit data use DMA
SPI_DMA_RECEIVE	SPI receive data use DMA
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 transmit data DMA function */
spi_dma_disable(SPI0, SPI_DMA_TRANSMIT);
```

### spi\_transmit\_odd\_config

The description of spi\_transmit\_odd\_config is shown as below:

**Table 3-706. 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=0)</i>	SPI peripheral selection
<b>Input parameter{in}</b>	
<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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SPI0 total number of data to transmit by DMA is odd */
spi_transmit_odd_config (SPI0, SPI_TXDMA_ODD);
```

### spi\_receive\_odd\_config

The description of spi\_receive\_odd\_config is shown as below:

**Table 3-707. 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>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0)</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 SPI0 total number of data to receive by DMA is odd */
```

```
spi_receive_odd_config (SPI0, SPI_RXDMA_ODD);
```

### spi\_i2s\_data\_frame\_format\_config

The description of spi\_i2s\_data\_frame\_format\_config is shown as below:

**Table 3-708. Function spi\_i2s\_data\_frame\_format\_config**

<b>Function name</b>	spi_i2s_data_frame_format_config
<b>Function prototype</b>	void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
<b>Function descriptions</b>	configure SPI/I2S data frame format
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0, 1)</i>	SPI peripheral selection
Input parameter{in}	
<b>frame_format</b>	SPI frame size
<i>SPI_FRAME_SIZE_xBIT</i>	SPI frame size is x bits, x=4,5,..16
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SPI1/I2S1 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 3-709. Function spi\_fifo\_access\_size\_config

Function name	spi_fifo_access_size_config
Function prototype	void spi_fifo_access_size_config(uint32_t spi_periph, uint16_t fifo_access_size);
Function descriptions	configure SPI0 access size to FIFO(8bit or 16bit)
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx(x=0)	SPI peripheral selection
Input parameter{in}	
frame_format	SPI frame size
SPI_HALFWORD_ACCESS	half-word access to FIFO
SPI_BYTE_ACCESS	byte access to FIFO
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SPI0 access size half word */
```

```
spi_fifo_access_size_config (SPI0, SPI_HALFWORD_ACCESS);
```

### spi\_bidirectional\_transfer\_config

The description of spi\_bidirectional\_transfer\_config is shown as below:

Table 3-710. Function spi\_bidirectional\_transfer\_config

Function name	spi_bidirectional_transfer_config
Function prototype	void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);
Function descriptions	configure SPI bidirectional transfer direction
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx(x=0,1)	SPI peripheral selection
Input parameter{in}	
transfer_direction	SPI transfer direction
SPI_BIDIRECTIONAL_TRANSMIT	SPI work in transmit-only mode
SPI_BIDIRECTIONAL_	SPI work in receive-only mode

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

Example:

```
/* SPI0 works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

### spi\_i2s\_data\_transmit

The description of spi\_i2s\_data\_transmit is shown as below:

**Table 3-711. Function spi\_i2s\_data\_transmit**

Function name	spi_i2s_data_transmit
Function prototype	void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);
Function descriptions	SPI transmit data
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx(x=0,1)	SPI peripheral selection
Input parameter{in}	
data	16-bit data
Output parameter{out}	
-	-
Return value	
-	-

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 3-712. Function spi\_i2s\_data\_receive**

Function name	spi_i2s_data_receive
Function prototype	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 3-713. 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 3-714. 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 3-715. 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=0)</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(SPI0, SPI_CRC_8BIT);
```

## spi\_crc\_on

The description of spi\_crc\_on is shown as below:

**Table 3-716. 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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn on SPI0 CRC function */
```

```
spi_crc_on(SPI0);
```

## spi\_crc\_off

The description of spi\_crc\_off is shown as below:

**Table 3-717. Function spi\_crc\_off**

<b>Function name</b>	spi_crc_off
<b>Function prototype</b>	void spi_crc_off(uint32_t spi_periph);
<b>Function descriptions</b>	turn off SPI CRC function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<b>Output parameter{out}</b>	
-	-

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 3-718. Function spi\_crc\_next**

<b>Function name</b>	spi_crc_next
<b>Function prototype</b>	void spi_crc_next(uint32_t spi_periph);
<b>Function descriptions</b>	SPI next data is CRC value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(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 3-719. 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 3-720. Function spi\_crc\_error\_clear**

<b>Function name</b>	spi_crc_error_clear
<b>Function prototype</b>	void spi_crc_error_clear(uint32_t spi_periph);
<b>Function descriptions</b>	clear SPI CRC error flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	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 3-721. Function spi\_ti\_mode\_enable**

<b>Function name</b>	spi_ti_mode_enable
<b>Function prototype</b>	void spi_ti_mode_enable (uint32_t spi_periph);
<b>Function descriptions</b>	enable SPI TI mode

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	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 3-722. Function spi\_ti\_mode\_disable**

<b>Function name</b>	spi_ti_mode_disable
<b>Function prototype</b>	void spi_ti_mode_disable (uint32_t spi_periph);
<b>Function descriptions</b>	disable SPI TI mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-723. Function spi\_nssp\_mode\_enable**

<b>Function name</b>	spi_nssp_mode_enable
<b>Function prototype</b>	void spi_nssp_mode_enable(uint32_t spi_periph);

<b>Function descriptions</b>	enable SPI NSS pulse mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-724. Function spi\_nssp\_mode\_disable**

<b>Function name</b>	spi_nssp_mode_disable
<b>Function prototype</b>	void spi_nssp_mode_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable SPI NSS pulse mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx(x=0,1)</i>	SPI peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-725. Function spi\_quad\_enable**

<b>Function name</b>	spi_quad_enable
----------------------	-----------------

<b>Function prototype</b>	void spi_quad_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable quad wire SPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx(x=0)</b>	SPI peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable quad wire SPI */
spi_quad_enable(SPI0);
```

### spi\_quad\_disable

The description of spi\_quad\_disable is shown as below:

**Table 3-726. Function spi\_quad\_disable**

<b>Function name</b>	spi_quad_disable
<b>Function prototype</b>	void spi_quad_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable quad wire SPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx(x=0)</b>	SPI peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable quad wire SPI */
spi_quad_disable(SPI0);
```

### spi\_quad\_write\_enable

The description of spi\_quad\_write\_enable is shown as below:

Table 3-727. 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=0)	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable quad wire SPI write */
spi_quad_write_enable(SPI0);
```

### spi\_quad\_read\_enable

The description of spi\_quad\_read\_enable is shown as below:

Table 3-728. Function spi\_quad\_read\_enable

Function name	spi_quad_read_enable
Function prototype	void spi_quad_read_enable(uint32_t spi_periph);
Function descriptions	enable quad wire SPI read
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx(x=0)	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable quad wire SPI read */
spi_quad_read_enable(SPI0);
```

### spi\_i2s\_format\_error\_clear

The description of spi\_i2s\_format\_error\_clear is shown as below:

Table 3-729. Function spi\_i2s\_format\_error\_clear

Function name	spi_i2s_format_error_clear
Function prototype	void spi_i2s_format_error_clear(uint32_t spi_periph, uint32_t flag);
Function descriptions	clear SPI/I2S format error flag status
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx(x=0,1)	SPI peripheral selection
Input parameter{in}	
flag	SPI/I2S frame format error flag
SPI_FLAG_FERR	SPI format error flag
I2S_FLAG_FERR	I2S format error flag
Output parameter{out}	
-	-
Return value	
-	-

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 3-730. Function spi\_i2s\_flag\_get

Function name	spi_i2s_flag_get
Function prototype	FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag);
Function descriptions	get SPI and I2S flag status
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx(x=0,1)	SPI peripheral selection
Input parameter{in}	
flag	SPI/I2S flag status
SPI_FLAG_TBE	transmit buffer empty flag
SPI_FLAG_RBNE	receive buffer not empty flag
SPI_FLAG_TRANS	transmit on-going flag
SPI_FLAG_RXORERR	receive overrun error flag
SPI_FLAG_CONFERR	mode config error flag
SPI_FLAG_CRCERR	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>SPI_FLAG_RXLVL_FULL</i>	SPI RXFIFO is full
Output parameter{out}	
-	-
Return value	
FlagStatus	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 3-731. 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 3-732. 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 3-733. Function spi\_i2s\_interrupt\_flag\_get**

Function name	spi_i2s_interrupt_flag_get
Function prototype	FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt);
Function descriptions	get SPI and I2S interrupt status
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx(x=0,1)	SPI peripheral selection
Input parameter{in}	
interrupt	SPI/I2S interrupt flag status
SPI_I2S_INT_FLAG_TBE	transmit buffer empty interrupt
SPI_I2S_INT_FLAG_RBNE	receive buffer not empty interrupt
SPI_I2S_INT_FLAG_RXORERR	overrun interrupt
SPI_INT_FLAG_CONFIGERR	config error interrupt
SPI_INT_FLAG_CRCERR	CRC error interrupt
I2S_INT_FLAG_TXURERR	underrun error interrupt
SPI_I2S_INT_FLAG_FORMATERR	format error interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	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);
```

## 3.24. SYSCFG

The SYSCFG registers are listed in chapter [3.24.1](#), the SYSCFG firmware functions are introduced in chapter [3.24.2](#).

### 3.24.1. Descriptions of Peripheral registers

SYSCFG registers are listed in the table shown as below:

**Table 3-734. 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 (only for GD32L235)
SYSCFG_CPU_IRQ_LAT	IRQ Latency register
SYSCFG_TIMERxCFG	TIMERx configuration register, x = 0,1,2,8,11,14,40 (only for GD32L235)

### 3.24.2. Descriptions of Peripheral functions

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

**Table 3-735. SYSCFG firmware function**

Function name	Function description
syscfg_deinit	deinit syscfg module
syscfg_exti_line_config	configure the GPIO pin as EXTI Line
syscfg_pin_remap_enable	enable remap pin function for small packages
syscfg_pin_remap_disable	disable remap pin function for small packages
syscfg_high_current_enable	enable PBx(x=6,7,8,9) high current capability
syscfg_high_current_disable	disable PBx(x=6,7,8,9) high current capability
irq_latency_set	set the IRQ_LATENCY value
syscfg_bootmode_get	get the boot mode
syscfg_sram_waitstate_insert	insert wait state in read accesses of SRAM0/SRAM1 when LDO is 1.1V (only for GD32L235)
syscfg_sram_waitstate_cancel	cancel insertion of wait state in read accesses of SRAM0/SRAM1 when LDO is 1.1V (only for GD32L235)
syscfg_lock_config	connect TIMER0/14/40 break input to the selected parameter (only for GD32L235)
syscfg_flag_get	check if the specified flag in SYSCFG_CFG1 is set or not (only for GD32L235)
syscfg_flag_clear	clear the flag in SYSCFG_CFG1 by writing 1 (only for

Function name	Function description
	GD32L235)

## syscfg\_deinit

The description of syscfg\_deinit is shown as below:

**Table 3-736. Function syscfg\_deinit**

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

Example:

```
/* reset SYSCFG registers */
syscfg_deinit();
```

## syscfg\_exti\_line\_config

The description of syscfg\_exti\_line\_config is shown as below:

**Table 3-737. 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\_pin\_remap\_enable

The description of syscfg\_pin\_remap\_enable is shown as below:

**Table 3-738. Function syscfg\_pin\_remap\_enable**

Function name	syscfg_pin_remap_enable
Function prototype	void syscfg_pin_remap_enable(uint32_t remap_pin);
Function descriptions	enable remap pin function for small packages
Precondition	-
The called functions	-
Input parameter{in}	
remap_pin	remap pin
SYSCFG_PA11_PA12_REMAP	PA11 PA12 remap
SYSCFG_BOOT0_PD3_REMAP	BOOT0 PD3 remap
SYSCFG_PA8_REMAP	PA8 remap (only for GD32L235)
SYSCFG_PD0_PD1_PD2_REMAP	PD0 PD1 PD2 remap (only for GD32L235)
SYSCFG_PA11_PA12_PB6_PB8_REMAP	PA11 PA12 PB6 PB8 remap (only for GD32L235)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable BOOT0 remap to PD3 function */
syscfg_pin_remap_enable(SYSCFG_BOOT0_PD3_REMAP);
```

### syscfg\_pin\_remap\_disable

The description of syscfg\_pin\_remap\_disable is shown as below:

**Table 3-739. Function syscfg\_pin\_remap\_disable**

Function name	syscfg_pin_remap_disable
Function prototype	void syscfg_pin_remap_disable(void);
Function descriptions	disable remap pin function for small packages
Precondition	-

The called functions	-
Input parameter{in}	
remap_pin	remap pin
SYSCFG_PA11_PA12_REMAP	PA11 PA12 remap
SYSCFG_BOOT0_PD3_REMAP	BOOT0 PD3 remap
SYSCFG_PA8_REMAP	PA8 remap (only for GD32L235)
SYSCFG_PD0_PD1_PD2_REMAP	PD0 PD1 PD2 remap (only for GD32L235)
SYSCFG_PA11_PA12_PB6_PB8_REMAP	PA11 PA12 PB6 PB8 remap (only for GD32L235)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable BOOT0 remap to PD3 function */
```

```
syscfg_pin_remap_disable(SYSCFG_BOOT0_PD3_REMAP);
```

### syscfg\_high\_current\_enable

The description of syscfg\_high\_current\_enable is shown as below:

**Table 3-740. Function syscfg\_high\_current\_enable**

Function name	syscfg_high_current_enable
Function prototype	void syscfg_high_current_enable(uint32_t syscfg_gpio);
Function descriptions	enable PBx(x=6,7,8,9) high current capability
Precondition	-
The called functions	-
Input parameter{in}	
syscfg_gpio	SYSCFG GPIO
SYSCFG_PB6_HIGH_CURRENT	PB6 pin high current capability
SYSCFG_PB7_HIGH_CURRENT	PB7 pin high current capability
SYSCFG_PB8_HIGH_CURRENT	PB8 pin high current capability
SYSCFG_PB9_HIGH_CURRENT	PB9 pin high current capability
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* enable PB9 high current capability */

syscfg_high_current_enable(SYSCFG_PB9_HIGH_CURRENT);
```

### syscfg\_high\_current\_disable

The description of syscfg\_high\_current\_disable is shown as below:

**Table 3-741. Function syscfg\_high\_current\_disable**

Function name	syscfg_high_current_disable
Function prototype	void syscfg_high_current_disable(uint32_t syscfg_gpio);
Function descriptions	disable PB9 high current capability
Precondition	-
The called functions	-
Input parameter{in}	
syscfg_gpio	SYSCFG GPIO
SYSCFG_PB6_HIGH_CURRENT	PB6 pin high current capability
SYSCFG_PB7_HIGH_CURRENT	PB7 pin high current capability
SYSCFG_PB8_HIGH_CURRENT	PB8 pin high current capability
SYSCFG_PB9_HIGH_CURRENT	PB9 pin high current capability
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable PB9 high current capability */

syscfg_high_current_disable(SYSCFG_PB9_HIGH_CURRENT);
```

### irq\_latency\_set

The description of irq\_latency\_set is shown as below:

**Table 3-742. Function irq\_latency\_set**

Function name	irq_latency_set
Function prototype	void irq_latency_set(uint8_t irq_latency);
Function descriptions	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 */
```

```
irq_latency_set(0xFF);
```

### syscfg\_bootmode\_get

The description of syscfg\_bootmode\_get is shown as below:

**Table 3-743. Function syscfg\_bootmode\_get**

<b>Function name</b>	syscfg_bootmode_get
<b>Function prototype</b>	uint8_t 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>	
<b>uint8_t</b>	boot mode
SYSCFG_BOOTMODE_FLASH	boot from the main flash
SYSCFG_BOOTMODE_SYSTEM	boot from the system flash memory
SYSCFG_BOOTMODE_SRAM	boot from the embedded SRAM

Example:

```
/* get the current boot mode */
```

```
uint8_t boot_mode;
```

```
boot_mode = syscfg_bootmode_get();
```

## syscfg\_sram\_waitstate\_insert

The description of syscfg\_sram\_waitstate\_insert is shown as below:

**Table 3-744. Function syscfg\_sram\_waitstate\_insert**

<b>Function name</b>	syscfg_sram_waitstate_insert
<b>Function prototype</b>	void syscfg_sram_waitstate_insert(void);
<b>Function descriptions</b>	insert wait state in read accesses of SRAM0/SRAM1 when LDO is 1.1V (only for GD32L235)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* insert wait state in read accesses of SRAM0/SRAM1 */
```

```
syscfg_sram_waitstate_insert();
```

## syscfg\_sram\_waitstate\_cancel

The description of syscfg\_sram\_waitstate\_cancel is shown as below:

**Table 3-745. Function syscfg\_sram\_waitstate\_cancel**

<b>Function name</b>	syscfg_sram_waitstate_cancel
<b>Function prototype</b>	void syscfg_sram_waitstate_cancel(void);
<b>Function descriptions</b>	cancel insertion of wait state in read accesses of SRAM0/SRAM1 when LDO is 1.1V (only for GD32L235)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* cancel insertion of wait state in read accesses of SRAM0/SRAM1 */
```

```
syscfg_sram_waitstate_cancel();
```

## syscfg\_lock\_config

The description of syscfg\_lock\_config is shown as below:

**Table 3-746. Function syscfg\_lock\_config**

<b>Function name</b>	syscfg_lock_config
<b>Function prototype</b>	void syscfg_lock_config(uint32_t syscfg_lock);
<b>Function descriptions</b>	connect TIMER0/14/40 break input to the selected parameter (only for GD32L235)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_lock</b>	specify the parameter to be connected
<i>SYSCFG_LOCK_LOCKUP</i>	Cortex-M23 lockup output connected to the break input
<i>SYSCFG_LOCK_SRAM_PARITY_ERROR</i>	SRAM_PARITY check error connected to the break input
<i>SYSCFG_LOCK_LVD</i>	LVD interrupt connected to the break input
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* connect TIMER0/14/40 break input to the SYSCFG_LOCK_LOCKUP */
syscfg_lock_config(SYSCFG_LOCK_LOCKUP);
```

## syscfg\_flag\_get

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

**Table 3-747. Function syscfg\_flag\_get**

<b>Function name</b>	syscfg_flag_get
<b>Function prototype</b>	FlagStatus syscfg_flag_get(uint32_t syscfg_flag);
<b>Function descriptions</b>	check if the specified flag in SYSCFG_CFG1 is set or not (only for GD32L235)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_flag</b>	specify the flag in SYSCFG_CFG1 to check
<i>SYSCFG_FLAG_SRAM_PCEF</i>	SRAM parity check error flag
<b>Output parameter{out}</b>	
-	-

Return value	
FlagStatus	SET or RESET

Example:

```
FlagStatus tempflag;
```

```
/* get the SYSCFG flag */
```

```
tempflag = syscfg_flag_get(SYSCFG_FLAG_SRAM_PCEF);
```

### syscfg\_flag\_clear

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

**Table 3-748. Function syscfg\_flag\_clear**

Function name	syscfg_flag_clear
Function prototype	void syscfg_flag_clear(uint32_t syscfg_flag);
Function descriptions	clear the flag in SYSCFG_CFG1 by writing 1 (only for GD32L235)
Precondition	-
The called functions	-
Input parameter{in}	
syscfg_flag	specify the flag in SYSCFG_CFG1 to clear
SYSCFG_FLAG_SRAM_PCEF	SRAM parity check error flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the SYSCFG flag */
```

```
syscfg_flag_clear(SYSCFG_FLAG_SRAM_PCEF);
```

## 3.25. 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 (TIMER1, TIMER2), general level1 timer (TIMER8, TIMER11), general level3 timer (TIMER14, TIMER40), Basic timer (TIMER5, TIMER6). The specific functions of different types of timer are different. The TIMER registers are listed in chapter [3.25.1](#), the TIMER firmware functions are introduced in chapter [3.25.2](#).

### 3.25.1. Descriptions of Peripheral registers

TIMERx registers are listed in the table shown as below:

**Table 3-749. TIMERx Registers**

Registers	Descriptions
TIMER_CTL0(timerx, x=0,1, 2, 5, 6, 8, 11, 14, 40)	Control register 0
TIMERx_CTL1(timerx, x=0,1, 2, 5, 6, 14, 40)	Control register 1
TIMERx_SMCFG(timerx, x=0, 1, 2, 8, 11, 14, 40 )	Slave mode configuration register
TIMERx_DMAINTEN(timerx, x=0,1, 2, 5, 6, 8, 11, 14, 40)	DMA and interrupt enable register
TIMERx_INTF(timerx, x=0,1, 2, 5, 6, 8, 11, 14, 40)	Interrupt flag register
TIMERx_SWEVG(timerx, x=0,1, 2, 5, 6, 8, 11, 14, 40)	Software event generation register
TIMERx_CHCTL0(timerx, x=0, 1, 2, 8, 11 14, 40)	Channel control register 0
TIMERx_CHCTL1(timerx, x=0, 1, 2)	Channel control register 1
TIMERx_CHCTL2(timerx, x=0, 1, 2, 8, 11. 14, 40)	Channel control register 2
TIMERx_CNT(timerx, x=0, 1, 2, 5, 6, 8, 11 14, 40)	Counter register
TIMERx_PSC(timerx, x=0, 1, 2, 5, 6, 8, 11 14, 40)	Prescaler register
TIMERx_CAR(timerx, x=0, 1, 2, 5, 6, 8, 11 14, 40)	Counter auto reload register
TIMERx_CREP(timerx, x=0, 14, 40)	Counter repetition register
TIMERx_CH0CV(timerx, x=0, 1, 2, 8, 11 14, 40)	Channel 0 capture/compare value register
TIMERx_CH1CV(timerx, x=0, 1, 2, 8, 11 14, 40)	Channel 1 capture/compare value register
TIMERx_CH2CV(timerx, x=0, 1, 2)	Channel 2 capture/compare value register
TIMERx_CH3CV(timerx, x=0, 1, 2)	Channel 3 capture/compare value register
TIMERx_IRMP(timerx, x= 8, 11)	Channel complementary protection register
TIMERx_CCHP(timerx, x=0, 14, 40)	TIMER complementary channel protection register
TIMERx_DMACFG(timerx, x=0, 1, 2, 14 ,40)	DMA configuration register
TIMERx_DMATB(timerx, x=0, 1, 2, 14, 40)	DMA transfer buffer register
TIMERx_CFG(timerx, x=0, 1, 2, 8, 11, 14, 40)	Configuration register

### 3.25.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

**Table 3-750. 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 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

Function name	Function description
timer_channel_control_shadow_update_config	configure TIMER channel control shadow register update control
timer_channel_output_struct_para_init	initialize the parameters of TIMER channel output parameter struct with the default values
timer_channel_output_config	configure TIMER channel output function
timer_channel_output_mode_config	configure TIMER channel output compare mode
timer_channel_output_pulse_value_config	configure TIMER channel output pulse value
timer_channel_output_shadow_config	configure TIMER channel output shadow function
timer_channel_output_fast_config	configure TIMER channel output fast function
timer_channel_output_clear_config	configure TIMER channel output clear function
timer_channel_output_polarity_config	configure TIMER channel output polarity
timer_channel_complementary_output_polarity_config	configure TIMER channel complementary output polarity
timer_channel_output_state_config	configure TIMER channel enable state
timer_channel_complementary_output_state_config	configure TIMER channel complementary output enable state
timer_channel_input_struct_para_init	initialize the parameters of TIMER channel input parameter struct with the default values
timer_input_capture_config	configure TIMER input capture parameter
timer_channel_input_capture_prescaler_config	configure TIMER channel input capture prescaler value
timer_channel_capture_value_register_read	read TIMER channel capture compare register value
timer_input_pwm_capture_config	configure TIMER input pwm capture function
timer_hall_mode_config	configure TIMER hall sensor mode
timer_input_trigger_source_select	select TIMER input trigger source
timer_master_output_trigger_source_select	select TIMER master mode output trigger source
timer_slave_mode_select	select TIMER slave mode
timer_master_slave_mode_config	configure TIMER master slave mode
timer_external_trigger_config	configure TIMER external trigger input
timer_quadrature_decoder_mode_config	configure TIMER quadrature decoder mode
timer_internal_trigger_as_external_clock_config	configure TIMER the internal trigger as external clock input
timer_external_trigger_as_external_clock_config	configure TIMER the external trigger as external clock input
timer_external_clock_mode0_config	configure TIMER the external clock mode 0
timer_external_clock_mode1_config	configure TIMER the external clock mode 1

Function name	Function description
timer_external_clock_mode1_disable	disable TIMER the external clock mode 1
timer_channel_remap_config	configure TIMER channel remap function
timer_write_chxval_register_config	configure TIMER write CHxVAL register selection
timer_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 3-751. 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 (only for GD32L235)

**Table 3-752. Structure timer\_break\_parameter\_struct**

Member name	Function description
runoffstate	run mode off-state(TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE)
ideloffstate	idle mode off-state(TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE)
deadtime	dead time(0~255)
breakpolarity	break polarity(TIMER_BREAK_POLARITY_LOW, TIMER_BREAK_POLARITY_HIGH)
outputautostate	output automatic enable (TIMER_OUTAUTO_ENABLE, TIMER_OUTAUTO_DISABLE)
protectmode	complementary register protect control(TIMER_CCHP_PROT_OFF, TIMER_CCHP_PROT_0, TIMER_CCHP_PROT_1, TIMER_CCHP_PROT_2)
breakstate	break enable(TIMER_BREAK_ENABLE, TIMER_BREAK_DISABLE)



## Structure timer\_oc\_parameter\_struct

**Table 3-753. 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 3-754. Structure timer\_ic\_parameter\_struct**

Member name	Function description
icpolarity	channel input polarity (TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_FALLING, TIMER_IC_POLARITY_BOTH_EDGE)
icselection	channel input mode selection (TIMER_IC_SELECTION_DIRECTTI, TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS)
icprescaler	channel input capture prescaler (TIMER_IC_PSC_DIV1, TIMER_IC_PSC_DIV2, TIMER_IC_PSC_DIV4, TIMER_IC_PSC_DIV8)
icfilter	channel input capture filter control (0~15)

## timer\_deinit

The description of timer\_deinit is shown as below:

**Table 3-755. Function timer\_deinit**

Function name	timer_deinit
Function prototype	void timer_deinit(uint32_t timer_periph);
Function descriptions	deinit a TIMER
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
timer_periph	TIMER peripheral

<i>TIMERx</i>	TIMER peripheral selection (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset TIMER1 */
```

```
timer_deinit(TIMER1);
```

### timer\_struct\_para\_init

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

**Table 3-756. Function timer\_struct\_para\_init**

<b>Function name</b>	timer_struct_para_init
<b>Function prototype</b>	void timer_struct_para_init(timer_parameter_struct* initpara);
<b>Function descriptions</b>	initialize the parameters of TIMER init parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
initpara	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-751. Structure timer parameter struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize TIMER init parameter struct with a default value */
```

```
timer_parameter_struct timer_initpara;
```

```
timer_struct_para_init(timer_initpara);
```

### timer\_init

The description of timer\_init is shown as below:

**Table 3-757. Function timer\_init**

<b>Function name</b>	timer_init
<b>Function prototype</b>	void timer_init(uint32_t timer_periph, timer_parameter_struct* initpara);
<b>Function descriptions</b>	initialize TIMER counter

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<b>Input parameter{in}</b>	
<b>initpara</b>	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-751. Structure timer parameter struct.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* initialize TIMER1 */

timer_parameter_struct timer_initpara;

timer_initpara.prescaler      = 63;

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 3-758. 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=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* enable TIMER1 */
timer_enable(TIMER1);
```

### timer\_disable

The description of timer\_disable is shown as below:

**Table 3-759. 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=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER1 */
timer_disable(TIMER1);
```

### timer\_auto\_reload\_shadow\_enable

The description of timer\_auto\_reload\_shadow\_enable is shown as below:

**Table 3-760. 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=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the TIMER1 auto reload shadow function */
```

```
timer_auto_reload_shadow_enable(TIMER1);
```

### timer\_auto\_reload\_shadow\_disable

The description of timer\_auto\_reload\_shadow\_disable is shown as below:

**Table 3-761. 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=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the TIMER1 auto reload shadow function */
```

```
timer_auto_reload_shadow_disable(TIMER1);
```

### timer\_update\_event\_enable

The description of timer\_update\_event\_enable is shown as below:

**Table 3-762. 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>	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TIMER1 the update event */
```

```
timer_update_event_enable(TIMER1);
```

### timer\_update\_event\_disable

The description of timer\_update\_event\_disable is shown as below:

**Table 3-763. 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>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER1 the update event */
```

```
timer_update_event_disable(TIMER1);
```

### timer\_counter\_alignment

The description of timer\_counter\_alignment is shown as below:

**Table 3-764. Function timer\_counter\_alignment**

<b>Function name</b>	timer_counter_alignment
<b>Function prototype</b>	void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);

<b>Function descriptions</b>	set TIMER counter alignment mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<b>Input parameter{in}</b>	
<b>aligned</b>	alignment mode
<i>TIMER_COUNTER_EDGE</i>	No center-aligned mode (edge-aligned mode). The direction of the counter is specified by the DIR bit.
<i>TIMER_COUNTER_COUNTER_DOWN</i>	Center-aligned and counting down assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in <i>TIMERx_CHCTL0</i> register). Only when the counter is counting down, compare interrupt flag of channels can be set.
<i>TIMER_COUNTER_COUNTER_UP</i>	Center-aligned and counting up assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in <i>TIMERx_CHCTL0</i> register). Only when the counter is counting up, compare interrupt flag of channels can be set.
<i>TIMER_COUNTER_COUNTER_BOTH</i>	Center-aligned and counting up/down assert mode. The counter counts under center-aligned and channel is configured in output mode (CHxMS=00 in <i>TIMERx_CHCTL0</i> register). Both when the counter is counting up and counting down, compare interrupt flag of channels can be set.
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set TIMER1 counter center-aligned and counting up assert mode */
timer_counter_alignment(TIMER1, TIMER_COUNTER_COUNTER_UP);
```

### timer\_counter\_up\_direction

The description of timer\_counter\_up\_direction is shown as below:

**Table 3-765. 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=1,2 for GD32L233, x=0,1,2 for GD32L235)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set TIMER1 counter up direction */
timer_counter_up_direction(TIMER1);
```

### timer\_counter\_down\_direction

The description of timer\_counter\_down\_direction is shown as below:

**Table 3-766. 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=1,2 for GD32L233, x=0,1,2 for GD32L235)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set TIMER1 counter down direction */
timer_counter_down_direction(TIMER1);
```

### timer\_prescaler\_config

The description of timer\_prescaler\_config is shown as below:

**Table 3-767. Function timer\_prescaler\_config**

<b>Function name</b>	timer_prescaler_config
<b>Function prototype</b>	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint8_t pscreload);
<b>Function descriptions</b>	configure TIMER prescaler



<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<b>Input parameter{in}</b>	
<b>prescaler</b>	prescaler value (0~65535)
<b>Input parameter{in}</b>	
<b>pscreload</b>	prescaler reload mode
<i>TIMER_PSC_RELOAD_NOW</i>	the prescaler is loaded right now
<i>TIMER_PSC_RELOAD_UPDATE</i>	the prescaler is loaded at the next update event
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER1 prescaler */
```

```
timer_prescaler_config(TIMER1, 3000, TIMER_PSC_RELOAD_NOW);
```

### timer\_repetition\_value\_config

The description of timer\_repetition\_value\_config is shown as below:

**Table 3-768. Function timer\_repetition\_value\_config**

<b>Function name</b>	timer_repetition_value_config
<b>Function prototype</b>	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
<b>Function descriptions</b>	configure TIMER repetition register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection (x=0,14,40 for GD32L235)
<b>Input parameter{in}</b>	
<b>repetition</b>	the counter repetition value (0~255)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* configure TIMER0 repetition register value */
```

```
timer_repetition_value_config (TIMER0, 98);
```

### timer\_autoreload\_value\_config

The description of timer\_autoreload\_value\_config is shown as below:

**Table 3-769. Function timer\_autoreload\_value\_config**

<b>Function name</b>	timer_autoreload_value_config
<b>Function prototype</b>	void timer_autoreload_value_config(uint32_t timer_periph, uint16_t autoreload);
<b>Function descriptions</b>	configure TIMER autoreload register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<b>Input parameter{in}</b>	
<b>autoreload</b>	the counter auto-reload value (0-65535)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER1 autoreload register value */
```

```
timer_autoreload_value_config(TIMER1, 3000);
```

### timer\_counter\_value\_config

The description of timer\_counter\_value\_config is shown as below:

**Table 3-770. 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=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<b>Input parameter{in}</b>	
<b>counter</b>	the counter value(0-65535)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER1 counter register value */
```

```
timer_counter_value_config(TIMER1, 3000);
```

### timer\_counter\_read

The description of timer\_counter\_read is shown as below:

**Table 3-771. 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=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	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 3-772. 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=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
Output parameter{out}	
-	-
Return value	
uint16_t	prescaler register value (0~65535)

Example:

```
/* read TIMER1 prescaler value */
```

```
uint16_t i = 0;
```

```
i = timer_prescaler_read(TIMER1);
```

### timer\_single\_pulse\_mode\_config

The description of timer\_single\_pulse\_mode\_config is shown as below:

Table 3-773. Function timer\_single\_pulse\_mode\_config

Function name	timer_single_pulse_mode_config
Function prototype	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);
Function descriptions	configure TIMER single pulse mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	TIMER peripheral selection (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
Input parameter{in}	
spmode	pulse mode
TIMER_SP_MODE_SINGLE	single pulse mode
TIMER_SP_MODE_REPETITIVE	repetitive pulse mode
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure TIMER1 single pulse mode */
```

```
timer_single_pulse_mode_config(TIMER1, TIMER_SP_MODE_SINGLE);
```

### timer\_update\_source\_config

The description of timer\_update\_source\_config is shown as below:

**Table 3-774. Function timer\_update\_source\_config**

Function name	timer_update_source_config
Function prototype	void timer_update_source_config(uint32_t timer_periph, uint32_t update);
Function descriptions	configure TIMER update source
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
Input parameter{in}	
update	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 TIMER1 update only by counter overflow/underflow */
```

```
timer_update_source_config(TIMER1, TIMER_UPDATE_SRC_REGULAR);
```

### timer\_dma\_enable

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

**Table 3-775. Function timer\_dma\_enable**

Function name	timer_dma_enable
---------------	------------------

<b>Function prototype</b>	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
<b>Function descriptions</b>	enable the TIMER DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>dma</b>	timer DMA source enable
<i>TIMER_DMA_UPD</i>	update DMA enable TIMERx(x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA enable TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA enable TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_DMA_CMTD</i>	commutation DMA request enable TIMERx(x=0,14,40 for GD32L235)
<i>TIMER_DMA_TRGD</i>	trigger DMA enable TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the TIMER1 update DMA */
```

```
timer_dma_enable(TIMER1, TIMER_DMA_UPD);
```

## timer\_dma\_disable

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

**Table 3-776. 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>	-

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 disable
<i>TIMER_DMA_UPD</i>	update DMA enable TIMERx(x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA enable TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA enable TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_DMA_CMTD</i>	commutation DMA request enable TIMERx(x=0,14,40 for GD32L235)
<i>TIMER_DMA_TRGD</i>	trigger DMA enable TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER1 update DMA */
```

```
timer_dma_disable(TIMER1, TIMER_DMA_UPD);
```

### timer\_channel\_dma\_request\_source\_select

The description of timer\_channel\_dma\_request\_source\_select is shown as below:

**Table 3-777. Function timer\_channel\_dma\_request\_source\_select**

<b>Function name</b>	timer_channel_dma_request_source_select
<b>Function prototype</b>	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
<b>Function descriptions</b>	channel DMA request source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection

	(x=1,2 for GD32L233, x=0,1,2,14,40 for GD32L235)
<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:

```
/* TIMER1 channel DMA request of channel n is sent when channel y event occurs */
```

```
timer_channel_dma_request_source_select(TIMER1,  
TIMER_DMAREQUEST_CHANNELEVENT);
```

### timer\_dma\_transfer\_config

The description of timer\_dma\_transfer\_config is shown as below:

**Table 3-778. 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=1,2 for GD32L233, x=0,1,2,14,40 for GD32L235)
<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</i>	DMA transfer address is TIMER_INTF



<i>TA_INTF</i>	
<i>TIMER_DMACFG_DMA</i> <i>TA_SWEVG</i>	DMA transfer address is <i>TIMER_SWEVG</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CHCTL0</i>	DMA transfer address is <i>TIMER_CHCTL0</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CHCTL1</i>	DMA transfer address is <i>TIMER_CHCTL1</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CHCTL2</i>	DMA transfer address is <i>TIMER_CHCTL2</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CNT</i>	DMA transfer address is <i>TIMER_CNT</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_PSC</i>	DMA transfer address is <i>TIMER_PSC</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CAR</i>	MA transfer address is <i>TIMER_CAR</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CH0CV</i>	DMA transfer address is <i>TIMER_CH0CV</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CH1CV</i>	DMA transfer address is <i>TIMER_CH1CV</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CH2CV</i>	DMA transfer address is <i>TIMER_CH2CV</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CH3CV</i>	DMA transfer address is <i>TIMER_CH3CV</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CCHP</i>	DMA transfer address is <i>TIMER_CCHP</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_DMACFG</i>	DMA transfer address is <i>TIMER_DMACFG</i>
<b>Input parameter{in}</b>	
<b>dma_lenth</b>	DMA transfer count
<i>TIMER_DMACFG_DMA</i> <i>TC_xTRANSFER</i>	x=1..18, DMA transfer x time
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the TIMER1 DMA transfer */
```

```
timer_dma_transfer_config(TIMER1, TIMER_DMACFG_DMATA_CTL0,  
TIMER_DMACFG_DMATC_5TRANSFER);
```

## timer\_event\_software\_generate

The description of timer\_event\_software\_generate is shown as below:

**Table 3-779. 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=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<i>TIMER_EVENT_SRC_C0G</i>	channel 0 capture or compare event generation TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_EVENT_SRC_C1G</i>	channel 1 capture or compare event generation TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_EVENT_SRC_C2G</i>	channel 2 capture or compare event generation TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_EVENT_SRC_C3G</i>	channel 3 capture or compare event generation TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_EVENT_SRC_CMTG</i>	channel commutation event generation TIMERx(x=0,14,40 for GD32L235)
<i>TIMER_EVENT_SRC_TRG</i>	trigger event generation TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_EVENT_SRC_BRK</i>	break event generation TIMERx(x=0,14,40 for GD32L235)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* software generate update event*/
```

```
timer_event_software_generate(TIMER1, TIMER_EVENT_SRC_UPG);
```

### timer\_break\_struct\_para\_init (only for GD32L235)

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

**Table 3-780. 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 (only for GD32L235)

The description of timer\_break\_config is shown as below:

**Table 3-781. 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, 14, 40)</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>	

-	-
Return value	
-	-

Example:

```
/* configure TIMER0 break function */

timer_break_parameter_struct timer_breakpara;

timer_breakpara.runoffstate      = TIMER_ROS_STATE_DISABLE;
timer_breakpara.ideloffstate     = TIMER_IOS_STATE_DISABLE ;
timer_breakpara.deadtime         = 255;
timer_breakpara.breakpolarity    = TIMER_BREAK_POLARITY_LOW;
timer_breakpara.outputautostate  = TIMER_OUTAUTO_ENABLE;
timer_breakpara.protectmode      = TIMER_CCHP_PROT_0;
timer_breakpara.breakstate       = TIMER_BREAK_ENABLE;

timer_break_config(TIMER0, &timer_breakpara);
```

### timer\_break\_enable (only for GD32L235)

The description of timer\_break\_enable is shown as below:

**Table 3-782. Function timer\_break\_enable**

<b>Function name</b>	timer_break_enable
<b>Function prototype</b>	void timer_break_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable TIMER break function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14, 40)</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 (only for GD32L235)

The description of timer\_break\_disable is shown as below:

**Table 3-783. Function timer\_break\_disable**

<b>Function name</b>	timer_break_disable
<b>Function prototype</b>	void timer_break_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER break function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14, 40)</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 (only for GD32L235)

The description of timer\_automatic\_output\_enable is shown as below:

**Table 3-784. 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, 14, 40)</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 (only for GD32L235)

The description of timer\_automatic\_output\_disable is shown as below:

**Table 3-785. Function timer\_automatic\_output\_disable**

<b>Function name</b>	timer_automatic_output_disable
<b>Function prototype</b>	void timer_automatic_output_disable (uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER output automatic function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14, 40)</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 (only for GD32L235)

The description of timer\_primary\_output\_config is shown as below:

**Table 3-786. Function timer\_primary\_output\_config**

<b>Function name</b>	timer_primary_output_config
<b>Function prototype</b>	void timer_primary_output_config(uint32_t timer_periph, ControlStatus newvalue);
<b>Function descriptions</b>	configure TIMER primary output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14, 40)</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>	

-	-
Return value	
-	-

Example:

```
/* enable TIMER0 primary output function */
```

```
timer_primary_output_config (TIMER0, ENABLE);
```

### timer\_channel\_control\_shadow\_config (only for GD32L235)

The description of timer\_channel\_control\_shadow\_config is shown as below:

**Table 3-787. Function timer\_channel\_control\_shadow\_config**

<b>Function name</b>	timer_channel_control_shadow_config
<b>Function prototype</b>	void timer_channel_control_shadow_config(uint32_t timer_periph, ControlStatus newvalue);
<b>Function descriptions</b>	channel commutation control shadow register enable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14, 40)</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 (only for GD32L235)

The description of timer\_channel\_control\_shadow\_update\_config is shown as below:

**Table 3-788. Function timer\_channel\_control\_shadow\_update\_config**

<b>Function name</b>	timer_channel_control_shadow_update_config
<b>Function prototype</b>	void timer_channel_control_shadow_update_config(uint32_t timer_periph, uint8_t ccuctl);

<b>Function descriptions</b>	configure commutation control shadow register update control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14, 40)</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 3-789. 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>ocpa</b>	TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-753. 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 3-790. Function timer\_channel\_output\_config**

Function name	timer_channel_output_config
Function prototype	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara);
Function descriptions	configure TIMER channel output function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMEx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0 TIMEx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_CH_1	TIMER channel 1 TIMEx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_CH_2	TIMER channel 2 TIMEx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_CH_3	TIMER channel 3 TIMEx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
Input parameter{in}	
ocpara	TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-753. Structure timer_oc_parameter_struct</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER1 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(TIMER1, TIMER_CH_0, &timer_ocinitpara);
```

## timer\_channel\_output\_mode\_config

The description of timer\_channel\_output\_mode\_config is shown as below:

**Table 3-791. 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=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_1</i>	TIMER channel 1 TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_2</i>	TIMER channel 2 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_3</i>	IMER channel 3 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<b>Input parameter{in}</b>	
<b>ocmode</b>	channel output compare mode
<i>TIMER_OC_MODE_TIMING</i>	timing mode
<i>TIMER_OC_MODE_ACTIVE</i>	set the channel output
<i>TIMER_OC_MODE_INACTIVE</i>	clear the channel output
<i>TIMER_OC_MODE_TOGGLE</i>	toggle on match
<i>TIMER_OC_MODE_LOW</i>	force low mode
<i>TIMER_OC_MODE_HIGH</i>	force high mode
<i>TIMER_OC_MODE_PWM0</i>	PWM mode 0
<i>TIMER_OC_MODE_PWM1</i>	PWM mode 1
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* configure TIMER1 channel PWM 0 mode */
```

```
timer_channel_output_mode_config(TIMER1, TIMER_CH_0, TIMER_OC_MODE_PWM0);
```

### timer\_channel\_output\_pulse\_value\_config

The description of timer\_channel\_output\_pulse\_value\_config is shown as below:

**Table 3-792. 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=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_1</i>	TIMER channel 1 TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_2</i>	TIMER channel 2 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_3</i>	TIMER channel 3 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<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 TIMER1 channel 0 output pulse value */
```

```
timer_channel_output_pulse_value_config(TIMER1, TIMER_CH_0, 399);
```

## timer\_channel\_output\_shadow\_config

The description of timer\_channel\_output\_shadow\_config is shown as below:

**Table 3-793. 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=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_1</i>	TIMER channel 1 TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_2</i>	TIMER channel 2 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_3</i>	TIMER channel 3 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<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 TIMER1 channel 0 output shadow function */
```

```
timer_channel_output_shadow_config(TIMER1, 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 3-794. 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=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_1</i>	TIMER channel 1 TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_2</i>	TIMER channel 2 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_3</i>	TIMER channel 3 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<b>Input parameter{in}</b>	
<b>ocfast</b>	channel output fast function
<i>TIMER_OC_FAST_ENABLE</i>	channel output fast function enable
<i>TIMER_OC_FAST_DISABLE</i>	channel output fast function disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER1 channel 0 output fast function */
```

```
timer_channel_output_fast_config(TIMER1, 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 3-795. Function timer\_channel\_output\_clear\_config

<b>Function name</b>	timer_channel_output_clear_config
<b>Function prototype</b>	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);

<b>Function descriptions</b>	configure TIMER channel output clear function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER periphera
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_1</i>	TIMER channel 1 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_2</i>	TIMER channel 2 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_3</i>	TIMER channel 3 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<b>Input parameter{in}</b>	
<b>occlear</b>	channel output clear function
<i>TIMER_OC_CLEAR_ENABLE</i>	channel output clear function enable
<i>TIMER_OC_CLEAR_DISABLE</i>	channel output clear function disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER1 channel 0 output clear function */
```

```
timer_channel_output_clear_config(TIMER1, 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 3-796. 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>	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_1</i>	TIMER channel 1 TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_2</i>	TIMER channel 2 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_3</i>	TIMER channel 3 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
Input parameter{in}	
<b>ocpolarity</b>	channel output polarity
<i>TIMER_OC_POLARITY_HIGH</i>	channel output polarity is high
<i>TIMER_OC_POLARITY_LOW</i>	channel output polarity is low
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER1 channel 0 output polarity */
```

```
timer_channel_output_polarity_config(TIMER1, TIMER_CH_0,  
TIMER_OC_POLARITY_HIGH);
```

### timer\_channel\_complementary\_output\_polarity\_config (only for GD32L235)

The description of timer\_channel\_complementary\_output\_polarity\_config is shown as below:

**Table 3-797. Function timer\_channel\_complementary\_output\_polarity\_config**

<b>Function name</b>	timer_channel_complementary_output_polarity_config
<b>Function prototype</b>	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
<b>Function descriptions</b>	configure TIMER channel complementary output polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection

Input parameter{in}	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 $TIMERx(x=0, 14, 40)$
<i>TIMER_CH_1</i>	TIMER channel 1 $TIMERx(x=0)$
<i>TIMER_CH_2</i>	TIMER channel 2 $TIMERx(x=0)$
Input parameter{in}	
<b>ocpolarity</b>	channel complementary output polarity
<i>TIMER_OCN_POLARITY_HIGH</i>	channel complementary output polarity is high
<i>TIMER_OCN_POLARITY_LOW</i>	channel complementary output polarity is low
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 complementary output polarity */
```

```
timer_channel_complementary_output_polarity_config (TIMER0, TIMER_CH_0,  
TIMER_OCN_POLARITY_HIGH);
```

### timer\_channel\_output\_state\_config

The description of timer\_channel\_output\_state\_config is shown as below:

**Table 3-798. 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>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 $TIMERx(x=1,2,8,11 \text{ for GD32L233}, x=0,1,2,8,11,14,40 \text{ for GD32L235})$
<i>TIMER_CH_1</i>	TIMER channel 1 $TIMERx(x=1,2,8,11 \text{ for GD32L233}, x=0,1,2,8,11,14,40 \text{ for GD32L235})$
<i>TIMER_CH_2</i>	TIMER channel 2 $TIMERx(x=1,2 \text{ for GD32L233}, x=0,1,2 \text{ for GD32L235})$



<i>TIMER_CH_3</i>	TIMER channel 3 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<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 TIMER1 channel 0 enable state */
```

```
timer_channel_output_state_config(TIMER1, TIMER_CH_0, TIMER_CCX_ENABLE);
```

### timer\_channel\_complementary\_output\_state\_config (only for GD32L235)

The description of timer\_channel\_complementary\_output\_state\_config is shown as below:

**Table 3-799. 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 TIMERx(x=0, 14, 40)
<i>TIMER_CH_1</i>	TIMER channel 1 TIMERx(x=0)
<i>TIMER_CH_2</i>	TIMER channel 2 TIMERx(x=0)
<b>Input parameter{in}</b>	
<b>state</b>	TIMER channel complementary output enable state
<i>TIMER_CCXN_ENABLE</i>	channel complementary enable
<i>TIMER_CCXN_DISABLE</i>	channel complementary disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 complementary output enable state */

timer_channel_complementary_output_state_config (TIMER0, TIMER_CH_0,
TIMER_CCXN_ENABLE);
```

### timer\_channel\_input\_struct\_para\_init

The description of timer\_channel\_input\_struct\_para\_init is shown as below:

**Table 3-800. Function timer\_channel\_input\_struct\_para\_init**

<b>Function name</b>	timer_channel_input_struct_para_init
<b>Function prototype</b>	void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);
<b>Function descriptions</b>	initialize the parameters of TIMER channel input parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>icpara</b>	TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-754. Structure timer_ic_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize TIMER channel input parameter struct with a default value */

timer_ic_parameter_struct timer_icinitpara;

timer_channel_input_struct_para_init(&timer_icinitpara);
```

### timer\_input\_capture\_config

The description of timer\_input\_capture\_config is shown as below:

**Table 3-801. Function timer\_input\_capture\_config**

<b>Function name</b>	timer_input_capture_config
<b>Function prototype</b>	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);
<b>Function descriptions</b>	configure TIMER input capture parameter
<b>Precondition</b>	-
<b>The called functions</b>	timer_channel_input_capture_prescaler_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral

<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_1</i>	TIMER channel 1 TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_2</i>	TIMER channel 2 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_3</i>	TIMER channel 3 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<b>Input parameter{in}</b>	
<b>icpara</b>	TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-754. Structure timer_ic_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure TIMER1 input capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_capture_config(TIMER1, 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 3-802. 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=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_1</i>	TIMER channel 1 TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_2</i>	TIMER channel 2 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_3</i>	TIMER channel 3 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<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 TIMER1 channel 0 input capture prescaler value */
```

```
timer_channel_input_capture_prescaler_config(TIMER1, 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 3-803. 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=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_1</i>	TIMER channel 1 TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_2</i>	TIMER channel 2 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_3</i>	TIMER channel 3 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<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 3-804. 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 x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235
<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 3-754. Structure timer_ic_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```

/* configure TIMER1 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(TIMER1, TIMER_CH_0, &timer_icinitpara);

```

### timer\_hall\_mode\_config

The description of timer\_hall\_mode\_config is shown as below:

**Table 3-805. 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 x=1,2 for GD32L233, x=0,1,2 for GD32L235
<b>Input parameter{in}</b>	
<b>hallmode</b>	TIMER hall sensor mode state
<i>TIMER_HALLINTERFACE_ENABLE</i>	TIMER hall sensor mode enable
<i>TIMER_HALLINTERFACE_DISABLE</i>	TIMER hall sensor mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure TIMER1 hall sensor mode */

timer_hall_mode_config(TIMER1, TIMER_HALLINTERFACE_ENABLE);

```

## timer\_input\_trigger\_source\_select

The description of timer\_input\_trigger\_source\_select is shown as below:

**Table 3-806. 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 TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI1</i>	Internal trigger input 1 TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI2</i>	Internal trigger input 2 TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI3</i>	Internal trigger input 3 TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOF_ED</i>	CIO edge flag TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOFE0</i>	channel 0 input Filtered output TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SMCFG_TRGS</i> <i>EL_CI1FE1</i>	channel 1 input Filtered output TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ETIFP</i>	External trigger input filter output TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select TIMER1 input trigger source */
```

```
timer_input_trigger_source_select(TIMER1, TIMER_SMCFG_TRGSEL_ITI0);
```

## timer\_master\_output\_trigger\_source\_select

The description of timer\_master\_output\_trigger\_source\_select is shown as below:

**Table 3-807. 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_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 TIMER1 master mode output trigger source */
```

```
timer_master_output_trigger_source_select(TIMER1, TIMER_TRI_OUT_SRC_RESET);
```

### timer\_slave\_mode\_select

The description of timer\_slave\_mode\_select is shown as below:

**Table 3-808. 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 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_QUAD_DECODER_MODE1</i>	quadrature decoder mode 1 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_QUAD_DECODER_MODE2</i>	quadrature decoder mode 2 TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_SLAVE_MODE_RESTART</i>	restart mode TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SLAVE_MODE_PAUSE</i>	pause mode TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SLAVE_MODE_EVENT</i>	event mode TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SLAVE_MODE_EXTERNAL0</i>	external clock mode 0 (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select TIMER1 slave mode */
```

```
timer_slave_mode_select(TIMER1, TIMER_QUAD_DECODER_MODE0);
```

## timer\_master\_slave\_mode\_config

The description of timer\_master\_slave\_mode\_config is shown as below:

**Table 3-809. Function timer\_master\_slave\_mode\_config**

<b>Function name</b>	timer_master_slave_mode_config
<b>Function prototype</b>	void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t masterslave);
<b>Function descriptions</b>	configure TIMER master slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235
<b>Input parameter{in}</b>	
<b>masterslave</b>	master slave mode state
<i>TIMER_MASTER_SLAVE_MODE_ENABLE</i>	master slave mode enable
<i>TIMER_MASTER_SLAVE_MODE_DISABLE</i>	master slave mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER1 master slave mode */
timer_master_slave_mode_config(TIMER1, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

## timer\_external\_trigger\_config

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

**Table 3-810. 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

	x=1,2 for GD32L233, x=0,1,2 for GD32L235
<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 TIMER1 external trigger input */
```

```
timer_external_trigger_config(TIMER1,                                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 3-811. Function timer\_quadrature\_decoder\_mode\_config**

<b>Function name</b>	timer_quadrature_decoder_mode_config
<b>Function prototype</b>	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decompode, uint16_t ic0polarity, uint16_t ic1polarity);
<b>Function descriptions</b>	configure TIMER quadrature decoder mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection x=1,2 for GD32L233, x=0,1,2 for GD32L235
<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 TIMER1 quadrature decoder mode */
```

```
timer_quadrature_decoder_mode_config(TIMER1, 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 3-812. Function timer\_internal\_clock\_config**

<b>Function name</b>	timer_internal_clock_config
<b>Function prototype</b>	void timer_internal_clock_config(uint32_t timer_periph);
<b>Function descriptions</b>	configure TIMER internal clock mode
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 internal clock mode */
```

```
timer_internal_clock_config (TIMER0);
```

### timer\_internal\_trigger\_as\_external\_clock\_config

The description of timer\_internal\_trigger\_as\_external\_clock\_config is shown as below:

**Table 3-813. Function timer\_internal\_trigger\_as\_external\_clock\_config**

<b>Function name</b>	timer_internal_trigger_as_external_clock_config
<b>Function prototype</b>	void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);
<b>Function descriptions</b>	configure TIMER the internal trigger as external clock input
<b>Precondition</b>	-
<b>The called functions</b>	timer_input_trigger_source_select
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection
Input parameter{in}	
<b>intrigger</b>	trigger selection
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI0</i>	Internal trigger input 0 (ITI0) TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI1</i>	Internal trigger input 1 (ITI1) TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI2</i>	Internal trigger input 2 (ITI2) TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI3</i>	Internal trigger input 3 (ITI3) TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER1 the internal trigger ITIO as external clock input */
```

```
timer_internal_trigger_as_external_clock_config(TIMER1, TIMER_SMCFG_TRGSEL_ITIO);
```

## timer\_external\_trigger\_as\_external\_clock\_config

The description of timer\_external\_trigger\_as\_external\_clock\_config is shown as below:

**Table 3-814. Function timer\_external\_trigger\_as\_external\_clock\_config**

<b>Function name</b>	timer_external_trigger_as_external_clock_config
<b>Function prototype</b>	void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t extrigger, uint16_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external trigger as external clock input
<b>Precondition</b>	-
<b>The called functions</b>	timer_input_trigger_source_select
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235
<b>Input parameter{in}</b>	
<b>extrigger</b>	external trigger selection
<i>TIMER_SMCFG_TRGS_EL_CIOF_ED</i>	CIO edge flag (CIOF_ED)
<i>TIMER_SMCFG_TRGS_EL_CIOFE0</i>	channel 0 input Filtered output (CIOFE0)
<i>TIMER_SMCFG_TRGS_EL_C1FE1</i>	channel 1 input Filtered output (C11FE1)
<b>Input parameter{in}</b>	
<b>expolarity</b>	external trigger polarity
<i>TIMER_IC_POLARITY_RISING</i>	active high or rising edge active
<i>TIMER_IC_POLARITY_FALLING</i>	active low or falling edge active
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	falling edge or rising edge active
<b>Input parameter{in}</b>	
<b>extfilter</b>	external trigger filter control(0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER1 the external trigger CIOFE0 as external clock input */
```

```
timer_external_trigger_as_external_clock_config(TIMER1,
TIMER_SMCFG_TRGSEL_CIOFE0, TIMER_IC_POLARITY_RISING, 0);
```

### timer\_external\_clock\_mode0\_config

The description of timer\_external\_clock\_mode0\_config is shown as below:

**Table 3-815. 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 x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235
<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 TIMER1 the external clock mode0 */

timer_external_clock_mode0_config(TIMER1,                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 3-816. Function timer\_external\_clock\_mode1\_config**

<b>Function name</b>	timer_external_clock_mode1_config
<b>Function prototype</b>	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external clock mode1
<b>Precondition</b>	-
<b>The called functions</b>	timer_external_trigger_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection x=1,2 for GD32L233, x=0,1,2 for GD32L235
<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 TIMER1 the external clock mode1 */
```

```
timer_external_clock_mode1_config(TIMER1,                TIMER_EXT_TRI_PSC_DIV2,
timer_external_clock_mode1_config(TIMER1,                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 3-817. 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 x=1,2 for GD32L233, x=0,1,2 for GD32L235
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER1 the external clock mode1 */
```

```
timer_external_clock_mode1_disable(TIMER1);
```

## timer\_channel\_remap\_config

The description of timer\_channel\_remap\_config is shown as below:

**Table 3-818. Function timer\_channel\_remap\_config**

<b>Function name</b>	timer_channel_remap_config
<b>Function prototype</b>	void timer_channel_remap_config(uint32_t timer_periph, uint32_t remap);
<b>Function descriptions</b>	configure TIMER channel remap function
<b>Precondition</b>	-
<b>The called functions</b>	timer_external_trigger_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=8, 11)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>remap</b>	remap function selection
<i>TIMER8_CIO_RMP_GPIO</i> <i>O</i>	Timer8 channel 0 input is connected to GPIO(TIMER8_CH0) TIMERx(x=8)
<i>TIMER8_CIO_RMP_LXT</i> <i>AL</i>	Timer8 channel 0 input is connected to the LXTAL TIMERx(x=8)
<i>TIMER8_CIO_RMP_HXT</i> <i>AL_DIV32</i>	Timer8 channel 0 input is connected to HXTAL_DIV32 clock TIMERx(x=8)

<i>TIMER8_CIO_RMP_CK OUTSEL</i>	Timer8 channel 0 input is connected to CKOUTSEL TIMERx(x=8)
<i>TIMER11_CIO_RMP_GP IO</i>	Timer11 channel 0 input is connected to GPIO(TIMER11_CH0) TIMERx(x=11)
<i>TIMER11_CIO_RMP_IR C32K</i>	Timer11 channel 0 input is connected to the IRC32K TIMERx(x=11)
<i>TIMER11_CIO_RMP_LX TAL</i>	Timer11 channel 0 input is connected to LXTAL clock TIMERx(x=11)
<i>TIMER11_CIO_RMP_RT C_OUT</i>	Timer11 channel 0 input is connected to RTC_OUT TIMERx(x=11)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER8 channel 0 input is connected to GPIO */
```

```
timer_channel_remap_config(TIMER8, TIMER8_CIO_RMP_GPIO);
```

### timer\_write\_chxval\_register\_config

The description of timer\_write\_chxval\_register\_config is shown as below:

**Table 3-819. 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 x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235
<b>Input parameter{in}</b>	
<b>ccsel</b>	write CHxVAL register selection
<i>TIMER_CHVSEL_DISA BLE</i>	no effect
<i>TIMER_CHVSEL_ENAB LE</i>	when write the CHxVAL register, if the write value is same as the CHxVAL value, the write access is ignored
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* configure TIMER1 write CHxVAL register selection */
```

```
timer_write_chxval_register_config(TIMER1, TIMER_CHVSEL_ENABLE);
```

### timer\_flag\_get

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

**Table 3-820. Function timer\_flag\_get**

<b>Function name</b>	timer_flag_get
<b>Function prototype</b>	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);
<b>Function descriptions</b>	get TIMER flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>flag</b>	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag TIMERx(x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<i>TIMER_FLAG_CH0</i>	channel 0 flag TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_FLAG_CH1</i>	channel 1 flag TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_FLAG_CH2</i>	channel 2 flag TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_FLAG_CH3</i>	channel 3 flag TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_FLAG_CMT</i>	channel commutation flag TIMERx(x=0,14,40 for GD32L235)
<i>TIMER_FLAG_TRG</i>	trigger flag TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_FLAG_BRK</i>	break flag TIMERx(x=0,14,40 for GD32L235)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag

	TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TIMER1 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get(TIMER1, TIMER_FLAG_UP);
```

### timer\_flag\_clear

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

**Table 3-821. Function timer\_flag\_clear**

<b>Function name</b>	timer_flag_clear
<b>Function prototype</b>	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
<b>Function descriptions</b>	clear TIMER flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>flag</b>	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag TIMERx(x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<i>TIMER_FLAG_CH0</i>	channel 0 flag TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_FLAG_CH1</i>	channel 1 flag TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_FLAG_CH2</i>	channel 2 flag TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_FLAG_CH3</i>	channel 3 flag TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_FLAG_CMT</i>	channel commutation flag TIMERx(x=0,14,40 for GD32L235)
<i>TIMER_FLAG_TRG</i>	trigger flag TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)

<i>TIMER_FLAG_BRK</i>	break flag TIMERx(x=0,14,40 for GD32L235)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TIMER1 update flags */
```

```
timer_flag_clear(TIMER1, TIMER_FLAG_UP);
```

## timer\_interrupt\_enable

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

**Table 3-822. Function timer\_interrupt\_enable**

<b>Function name</b>	timer_interrupt_enable
<b>Function prototype</b>	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable the TIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	timer interrupt enable source
<i>TIMER_INT_UP</i>	update interrupt enable TIMERx(x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<i>TIMER_INT_CH0</i>	channel 0 interrupt enable TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_INT_CH1</i>	channel 1 interrupt enable TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_INT_CH2</i>	channel 2 interrupt enable TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)

<i>TIMER_INT_CH3</i>	channel 3 interrupt enable TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_INT_CMT</i>	commutation interrupt enable TIMERx(x=0,14,40 for GD32L235)
<i>TIMER_INT_TRG</i>	trigger interrupt enable TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_INT_BRK</i>	break interrupt enable TIMERx(x=0,14,40 for GD32L235)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the TIMER1 update interrupt */
```

```
timer_interrupt_enable(TIMER1, TIMER_INT_UP);
```

### timer\_interrupt\_disable

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

**Table 3-823. 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 TIMERx(x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<i>TIMER_INT_CH0</i>	channel 0 interrupt enable TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_INT_CH1</i>	channel 1 interrupt enable TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_INT_CH2</i>	channel 2 interrupt enable TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_INT_CH3</i>	channel 3 interrupt enable TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)

<i>TIMER_INT_CMT</i>	commutation interrupt enable TIMERx(x=0,14,40 for GD32L235)
<i>TIMER_INT_TRG</i>	trigger interrupt enable TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_INT_BRK</i>	break interrupt enable TIMERx(x=0,14,40 for GD32L235)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the TIMER1 update interrupt */
```

```
timer_interrupt_disable(TIMER1, TIMER_INT_UP);
```

### timer\_interrupt\_flag\_get

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

**Table 3-824. Function timer\_interrupt\_flag\_get**

<b>Function name</b>	timer_interrupt_flag_get
<b>Function prototype</b>	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	get timer interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt enable TIMERx(x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt enable TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt enable TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt enable TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt enable TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_INT_FLAG_CM</i>	commutation interrupt enable

<i>T</i>	TIMERx(x=0,14,40 for GD32L235)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt enable TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt enable TIMERx(x=0,14,40 for GD32L235)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TIMER1 update interrupt flag */
```

```
FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = timer_interrupt_flag_get(TIMER1, TIMER_INT_FLAG_UP);
```

### timer\_interrupt\_flag\_clear

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

**Table 3-825. 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 enable TIMERx(x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt enable TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt enable TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt enable TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt enable TIMERx(x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_INT_FLAG_CM</i> <i>T</i>	commutation interrupt enable TIMERx(x=0,14,40 for GD32L235)



<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt enable TIMERx(x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt enable TIMERx(x=0,14,40 for GD32L235)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TIMER1 update interrupt flag */
```

```
timer_interrupt_flag_clear(TIMER1, TIMER_INT_FLAG_UP);
```

## 3.26. TRNG

The true random number generator (TRNG) module can generate a 32-bit value using continuous analog noise. The TRNG registers are listed in chapter [3.26.1](#). the TRNG firmware functions are introduced in chapter [3.26.2](#).

### 3.26.1. Descriptions of Peripheral registers

TRNG registers are listed in the table shown as below:

**Table 3-826 TRNG Registers**

Registers	Descriptions
TRNG_CTL	TRNG control register
TRNG_STAT	TRNG status register
TRNG_DATA	TRNG data register

### 3.26.2. Descriptions of Peripheral functions

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

**Table 3-827. TRNG firmware function**

Function name	Function description
trng_deinit	deinitialize the TRNG
trng_enable	enable the TRNG
trng_disable	disable the TRNG
trng_get_true_random_data	get the true random data
trng_flag_get	get the TRNG status flags
trng_interrupt_enable	enable the TRNG interrupt
trng_interrupt_disable	disable the TRNG interrupt
trng_interrupt_flag_get	get the TRNG interrupt flags

Function name	Function description
trng_interrupt_flag_clear	clear the TRNG interrupt flags

### Enum trng\_flag\_enum

Table 3-828. Enum trng\_flag\_enum

Member name	Function description
TRNG_FLAG_DRDY	random data ready status
TRNG_FLAG_CECS	clock error current status
TRNG_FLAG_SECS	seed error current status

### Enum trng\_int\_flag\_enum

Table 3-829. Enum trng\_int\_flag\_enum

Member name	Function description
TRNG_INT_FLAG_CEIF	clock error interrupt flag
TRNG_INT_FLAG_SEIF	seed error interrupt flag

### trng\_deinit

The description of trng\_deinit is shown as below:

Table 3-830. Function trng\_deinit

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

Example:

```
/* reset TRNG */
```

```
trng_deinit();
```

### trng\_enable

The description of trng\_enable is shown as below:

Table 3-831. Function trng\_enable

Function name	trng_enable
---------------	-------------

Function prototype	void trng_enable(void);
Function descriptions	enable the TRNG
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TRNG */
trng_enable();
```

### trng\_disable

The description of trng\_disable is shown as below:

**Table 3-832 Function trng\_disable**

Function name	trng_disable
Function prototype	void trng_disable(void);
Function descriptions	disable the TRNG
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TRNG */
trng_disable();
```

### trng\_get\_true\_random\_data

The description of trng\_get\_true\_random\_data is shown as below:

**Table 3-833 Function trng\_get\_true\_random\_data**

Function name	trng_get_true_random_data
Function prototype	uint32_t trng_get_true_random_data(void);

<b>Function descriptions</b>	get the true random data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	0x0 – 0xFFFFFFFF

Example:

```
/* get true random data */

uint32_t data;

data = trng_get_true_random_data();
```

### trng\_flag\_get

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

**Table 3-834 trng\_flag\_get**

<b>Function name</b>	trng_flag_get
<b>Function prototype</b>	FlagStatus trng_flag_get(trng_flag_enum flag);
<b>Function descriptions</b>	get the trng status flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
flag	TRNG status flag, refer to <a href="#">Table 3-828. Enum trng_flag_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
FlagStatus	SET or RESET

Example:

```
/* get TRNG clock error current flag status */

FlagStatus flag_status = RESET;

flag_status == trng_flag_get(TRNG_FLAG_CECS);
```

### trng\_interrupt\_enable

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

**Table 3-835 trng\_interrupt\_enable**

<b>Function name</b>	trng_interrupt_enable
----------------------	-----------------------

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

Example:

```
/* enable TRNG interrupt */
trng_interrupt_enable();
```

### trng\_interrupt\_disable

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

**Table 3-836 trng\_interrupt\_disable**

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

Example:

```
/* disable TRNG interrupt */
trng_interrupt_disable();
```

### trng\_interrupt\_flag\_get

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

**Table 3-837 trng\_interrupt\_flag\_get**

<b>Function name</b>	trng_interrupt_flag_get
<b>Function prototype</b>	FlagStatus trng_interrupt_flag_get(trng_int_flag_enum int_flag);

<b>Function descriptions</b>	get the trng interrupt flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	TRNG interrupt flag, refer to <a href="#">Table 3-829. Enum trng_int_flag_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TRNG clock error interrupt flag */
```

```
FlagStatus interrupt_flag = RESET;
```

```
interrupt_flag = trng_interrupt_flag_get(TRNG_INT_FLAG_CEIF);
```

### trng\_interrupt\_flag\_clear

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

**Table 3-838 trng\_interrupt\_flag\_clear**

<b>Function name</b>	trng_interrupt_flag_clear
<b>Function prototype</b>	void trng_interrupt_flag_get(trng_int_flag_enum int_flag);
<b>Function descriptions</b>	clear the trng interrupt flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	TRNG interrupt flag, refer to <a href="#">Table 3-829. Enum trng_int_flag_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TRNG clock error interrupt flag */
```

```
trng_interrupt_flag_clear(TRNG_INT_FLAG_CEIF);
```

## 3.27. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter [3.27.1](#), the USART firmware functions are introduced in chapter [3.27.2](#).

### 3.27.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

**Table 3-839. 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

### 3.27.2. Descriptions of Peripheral functions

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

**Table 3-840. USART firmware function**

Function name	Function description
usart_deinit	reset USART
usart_baudrate_set	configure USART baud rate value
usart_parity_config	configure USART parity function
usart_word_length_set	configure USART word length
usart_stop_bit_set	configure USART stop bit length
usart_enable	enable USART
usart_disable	disable USART
usart_transmit_config	configure USART transmitter
usart_receive_config	configure USART receiver
usart_data_first_config	data is transmitted/received with the LSB/MSB first
usart_invert_config	configure USART inverted
usart_overrun_enable	enable the USART overrun function
usart_overrun_disable	disable the USART overrun function
usart_oversample_config	configure the USART oversample mode
usart_sample_bit_config	configure sample bit method
usart_receiver_timeout_enable	enable receiver timeout
usart_receiver_timeout_disable	disable receiver timeout

Function name	Function description
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_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
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



Function name	Function description
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/RFCs 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 3-841. Enum usart\_flag\_enum**

Member name	Function description
USART_FLAG_REA	receive enable acknowledge flag
USART_FLAG_TEA	transmit enable acknowledge flag
USART_FLAG_WU	wakeup from Deep-sleep mode flag
USART_FLAG_RWU	receiver wakeup from mute mode
USART_FLAG_SB	send break flag
USART_FLAG_AM	ADDR match flag
USART_FLAG_BSY	busy flag
USART_FLAG_EB	end of block flag
USART_FLAG_RT	receiver timeout flag
USART_FLAG_CTS	CTS level
USART_FLAG_CTSF	CTS change flag
USART_FLAG_LBD	LIN break detected flag
USART_FLAG_TBE	transmit data buffer empty
USART_FLAG_TC	transmission complete
USART_FLAG_RBNE	read data buffer not empty
USART_FLAG_IDLE	IDLE line detected flag
USART_FLAG_ORERR	overrun error
USART_FLAG_NERR	noise error flag

Member name	Function description
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 3-842. 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_ORERRR	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_ORERRR	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 3-843. 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

Member name	Function description
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 3-844. 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 3-845. Function usart\_deinit**

<b>Function name</b>	usart_deinit
<b>Function prototype</b>	void usart_deinit(uint32_t usart_periph);
<b>Function descriptions</b>	reset USART
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset USART0 */
usart_deinit(USART0);
```

## usart\_baudrate\_set

The description of usart\_baudrate\_set is shown as below:

**Table 3-846. Function usart\_baudrate\_set**

<b>Function name</b>	usart_baudrate_set
<b>Function prototype</b>	void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);
<b>Function descriptions</b>	configure USART baud rate value
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get
<b>Input parameter{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>baudval</b>	baud rate value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 baud rate value */
usart_baudrate_set(USART0, 115200);
```

## usart\_parity\_config

The description of usart\_parity\_config is shown as below:

**Table 3-847. Function usart\_parity\_config**

<b>Function name</b>	usart_parity_config
<b>Function prototype</b>	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
<b>Function descriptions</b>	configure USART parity function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>paritycfg</b>	configure USART parity
USART_PM_NONE	no parity
USART_PM_ODD	odd parity
USART_PM_EVEN	even parity

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 parity */
```

```
usart_parity_config(USART0, USART_PM_EVEN);
```

### usart\_word\_length\_set

The description of usart\_word\_length\_set is shown as below:

**Table 3-848. Function usart\_word\_length\_set**

Function name	usart_word_length_set
Function prototype	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);
Function descriptions	configure USART word length
Precondition	-
The called functions	-
Input parameter{in}	
<b>usart_periph</b>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
Input parameter{in}	
<b>wlen</b>	USART word length configure
USART_WL_8BIT	8 bits
USART_WL_9BIT	9 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 word length */
```

```
usart_word_length_set(USART0, USART_WL_9BIT);
```

### usart\_stop\_bit\_set

The description of usart\_stop\_bit\_set is shown as below:

**Table 3-849. Function usart\_stop\_bit\_set**

Function name	usart_stop_bit_set
Function prototype	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);

<b>Function descriptions</b>	configure USART stop bit length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx
<i>USARTx</i>	x=0,1
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>stblen</b>	USART stop bit configure
<i>USART_STB_1BIT</i>	1 bit
<i>USART_STB_0_5BIT</i>	0.5 bit
<i>USART_STB_2BIT</i>	2 bits
<i>USART_STB_1_5BIT</i>	1.5 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 stop bit length */
usart_stop_bit_set(USART0, USART_STB_1_5BIT);
```

### usart\_enable

The description of usart\_enable is shown as below:

**Table 3-850. 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>	外设USARTx/UARTx
<i>USARTx</i>	x=0,1
<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 */
```

```
usart_enable(USART0);
```

## usart\_disable

The description of usart\_disable is shown as below:

**Table 3-851. 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>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 */
```

```
usart_disable(USART0);
```

## usart\_transmit\_config

The description of usart\_transmit\_config is shown as below:

**Table 3-852. 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>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>txconfig</b>	enable or disable USART transmitter
USART_TRANSMIT_ENABLE	enable USART transmission
USART_TRANSMIT_DISABLE	disable USART transmission

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

Example:

```
/* configure USART0 transmitter */
```

```
usart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

### usart\_receive\_config

The description of usart\_receive\_config is shown as below:

**Table 3-853. 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>	外设USARTx/UARTx
<i>USARTx</i>	x=0,1
<i>UARTx</i>	x=3,4
Input parameter{in}	
<b>rxconfig</b>	enable or disable USART receiver
<i>USART_RECEIVE_ENABLE</i>	enable USART reception
<i>USART_RECEIVE_DISABLE</i>	disable USART reception
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 receiver */
```

```
usart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

### usart\_data\_first\_config

The description of usart\_data\_first\_config is shown as below:



Table 3-854. Function usart\_data\_first\_config

Function name	usart_data_first_config
Function prototype	void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);
Function descriptions	data is transmitted/received with the LSB/MSB first
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
Input parameter{in}	
msbf	LSB/MSB
USART_MSBF_LSB	LSB first
USART_MSBF_MSB	MSB first
Output parameter{out}	
-	-
Return value	
-	-

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 3-855. Function usart\_invert\_config

Function name	usart_invert_config
Function prototype	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);
Function descriptions	configure USART inverted
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
Input parameter{in}	
invertpara	refer to <a href="#">Table 3-844. Enum usart_invert_enum</a>
USART_DINV_ENABLER	data bit level inversion
USART_DINV_DISABLER	data bit level not inversion

<i>E</i>	
<i>USART_TXPIN_ENAB</i> <i>LE</i>	TX pin level inversion
<i>USART_TXPIN_DISAB</i> <i>LE</i>	TX pin level not inversion
<i>USART_RXPIN_ENAB</i> <i>LE</i>	RX pin level inversion
<i>USART_RXPIN_DISAB</i> <i>LE</i>	RX pin level not inversion
<i>USART_SWAP_ENAB</i> <i>LE</i>	swap TX/RX pins
<i>USART_SWAP_DISAB</i> <i>LE</i>	not swap TX/RX pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 inversion */
```

```
usart_invert_config(USART0, USART_DINV_ENABLE);
```

### usart\_overrun\_enable

The description of usart\_overrun\_enable is shown as below:

**Table 3-856. Function usart\_overrun\_enable**

<b>Function name</b>	usart_overrun_enable
<b>Function prototype</b>	void usart_overrun_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable the USART overrun function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx
<i>USARTx</i>	x=0,1
<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 overrun */
```

```
usart_oversample_enable(USART0);
```

### usart\_oversample\_disable

The description of usart\_oversample\_disable is shown as below:

**Table 3-857. Function usart\_oversample\_disable**

<b>Function name</b>	usart_oversample_disable
<b>Function prototype</b>	void usart_oversample_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable the USART oversample function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 oversample */
```

```
usart_oversample_disable(USART0);
```

### usart\_oversample\_config

The description of usart\_oversample\_config is shown as below:

**Table 3-858. 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>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
<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>	

-	-
Return value	
-	-

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 3-859. 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>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
<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 3-860. 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=0,1
<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 3-861. 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=0,1
<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 3-862. Function usart\_receiver\_timeout\_threshold\_config**

<b>Function name</b>	usart_receiver_timeout_threshold_config
<b>Function prototype</b>	void usart_receiver_timeout_threshold_config(uint32_t usart_periph,

	uint32_t rtimeout);
<b>Function descriptions</b>	configure receiver timeout threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>rtimeout</b>	receiver timeout (0x00000000-0x00FFFFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the receiver timeout threshold of USART0*/
```

```
usart_receiver_timeout_threshold_config(USART0, 115200*3);
```

### usart\_data\_transmit

The description of usart\_data\_transmit is shown as below:

**Table 3-863. Function usart\_data\_transmit**

<b>Function name</b>	usart_data_transmit
<b>Function prototype</b>	void usart_data_transmit(uint32_t usart_periph, uint16_t data);
<b>Function descriptions</b>	USART transmit data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx
<i>USARTx</i>	x=0,1
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>data</b>	data of transmission (0x00-0x1FF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 transmit data */
```

```
usart_data_transmit(USART0, 0xAA);
```

## usart\_data\_receive

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

**Table 3-864. Function usart\_data\_receive**

<b>Function name</b>	usart_data_receive
<b>Function prototype</b>	uint16_t usart_data_receive(uint32_t usart_periph);
<b>Function descriptions</b>	USART receive data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	data of received (0x00-0x1FF)

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 3-865. Function usart\_command\_enable**

<b>Function name</b>	usart_command_enable
<b>Function prototype</b>	void usart_command_enable(uint32_t usart_periph, uint32_t cmdtype);
<b>Function descriptions</b>	enable USART command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>cmdtype</b>	command type
USART_CMD_SBKCMD	send break command
USART_CMD_MMCMMD	mute mode command

<i>USART_CMD_RXFCM</i> <i>D</i>	receive data flush command
<i>USART_CMD_TXFCM</i> <i>D</i>	transmit data flush request
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 command */
```

```
usart_command_enable(USART0, USART_CMD_SBKCMD);
```

### usart\_address\_config

The description of usart\_address\_config is shown as below:

**Table 3-866. Function usart\_address\_config**

<b>Function name</b>	usart_address_config
<b>Function prototype</b>	void usart_address_config(uint32_t usart_periph, uint8_t addr);
<b>Function descriptions</b>	configure the address of the USART
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx
<i>USARTx</i>	x=0,1
<i>UARTx</i>	x=3,4
<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 3-867. Function `usart_address_detection_mode_config`

Function name	<code>usart_address_detection_mode_config</code>
Function prototype	<code>void usart_address_detection_mode_config(uint32_t usart_periph, uint32_t addmod);</code>
Function descriptions	configure address detection mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	外设USARTx/UARTx
<code>USARTx</code>	x=0,1
<code>UARTx</code>	x=3,4
Input parameter{in}	
<code>addmod</code>	address detection mode
<code>USART_ADDDM_4BIT</code>	4 bits
<code>USART_ADDDM_FULLBIT</code>	full bits
Output parameter{out}	
-	-
Return value	
-	-

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 3-868. Function `usart_mute_mode_enable`

Function name	<code>usart_mute_mode_enable</code>
Function prototype	<code>void usart_mute_mode_enable(uint32_t usart_periph);</code>
Function descriptions	enable mute mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	外设USARTx/UARTx
<code>USARTx</code>	x=0,1
<code>UARTx</code>	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 receiver in mute mode */
usart_mute_mode_enable(USART0);
```

### usart\_mute\_mode\_disable

The description of usart\_mute\_mode\_disable is shown as below:

**Table 3-869. 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>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
<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 3-870. 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>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
<b>Input parameter{in}</b>	

<b>wmethod</b>	two methods be used to enter or exit the mute mode
<i>USART_WM_IDLE</i>	idle line
<i>USART_WM_ADDR</i>	address mask
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 wakeup method in mute mode */
```

```
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

### usart\_lin\_mode\_enable

The description of usart\_lin\_mode\_enable is shown as below:

**Table 3-871. 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
<i>USARTx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 LIN mode enable */
```

```
usart_lin_mode_enable(USART0);
```

### usart\_lin\_mode\_disable

The description of usart\_lin\_mode\_disable is shown as below:

**Table 3-872. Function usart\_lin\_mode\_disable**

<b>Function name</b>	usart_lin_mode_disable
<b>Function prototype</b>	void usart_lin_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable LIN mode
<b>Precondition</b>	-

The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1
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 3-873. 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}	
usart_periph	usart peripheral
USARTx	x=0,1
Input parameter{in}	
lblen	two methods be used to enter or exit the mute mode
USART_LBLEN_10B	10 bits
USART_LBLEN_11B	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 3-874. Function `usart_halfduplex_enable`

Function name	<code>usart_halfduplex_enable</code>
Function prototype	<code>void usart_halfduplex_enable(uint32_t usart_periph);</code>
Function descriptions	enable half-duplex mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	外设USARTx/UARTx
<code>USARTx</code>	x=0,1
<code>UARTx</code>	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 half duplex mode*/
usart_halfduplex_enable(USART0);
```

### `usart_halfduplex_disable`

The description of `usart_halfduplex_disable` is shown as below:

Table 3-875. 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>	外设USARTx/UARTx
<code>USARTx</code>	x=0,1
<code>UARTx</code>	x=3,4
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 3-876. Function usart\_clock\_enable**

<b>Function name</b>	usart_clock_enable
<b>Function prototype</b>	void usart_clock_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable 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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable clock */
usart_clock_enable(USART0);
```

## usart\_clock\_disable

The description of usart\_clock\_disable is shown as below:

**Table 3-877. Function usart\_clock\_disable**

<b>Function name</b>	usart_clock_disable
<b>Function prototype</b>	void usart_clock_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable 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
<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 3-878. Function usart\_synchronous\_clock\_config**

<b>Function name</b>	usart_synchronous_clock_config
<b>Function prototype</b>	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
<b>Function descriptions</b>	configure USART synchronous mode parameters
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1
<b>Input parameter{in}</b>	
<b>clen</b>	last bit clock pulse
USART_CLEN_NONE	clock pulse of the last data bit (MSB) is not output to the CK pin
USART_CLEN_EN	clock pulse of the last data bit (MSB) is output to the CK pin
<b>Input parameter{in}</b>	
<b>cph</b>	clock phase
USART_CPH_1CK	first clock transition is the first data capture edge
USART_CPH_2CK	second clock transition is the first data capture edge
<b>Input parameter{in}</b>	
<b>cpl</b>	clock polarity
USART_CPL_LOW	steady low value on CK pin
USART_CPL_HIGH	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 3-879. 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=0,1
<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 3-880. Function usart\_smartcard\_mode\_enable**

<b>Function name</b>	usart_smartcard_mode_enable
<b>Function prototype</b>	void usart_smartcard_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<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 3-881. Function `usart_smartcard_mode_disable`

<b>Function name</b>	<code>usart_smartcard_mode_disable</code>
<b>Function prototype</b>	<code>void usart_smartcard_mode_disable(uint32_t usart_periph);</code>
<b>Function descriptions</b>	disable smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<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 3-882. Function `usart_smartcard_mode_nack_enable`

<b>Function name</b>	<code>usart_smartcard_mode_nack_enable</code>
<b>Function prototype</b>	<code>void usart_smartcard_mode_nack_enable(uint32_t usart_periph);</code>
<b>Function descriptions</b>	enable NACK in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<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 3-883. Function `usart_smartcard_mode_nack_disable`

<b>Function name</b>	<code>usart_smartcard_mode_nack_disable</code>
<b>Function prototype</b>	<code>void usart_smartcard_mode_nack_disable(uint32_t usart_periph);</code>
<b>Function descriptions</b>	disable NACK in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 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 3-884. Function `usart_smartcard_mode_early_nack_enable`

<b>Function name</b>	<code>usart_smartcard_mode_early_nack_enable</code>
<b>Function prototype</b>	<code>void usart_smartcard_mode_early_nack_enable(uint32_t usart_periph);</code>
<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=0,1
<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 3-885. Function `usart_smartcard_mode_early_nack_disable`

<b>Function name</b>	<code>usart_smartcard_mode_early_nack_disable</code>
<b>Function prototype</b>	<code>void usart_smartcard_mode_early_nack_disable(uint32_t usart_periph);</code>
<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=0,1
<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 3-886. 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=0,1
<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 3-887. Function usart\_block\_length\_config**

<b>Function name</b>	usart_block_length_config
<b>Function prototype</b>	void usart_block_length_config(uint32_t usart_periph, uint32_t bl);
<b>Function descriptions</b>	configure block length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
<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 3-888. 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=0,1
<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 3-889. Function usart\_irda\_mode\_disable**

<b>Function name</b>	usart_irda_mode_disable
<b>Function prototype</b>	void usart_irda_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable IrDA mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
USARTx	x=0,1
<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 3-890. 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=0,1
<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 3-891. 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=0,1
<b>Input parameter{in}</b>	
<b>irlp</b>	IrDA low-power or normal
USART_IRLP_LOW	low-power
USART_IRLP_NORMAL	normal
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 IrDA low-power */
```

```
usart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

### usart\_hardware\_flow\_rts\_config

The description of usart\_hardware\_flow\_rts\_config is shown as below:

**Table 3-892. Function usart\_hardware\_flow\_rts\_config**

<b>Function name</b>	usart_hardware_flow_rts_config
<b>Function prototype</b>	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
<b>Function descriptions</b>	configure hardware flow control RTS
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral

<i>USARTx</i>	<i>x</i> =0,1
<b>Input parameter{in}</b>	
<b>rtsconfig</b>	enable or disable RTS
<i>USART_RTS_ENABLE</i>	enable RTS
<i>USART_RTS_DISABLE</i> <i>E</i>	disable RTS
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 hardware flow control RTS */
```

```
usart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);
```

### usart\_hardware\_flow\_cts\_config

The description of usart\_hardware\_flow\_cts\_config is shown as below:

**Table 3-893. Function usart\_hardware\_flow\_cts\_config**

<b>Function name</b>	usart_hardware_flow_cts_config
<b>Function prototype</b>	void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
<b>Function descriptions</b>	configure hardware flow control CTS
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	<i>x</i> =0,1
<b>Input parameter{in}</b>	
<b>ctsconfig</b>	enable or disable CTS
<i>USART_CTS_ENABLE</i>	enable CTS
<i>USART_CTS_DISABLE</i> <i>E</i>	disable CTS
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 hardware flow control CTS */
```

```
usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

## usart\_hardware\_flow\_coherence\_config

The description of usart\_hardware\_flow\_coherence\_config is shown as below:

**Table 3-894. 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
<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 3-895. 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
USARTx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-



Example:

```
/* enable USART0 RS485 driver */

usart_rs485_driver_enable(USART0);
```

### usart\_rs485\_driver\_disable

The description of usart\_rs485\_driver\_disable is shown as below:

**Table 3-896. 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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 RS485 driver */

usart_rs485_driver_disable(USART0);
```

### usart\_driver\_asserttime\_config

The description of usart\_driver\_asserttime\_config is shown as below:

**Table 3-897. Function usart\_driver\_asserttime\_config**

<b>Function name</b>	usart_driver_asserttime_config
<b>Function prototype</b>	void usart_driver_asserttime_config(uint32_t usart_periph, uint32_t deatime);
<b>Function descriptions</b>	configure driver enable assertion time
<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
<b>Input parameter{in}</b>	
<b>deatime</b>	driver enable assertion time (0x00-0x0000001F)
<b>Output parameter{out}</b>	

-	-
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 3-898. Function usart\_driver\_deassertime\_config**

<b>Function name</b>	usart_driver_deassertime_config
<b>Function prototype</b>	void usart_driver_deassertime_config(uint32_t usart_periph, uint32_t dedtime);
<b>Function descriptions</b>	configure driver enable de-assertion time
<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
<b>Input parameter{in}</b>	
<b>dedtime</b>	driver enable de-assertion time (0x00-0x0000001F)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set USART0 driver deassertime */
```

```
usart_driver_deassertime_config(USART0, 0x0000001F);
```

### usart\_depolarity\_config

The description of usart\_depolarity\_config is shown as below:

**Table 3-899. 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
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 3-900. Function usart\_dma\_receive\_config**

<b>Function name</b>	usart_dma_receive_config
<b>Function prototype</b>	void usart_dma_receive_config(uint32_t usart_periph, uint32_t dmacmd);
<b>Function descriptions</b>	configure USART DMA for reception
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	外设USARTx/UARTx
<i>USARTx</i>	x=0,1
<i>UARTx</i>	x=3,4
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

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 3-901. Function usart\_dma\_transmit\_config**

<b>Function name</b>	usart_dma_transmit_config
<b>Function prototype</b>	void usart_dma_transmit_config(uint32_t usart_periph, uint32_t dmacmd);
<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>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>dmacmd</b>	enable or disable DMA for transmission
USART_TRANSMIT_DMA_ENABLE	DMA enable for transmission
USART_TRANSMIT_DMA_DISABLE	DMA disable for transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 DMA enable for transmission */
```

```
usart_dma_transmit_config(USART0, USART_TRANSMIT_DMA_ENABLE);
```

### usart\_reception\_error\_dma\_disable

The description of usart\_reception\_error\_dma\_disable is shown as below:

**Table 3-902. Function usart\_reception\_error\_dma\_disable**

<b>Function name</b>	usart_reception_error_dma_disable
<b>Function prototype</b>	void usart_reception_error_dma_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable DMA on reception error
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx
USARTx	x=0,1

<i>UARTx</i>	<i>x=3,4</i>
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 3-903. Function usart\_reception\_error\_dma\_enable**

<b>Function name</b>	usart_reception_error_dma_enable
<b>Function prototype</b>	void usart_reception_error_dma_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable DMA on reception error
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	外设USARTx/UARTx
<i>USARTx</i>	<i>x=0,1</i>
<i>UARTx</i>	<i>x=3,4</i>
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 3-904. Function usart\_wakeup\_enable**

<b>Function name</b>	usart_wakeup_enable
<b>Function prototype</b>	void usart_wakeup_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable USART to wakeup the mcu from deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
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 3-905. Function usart\_wakeup\_disable**

<b>Function name</b>	usart_wakeup_disable
<b>Function prototype</b>	void usart_wakeup_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable USART to wakeup the mcu from deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1
Output parameter{out}	
-	-
Return value	
-	-

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 3-906. 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>	-

The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1
Input parameter{in}	
wum	wakeup mode
USART_WUM_ADDR	WUF active on address match
USART_WUM_START B	WUF active on start bit
USART_WUM_RBNE	WUF active on RBNE
Output parameter{out}	
-	-
Return value	
-	-

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 3-907. Function usart\_receive\_fifo\_enable**

Function name	usart_receive_fifo_enable
Function prototype	void usart_receive_fifo_enable(uint32_t usart_periph);
Function descriptions	enable receive FIFO
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
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 3-908. Function usart\_receive\_fifo\_disable**

<b>Function name</b>	usart_receive_fifo_disable
<b>Function prototype</b>	void usart_receive_fifo_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable receive FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-909. Function usart\_receive\_fifo\_counter\_number**

<b>Function name</b>	usart_receive_fifo_counter_number
<b>Function prototype</b>	uint8_t usart_receive_fifo_counter_number(uint32_t usart_periph);
<b>Function descriptions</b>	read receive FIFO counter number
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint8_t	receive FIFO counter number

Example:

```
/* read receive FIFO counter number */
```



```
uint8_t temp;
```

```
temp = usart_receive_fifo_counter_number(USART0);
```

## usart\_flag\_get

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

**Table 3-910. Function usart\_flag\_get**

<b>Function name</b>	usart_flag_get
<b>Function prototype</b>	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
<b>Function descriptions</b>	get flag in STAT/RFCR register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags, refer to <a href="#">Table 3-841. Enum usart_flag_enum</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
USART_FLAG_CTSF	CTS change flag
USART_FLAG_CTS	CTS level
USART_FLAG_RT	receiver timeout flag
USART_FLAG_EB	end of block flag
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
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

USART_FLAG_RFF	receive FIFO full flag
USART_FLAG_RFFINT	receive FIFO full interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get flag USART0 state */
```

```
FlagStatus status;
```

```
status = usart_flag_get(USART0, USART_FLAG_TBE);
```

### usart\_flag\_clear

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

**Table 3-911. 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>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags, refer to <a href="#">Table 3-841. Enum usart_flag_enum</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
USART_FLAG_CTSF	CTS change flag
USART_FLAG_RT	receiver timeout flag
USART_FLAG_EB	end of block flag
USART_FLAG_AM	address match flag
USART_FLAG_WU	wakeup from deep-sleep mode flag

USART_FLAG_EPERR	early parity error flag
Output parameter{out}	
-	-
Return value	
-	-

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 3-912. 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>	-
Input parameter{in}	
<b>usart_periph</b>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
Input parameter{in}	
<b>interrupt</b>	interrupt type, refer to <a href="#">Table 3-843. Enum usart_interrupt_enum</a> only one among these parameters can be selected
USART_INT_IDLE	idle interrupt
USART_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt enable interrupt
USART_INT_TC	transmission complete interrupt
USART_INT_TBE	transmit data register empty interrupt
USART_INT_PERR	parity error interrupt
USART_INT_AM	address match interrupt
USART_INT_RT	receiver timeout interrupt
USART_INT_EB	end of block interrupt
USART_INT_LBD	LIN break detection interrupt
USART_INT_ERR	error interrupt enable in multibuffer communication
USART_INT_CTS	CTS interrupt
USART_INT_WU	wakeup from deep-sleep mode interrupt
USART_INT_RFF	receive FIFO full interrupt enable
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable USART0 TBE interrupt */
```

```
usart_interrupt_enable(USART0, USART_INT_TBE);
```

### usart\_interrupt\_disable

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

**Table 3-913. 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>	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt type, refer to <a href="#">Table 3-843. Enum usart_interrupt_enum</a> only one among these parameters can be selected
USART_INT_IDLE	idle interrupt
USART_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt enable interrupt
USART_INT_TC	transmission complete interrupt
USART_INT_TBE	transmit data register empty interrupt
USART_INT_PERR	parity error interrupt
USART_INT_AM	address match interrupt
USART_INT_RT	receiver timeout interrupt
USART_INT_EB	end of block interrupt
USART_INT_LBD	LIN break detection interrupt
USART_INT_ERR	error interrupt enable in multibuffer communication
USART_INT_CTS	CTS interrupt
USART_INT_WU	wakeup from deep-sleep mode interrupt
USART_INT_RFF	receive FIFO full interrupt enable
<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 3-914. Function usart\_interrupt\_flag\_get**

Function name	usart_interrupt_flag_get
Function prototype	FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
Function descriptions	get USART interrupt and flag status
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
Input parameter{in}	
int_flag	USART interrupt flag, refer to <a href="#">Table 3-842. Enum usart_interrupt_flag_enum</a> , only one among these parameters can be selected
USART_INT_FLAG_EB	end of block interrupt and flag
USART_INT_FLAG_RT	receiver timeout interrupt and flag
USART_INT_FLAG_A M	address match interrupt and flag
USART_INT_FLAG_PE RR	parity error interrupt and flag
USART_INT_FLAG_TB E	transmitter buffer empty interrupt and flag
USART_INT_FLAG_TC	transmission complete interrupt and flag
USART_INT_FLAG_RB NE	read data buffer not empty interrupt and flag
USART_INT_FLAG_RB NE_ORERR	read data buffer not empty interrupt and overrun error flag
USART_INT_FLAG_ID LE	IDLE line detected interrupt and flag
USART_INT_FLAG_LB D	LIN break detected interrupt and flag
USART_INT_FLAG_W	wakeup from deep-sleep mode interrupt and flag

<i>U</i>	
USART_INT_FLAG_CTS	CTS interrupt and flag
USART_INT_FLAG_ERR_NERR	error interrupt and noise error flag
USART_INT_FLAG_ERR_ORERR	error interrupt and overrun error
USART_INT_FLAG_ERR_FERR	error interrupt and frame error flag
USART_INT_FLAG_RF	receive FIFO full interrupt and flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the USART0 interrupt flag status */
```

```
FlagStatus status;
```

```
status = usart_interrupt_flag_get(USART0, USART_INT_FLAG_RBNE);
```

### usart\_interrupt\_flag\_clear

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

**Table 3-915. 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>	-
Input parameter{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
Input parameter{in}	
int_flag	USART interrupt flag, refer to <a href="#">Table 3-842. Enum usart_interrupt_flag_enum</a> , only one among these parameters can be selected
USART_INT_FLAG_PERR	parity error flag
USART_INT_FLAG_ER	frame error flag

<i>R_FERR</i>	
<i>USART_INT_FLAG_ER</i> <i>R_NERR</i>	noise detected flag
<i>USART_INT_FLAG_RB</i> <i>NE_ORERR</i>	read data buffer not empty interrupt and overrun error flag
<i>USART_INT_FLAG_ER</i> <i>R_ORERR</i>	error interrupt and overrun error
<i>USART_INT_FLAG_ID</i> <i>LE</i>	idle line detected flag
<i>USART_INT_FLAG_TC</i>	transmission complete flag
<i>USART_INT_FLAG_LB</i> <i>D</i>	LIN break detected flag
<i>USART_INT_FLAG_CT</i> <i>S</i>	CTS change flag
<i>USART_INT_FLAG_RT</i>	receiver timeout flag
<i>USART_INT_FLAG_EB</i>	end of block flag
<i>USART_INT_FLAG_A</i> <i>M</i>	address match flag
<i>USART_INT_FLAG_W</i> <i>U</i>	wakeup from deep-sleep mode flag
<i>USART_INT_FLAG_RF</i> <i>F</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);
```

## 3.28. VREF

The precision internal reference is used to provide reference voltage for ADC/DAC, or used by off-chip circuit connecting to V<sub>REF</sub> pin. The VREF registers are listed in chapter [3.28.1](#), the VREF firmware functions are introduced in chapter [3.28.2](#).

### 3.28.1. Descriptions of Peripheral registers

VREF registers are listed in the table shown as below:

Table 3-916. WWDGT Registers

Registers	Descriptions
VREF_CS	VREF Control and status register
VREF_CALIB	VREF Calibration register

### 3.28.2. Descriptions of Peripheral functions

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

Table 3-917. VREF firmware function

Function name	Function description
vref_deinit	deinitialize the VREF
vref_enable	enable VREF
vref_disable	disable VREF
vref_high_impedance_mode_enable	enable VREF high impedance mode
vref_high_impedance_mode_disable	disable VREF high impedance mode
vref_voltage_select	select VREF voltage reference (only for L235xx series)
vref_status_get	get the status of VREF
vref_calib_value_set	set the calibration value of VREF
vref_calib_value_get	get the calibration value of VREF

#### vref\_deinit

The description of vref\_deinit is shown as below:

Table 3-918. Function vref\_deinit

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

Example:

```
/* deinitialize the VREF */
vref_deinit();
```



## vref\_enable

The description of vref\_enable is shown as below:

**Table 3-919. Function vref\_enable**

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

example:

```
/* enable VREF */
vref_enable();
```

## vref\_disable

The description of vref\_disable is shown as below:

**Table 3-920. Function vref\_disable**

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

example:

```
/* disable VREF */
vref_disable();
```

## vref\_high\_impedance\_mode\_enable

The description of vref\_high\_impedance\_mode\_enable is shown as below:

**Table 3-921. Function vref\_high\_impedance\_mode\_enable**

<b>Function name</b>	vref_high_impedance_mode_enable
<b>Function prototype</b>	void vref_high_impedance_mode_enable(void);
<b>Function descriptions</b>	enable VREF high impedance 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 VREF high impedance mode */
vref_high_impedance_mode_enable();
```

## vref\_high\_impedance\_mode\_disable

The description of vref\_high\_impedance\_mode\_disable is shown as below:

**Table 3-922. Function vref\_high\_impedance\_mode\_disable**

<b>Function name</b>	vref_high_impedance_mode_disable
<b>Function prototype</b>	void vref_high_impedance_mode_disable(void);
<b>Function descriptions</b>	disable VREF high impedance 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 VREF high impedance mode */
vref_high_impedance_mode_disable();
```

**vref\_voltage\_select (only for L235xx series)**

The description of vref\_voltage\_select is shown as below:

**Table 3-923. Function vref\_voltage\_select**

<b>Function name</b>	vref_voltage_select
<b>Function prototype</b>	void vref_voltage_select(uint32_t vref_voltage);
<b>Function descriptions</b>	select VREF voltage reference
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>vref_voltage</b>	reference voltage
VREF_VOLTAGE_SEL_LEVEL_0	2.048 V
VREF_VOLTAGE_SEL_LEVEL_1	2.5 V
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

example:

```
/* VREF voltage reference select 2.5V */
```

```
vref_voltage_select(VREF_VOLTAGE_SEL_LEVEL_1);
```

**vref\_status\_get**

The description of vref\_status\_get is shown as below:

**Table 3-924. Function vref\_status\_get**

<b>Function name</b>	vref_status_get
<b>Function prototype</b>	FlagStatus vref_status_get(void);
<b>Function descriptions</b>	get the status of VREF
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

example:

```
/* get the status of VREF */
```

```
FlagStatus status;
```

```
status = vref_status_get();
```

### vref\_calib\_value\_set

The description of vref\_calib\_value\_set is shown as below:

**Table 3-925. Function vref\_calib\_value\_set**

<b>Function name</b>	vref_calib_value_set
<b>Function prototype</b>	void vref_calib_value_set(uint8_t value);
<b>Function descriptions</b>	set the calibration value of VREF
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>value</b>	calibration value (0x00 - 0x3F)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

example:

```
/* set the calibration value of VREF */
```

```
vref_calib_value_set(0x0A);
```

### vref\_calib\_value\_get

The description of vref\_calib\_value\_get is shown as below:

**Table 3-926. Function vref\_calib\_value\_get**

<b>Function name</b>	vref_calib_value_get
<b>Function prototype</b>	uint8_t vref_calib_value_get(void);
<b>Function descriptions</b>	get the calibration value of VREF
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	calibration value (0x00 - 0x3F)

example:

```
/* get the calibration value of VREF */
```

```
uint8_t cal_val;

cal_val = vref_calib_value_get();
```

## 3.29. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.29.1](#), the WWDGT firmware functions are introduced in chapter [3.29.2](#).

### 3.29.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

**Table 3-927. WWDGT Registers**

Registers	Descriptions
WWDGT_CTL	WWDGT control register
WWDGT_CFG	WWDGT configuration register
WWDGT_STAT	WWDGT status register

### 3.29.2. Descriptions of Peripheral functions

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

**Table 3-928. WWDGT firmware function**

Function name	Function description
wwdgt_deinit	reset the window watchdog timer configuration
wwdgt_enable	start the window watchdog timer counter
wwdgt_counter_update	configure the window watchdog timer counter value
wwdgt_config	configure counter value, window value, and prescaler divider value
wwdgt_interrupt_enable	enable early wakeup interrupt of WWDGT
wwdgt_flag_get	check early wakeup interrupt state of WWDGT
wwdgt_flag_clear	clear early wakeup interrupt state of WWDGT

#### wwdgt\_deinit

The description of wwdgt\_deinit is shown as below:

**Table 3-929. 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 3-930. 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\_counter\_update

The description of wwdgt\_counter\_update is shown as below:

**Table 3-931. Function wwdgt\_counter\_update**

Function name	wwdgt_counter_update
Function prototype	void wwdgt_counter_update(uint16_t counter_value);
Function descriptions	configure the window watchdog timer counter value
Precondition	-
The called functions	-
Input parameter{in}	

<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 3-932. 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	the time base of WWDGT counter = (PCLK1 / 4096) / 256

IV256	
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 IV8192	the time base of WWDGT counter = (PCLK1 / 4096) / 8192
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\_flag\_get

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

**Table 3-933. Function wwdgt\_flag\_get**

<b>Function name</b>	wwdgt_flag_get
<b>Function prototype</b>	FlagStatus wwdgt_flag_get(void);
<b>Function descriptions</b>	check early wakeup interrupt state of WWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* test if the counter value update has reached the 0x40 */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get();
```



## wwdgt\_flag\_clear

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

**Table 3-934. Function wwdgt\_flag\_clear**

<b>Function name</b>	wwdgt_flag_clear
<b>Function prototype</b>	void wwdgt_flag_clear(void);
<b>Function descriptions</b>	clear early wakeup interrupt state of WWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear early wakeup interrupt state of WWDGT */
```

```
wwdgt_flag_clear();
```

## 4. Revision history

**Table 4-1. Revision history**

Revision No.	Description	Date
1.0	Initial Release	Aug.25, 2021
1.1	Adjust the formatting of the document	Jul.8, 2022
1.2	Modify some function entries to be consistent with the firmware library code	Dec.21, 2022
1.3	Function descriptionS and Input parameter descriptionS Add SLCD in <b><u>Table 4-2.</u></b> <b><u>Functionrcu rtc clock config</u></b>	Jun.21, 2023
2.0	Add support GD32L235xx series	Jul.17, 2023
2.1	Modify some function descriptions	Nov.2, 2023
2.2	Delete pmu_lowepower_ldo_enable and pmu_lowepower_ldo_disable function for GD32L235	Jan.29, 2024
2.3	Update EXTI Line enum table in 3.11 EXTI chapter	Jul.30, 2024
2.4	1. Add rtc_lxtal_stab_reset_disable/rtc_lxtal_stab_reset_enable function 2. Delete i2c_nack_disable(uint32_t i2c_periph) function	Aug.8, 2025
2.5	1. Delete the spi_quad_io23_output_enable and spi_quad_io23_output_disable functions 2. The clock source of L232 LPTIEMR has been updated from APB1 to APB2, and the parameter of the rcu_lptimer_clock_config function has been modified to RCU_LPTIMERSRC_CKAPB2	Feb 2, 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.