

**GigaDevice Semiconductor Inc.**

**GD32F3x0**  
**ARM® Cortex™-M4 32-bit MCU**

**Firmware Library  
User Guide**

Revison 1.0

(Jun. 2019)

## Table of Contents

<b>Table of Contents .....</b>	<b>2</b>
<b>List of Figures .....</b>	<b>5</b>
<b>List of Tables .....</b>	<b>6</b>
<b>1. Introduction .....</b>	<b>22</b>
<b>1.1. Rules of User Manual and Firmware Library .....</b>	<b>22</b>
1.1.1. Peripherals .....	22
1.1.2. Naming rules .....	23
<b>2. Firmware Library Overview .....</b>	<b>24</b>
<b>2.1. File Structure of Firmware Library .....</b>	<b>24</b>
2.1.1. Examples Folder .....	25
2.1.2. Firmware Folder .....	25
2.1.3. Template Folder .....	25
2.1.4. Utilities Folder .....	29
<b>2.2. File descriptions of Firmware Library .....</b>	<b>29</b>
<b>3. Firmware Library of Standard Peripherals .....</b>	<b>31</b>
<b>3.1. Overview of Firmware Library of Standard Peripherals .....</b>	<b>31</b>
<b>3.2. ADC .....</b>	<b>31</b>
3.2.1. Descriptions of Peripheral registers .....	31
3.2.2. Descriptions of Peripheral functions .....	32
<b>3.3. CEC .....</b>	<b>55</b>
3.3.1. Descriptions of Peripheral registers .....	55
3.3.2. Descriptions of Peripheral functions .....	56
<b>3.4. CMP .....</b>	<b>72</b>
3.4.1. Descriptions of Peripheral registers .....	72
3.4.2. Descriptions of Peripheral functions .....	72
<b>3.5. CRC .....</b>	<b>81</b>
3.5.1. Descriptions of Peripheral registers .....	81
3.5.2. Descriptions of Peripheral functions .....	81
<b>3.6. CTC .....</b>	<b>89</b>
3.6.1. Descriptions of Peripheral registers .....	89
3.6.2. Descriptions of Peripheral functions .....	89
<b>3.7. DAC .....</b>	<b>102</b>
3.7.1. Descriptions of Peripheral registers .....	102
3.7.2. Descriptions of Peripheral functions .....	102

---

<b>3.8. DBG .....</b>	<b>116</b>
3.8.1. Descriptions of Peripheral registers .....	116
3.8.2. Descriptions of Peripheral functions .....	116
<b>3.9. DMA .....</b>	<b>120</b>
3.9.1. Descriptions of Peripheral registers .....	120
3.9.2. Descriptions of Peripheral functions .....	121
<b>3.10. EXTI .....</b>	<b>139</b>
3.10.1. Descriptions of Peripheral registers .....	139
3.10.2. Descriptions of Peripheral functions .....	139
<b>3.11. FMC .....</b>	<b>147</b>
3.11.1. Descriptions of Peripheral registers .....	147
3.11.2. Descriptions of Peripheral functions .....	148
<b>3.12. FWDGT .....</b>	<b>166</b>
3.12.1. Descriptions of Peripheral registers .....	166
3.12.2. Descriptions of Peripheral functions .....	166
<b>3.13. GPIO/AFIO .....</b>	<b>170</b>
3.13.1. Descriptions of Peripheral registers .....	170
3.13.2. Descriptions of Peripheral functions .....	171
<b>3.14. I2C .....</b>	<b>181</b>
3.14.1. Descriptions of Peripheral registers .....	181
3.14.2. Descriptions of Peripheral functions .....	182
<b>3.15. MISC .....</b>	<b>204</b>
3.15.1. Descriptions of Peripheral registers .....	204
3.15.2. Descriptions of Peripheral functions .....	205
<b>3.16. PMU .....</b>	<b>210</b>
3.16.1. Descriptions of Peripheral registers .....	211
3.16.2. Descriptions of Peripheral functions .....	211
<b>3.17. RCU .....</b>	<b>222</b>
3.17.1. Descriptions of Peripheral registers .....	223
3.17.2. Descriptions of Peripheral functions .....	223
<b>3.18. RTC .....</b>	<b>257</b>
3.18.1. Descriptions of Peripheral registers .....	257
3.18.2. Descriptions of Peripheral functions .....	258
<b>3.19. SPI/I2S .....</b>	<b>278</b>
3.19.1. Descriptions of Peripheral registers .....	278
3.19.2. Descriptions of Peripheral functions .....	278
<b>3.20. SYSCFG .....</b>	<b>304</b>
3.20.1. Descriptions of Peripheral registers .....	304
3.20.2. Descriptions of Peripheral functions .....	304



## GD32F3x0 Firmware Library User Guide

---

<b>3.21. TIMER.....</b>	<b>311</b>
3.21.1. Descriptions of Peripheral registers.....	311
3.21.2. Descriptions of Peripheral functions .....	312
<b>3.22. TSI .....</b>	<b>369</b>
3.22.1. Descriptions of Peripheral registers.....	369
3.22.2. Descriptions of Peripheral functions .....	370
<b>3.23. USART.....</b>	<b>391</b>
3.23.1. Descriptions of Peripheral registers .....	391
3.23.2. Descriptions of Peripheral functions .....	392
<b>3.24. WWDGT.....</b>	<b>440</b>
3.24.1. Descriptions of Peripheral registers .....	440
3.24.2. Descriptions of Peripheral functions .....	441
<b>3.25. USBFS.....</b>	<b>444</b>
<b>4. Revision history .....</b>	<b>445</b>

## List of Figures

Figure 2-1. File structure of firmware library of GD32F3x0 .....	24
Figure 2-2. Select peripheral example files .....	26
Figure 2-3. Copy the peripheral example files .....	27
Figure 2-4. Open the project file .....	28
Figure 2-5. Configure project files .....	28
Figure 2-6. Compile-debug-download .....	29

## List of Tables

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

Table 3-39. Function adc_oversample_mode_disable .....	55
Table 3-40. CEC Registers .....	55
Table 3-41. CEC firmware function.....	56
Table 3-42. Function cec_deinit .....	56
Table 3-43. Function cec_init .....	57
Table 3-44. Function cec_error_config.....	58
Table 3-45. Function cec_enable .....	59
Table 3-46. Function cec_disable .....	60
Table 3-47. Function cec_transmission_start .....	60
Table 3-48. Function cec_transmission_end.....	61
Table 3-49. Function cec_listen_mode_enable .....	61
Table 3-50. Function cec_listen_mode_disable.....	62
Table 3-51. Function cec_own_address_config .....	62
Table 3-52. Function cec_sft_config .....	63
Table 3-53. Function cec_generate_errorbit_config .....	64
Table 3-54. Function cec_stop_receive_bre_config .....	65
Table 3-55. Function cec_reception_tolerance_enable .....	65
Table 3-56. Function cec_reception_tolerance_disable .....	66
Table 3-57. Function cec_data_send .....	66
Table 3-58. Function cec_data_receive .....	67
Table 3-59. Function cec_interrupt_enable .....	67
Table 3-60. Function cec_interrupt_disable .....	68
Table 3-61. Function cec_flag_get.....	69
Table 3-62. Function cec_flag_clear .....	70
Table 3-63. Function cec_interrupt_flag_get.....	70
Table 3-64. Function cec_interrupt_flag_clear.....	71
Table 3-65. CMP Registers .....	72
Table 3-66. CMP firmware function .....	72
Table 3-67. Enum operating_mode_enum .....	73
Table 3-68. Enum inverting_input_enum.....	73
Table 3-69. Enum cmp_hysteresis_enum .....	73
Table 3-70. Enum cmp_output_enum .....	74
Table 3-71. Function cmp_deinit .....	74
Table 3-72. Function cmp_mode_init.....	74
Table 3-73. Function cmp_output_init .....	76
Table 3-74. Function cmp_enablet .....	77
Table 3-75. Function cmp_disable.....	77
Table 3-76. Function cmp_switch_enable.....	78
Table 3-77. Function cmp_switch_disable.....	78
Table 3-78. Function cmp_window_enable.....	79
Table 3-79. Function cmp_window_disable.....	79
Table 3-80. Function cmp_lock_enable .....	80
Table 3-81. Function cmp_output_level_get .....	80
Table 3-82. CRC Registers .....	81

---

Table 3-83. CRC firmware function .....	81
Table 3-84. Function <code>crc_deinit</code> .....	82
Table 3-85. Function <code>crc_reverse_output_data_enable</code> .....	82
Table 3-86. Function <code>crc_reverse_output_data_disable</code> .....	83
Table 3-87. Function <code>crc_data_register_reset</code> .....	83
Table 3-88. Function <code>crc_data_register_read</code> .....	84
Table 3-89. Function <code>crc_free_data_register_read</code> .....	84
Table 3-90. Function <code>crc_free_data_register_write</code> .....	85
Table 3-91. Function <code>crc_init_data_register_write</code> .....	85
Table 3-92. Function <code>crc_input_data_reverse_config</code> .....	86
Table 3-93. Function <code>crc_polynomial_size_set</code> .....	86
Table 3-94. Function <code>crc_polynomial_set</code> .....	87
Table 3-95. Function <code>crc_single_data_calculate</code> .....	87
Table 3-96. Function <code>crc_block_data_calculate</code> .....	88
Table 3-97. CTC Registers .....	89
Table 3-98. CTC firmware function .....	89
Table 3-99. Function <code>ctc_deinit</code> .....	90
Table 3-100. Function <code>ctc_refresource_polarity_config</code> .....	90
Table 3-101. Function <code>ctc_refresource_signal_select</code> .....	91
Table 3-102. Function <code>ctc_refresource_prescaler_config</code> .....	91
Table 3-103. Function <code>ctc_clock_limit_value_config</code> .....	92
Table 3-104. Function <code>ctc_counter_reload_value_config</code> .....	93
Table 3-105. Function <code>ctc_counter_enable</code> .....	93
Table 3-106. Function <code>ctc_counter_disable</code> .....	94
Table 3-107. Function <code>ctc_irc48m_trim_value_config</code> .....	94
Table 3-108. Function <code>ctc_software_refresource_pulse_generate</code> .....	95
Table 3-109. Function <code>ctc_hardware_trim_mode_config</code> .....	95
Table 3-110. Function <code>ctc_counter_capture_value_read</code> .....	96
Table 3-111. Function <code>ctc_counter_direction_read</code> .....	96
Table 3-112. Function <code>ctc_counter_reload_value_read</code> .....	97
Table 3-113. Function <code>ctc_irc48m_trim_value_read</code> .....	97
Table 3-114. Function <code>ctc_interrupt_enable</code> .....	98
Table 3-115. Function <code>ctc_interrupt_disable</code> .....	98
Table 3-116. Function <code>ctc_flag_get</code> .....	99
Table 3-117. Function <code>ctc_flag_clear</code> .....	100
Table 3-118. Function <code>ctc_interrupt_flag_get</code> .....	100
Table 3-119. Function <code>ctc_interrupt_flag_clear</code> .....	101
Table 3-120. DAC Registers .....	102
Table 3-121. DAC firmware function .....	102
Table 3-122. Function <code>dac_deinit</code> .....	103
Table 3-123. Function <code>dac_enable</code> .....	104
Table 3-124. Function <code>dac_disable</code> .....	104
Table 3-125. Function <code>dac_dma_enable</code> .....	104
Table 3-126. Function <code>dac_dma_disable</code> .....	105

Table 3-127. Function <code>dac_output_buffer_enable</code> .....	105
Table 3-128. Function <code>dac_output_buffer_disable</code> .....	106
Table 3-129. Function <code>dac_trigger_enable</code> .....	106
Table 3-130. Function <code>dac_trigger_disable</code> .....	107
Table 3-131. Function <code>dac_software_trigger_enable</code> .....	107
Table 3-132. Function <code>dac_software_trigger_disable</code> .....	108
Table 3-133. Function <code>dac_interrupt_enable</code> .....	108
Table 3-134. Function <code>dac_interrupt_disable</code> .....	109
Table 3-135. Function <code>dac_trigger_source_config</code> .....	109
Table 3-136. Function <code>dac_wave_mode_config</code> .....	110
Table 3-137. Function <code>dac_wave_bit_width_config</code> .....	111
Table 3-138. Function <code>dac_lfsr_noise_config</code> .....	111
Table 3-139. Function <code>dac_triangle_noise_config</code> .....	112
Table 3-140. Function <code>dac_output_value_get</code> .....	112
Table 3-141. Function <code>dac_flag_get</code> .....	113
Table 3-142. Function <code>dac_flag_clear</code> .....	113
Table 3-143. Function <code>dac_interrupt_flag_get</code> .....	114
Table 3-144. Function <code>dac_interrupt_flag_clear</code> .....	115
Table 3-145. Function <code>dac_data_set</code> .....	115
Table 3-146. DBG Registers .....	116
Table 3-147. DBG firmware function .....	116
Table 3-148. Enum <code>dbg_periph_enum</code> .....	116
Table 3-149. Function <code>dbg_deinit</code> .....	117
Table 3-150. Function <code>dbg_id_get</code> .....	117
Table 3-151. Function <code>dbg_low_power_enable</code> .....	118
Table 3-152. Function <code>dbg_low_power_disable</code> .....	118
Table 3-153. Function <code>dbg_periph_enable</code> .....	119
Table 3-154. Function <code>dbg_periph_disable</code> .....	120
Table 3-155. DMA Registers .....	120
Table 3-156. DMA firmware function .....	121
Table 3-157. Enum <code>dma_channel_enum</code> .....	122
Table 3-158. Structure <code>dma_parameter_struct</code> .....	122
Table 3-159. Function <code>dma_deinit</code> .....	122
Table 3-160. Function <code>dma_para_init</code> .....	123
Table 3-161. Function <code>dma_init</code> .....	123
Table 3-162. Function <code>dma_circulation_enable</code> .....	124
Table 3-163. Function <code>dma_circulation_disable</code> .....	125
Table 3-164. Function <code>dma_memory_to_memory_enable</code> .....	125
Table 3-165. Function <code>dma_memory_to_memory_disable</code> .....	126
Table 3-166. Function <code>dma_channel_enable</code> .....	126
Table 3-167. Function <code>dma_channel_disable</code> .....	127
Table 3-168. Function <code>dma_periph_address_config</code> .....	127
Table 3-169. Function <code>dma_memory_address_config</code> .....	128
Table 3-170. Function <code>dma_transfer_number_config</code> .....	129

---

Table 3-171. Function <code>dma_transfer_number_get</code> .....	129
Table 3-172. Function <code>dma_priority_config</code> .....	130
Table 3-173. Function <code>dma_memory_width_config</code> .....	131
Table 3-174. Function <code>dma_periph_width_config</code> .....	131
Table 3-175. Function <code>dma_memory_increase_enable</code> .....	132
Table 3-176. Function <code>dma_memory_increase_disable</code> .....	133
Table 3-177. Function <code>dma_periph_increase_enable</code> .....	133
Table 3-178. Function <code>dma_periph_increase_disable</code> .....	134
Table 3-179. Function <code>dma_transfer_direction_config</code> .....	134
Table 3-180. Function <code>dma_flag_get</code> .....	135
Table 3-181. Function <code>dma_flag_clear</code> .....	136
Table 3-182. Function <code>dma_interrupt_flag_get</code> .....	136
Table 3-183. Function <code>dma_interrupt_flag_clear</code> .....	137
Table 3-184. Function <code>dma_interrupt_enable</code> .....	138
Table 3-185. Function <code>dma_interrupt_disable</code> .....	138
Table 3-186. EXTI Registers.....	139
Table 3-187. Enum <code>exti_line_enum</code> .....	139
Table 3-188. Enum <code>exti_mode_enum</code> .....	140
Table 3-189. Enum <code>exti_trig_type_enum</code> .....	140
Table 3-190. EXTI firmware function .....	141
Table 3-191. Function <code>exti_deinit</code> .....	141
Table 3-192. Function <code>exti_init</code> .....	141
Table 3-193. Function <code>exti_interrupt_enable</code> .....	142
Table 3-194. Function <code>exti_interrupt_disable</code> .....	143
Table 3-195. Function <code>exti_event_enable</code> .....	143
Table 3-196. Function <code>exti_event_disable</code> .....	144
Table 3-197. Function <code>exti_software_interrupt_enable</code> .....	144
Table 3-198. Function <code>exti_software_interrupt_disable</code> .....	145
Table 3-199. Function <code>exti_flag_get</code> .....	145
Table 3-200. Function <code>exti_flag_clear</code> .....	146
Table 3-201. Function <code>exti_interrupt_flag_get</code> .....	146
Table 3-202. Function <code>exti_interrupt_flag_clear</code> .....	147
Table 3-203. FMC Registers.....	147
Table 3-204. FMC firmware function .....	148
Table 3-205. Enum <code>fmc_state_enum</code> .....	149
Table 3-206. Structure <code>ob_parm_struct</code> .....	149
Table 3-207. Function <code>fmc_unlock</code> .....	149
Table 3-208. Function <code>fmc_lock</code> .....	150
Table 3-209. Function <code>fmc_wscnt_set</code> .....	150
Table 3-210. Function <code>fmc_wscnt_set</code> .....	151
Table 3-211. Function <code>fmc_wscnt_set</code> .....	151
Table 3-212. Function <code>fmc_page_erase</code> .....	152
Table 3-213. Function <code>fmc_mass_erase</code> .....	152
Table 3-214. Function <code>fmc_word_program</code> .....	153

Table 3-215. Function fmc_halfword_program .....	153
Table 3-216. Function fmc_word_reprogram.....	154
Table 3-217. Function ob_unlock.....	154
Table 3-218. Function ob_lock .....	155
Table 3-219. Function ob_reset.....	155
Table 3-220. Function ob_erase .....	156
Table 3-221. Function ob_write_protection_enable .....	156
Table 3-222. Function ob_security_protection_config .....	157
Table 3-223. Function ob_user_write .....	157
Table 3-224. Function ob_data_program.....	158
Table 3-225. Function ob_user_get .....	159
Table 3-226. Function ob_data_get .....	159
Table 3-227. Function ob_write_protection_get.....	159
Table 3-228. Function ob_obstat_plevel_get.....	160
Table 3-229. Function fmc_interrupt_enable .....	160
Table 3-230. Function fmc_interrupt_disable .....	161
Table 3-231. Function fmc_flag_get .....	162
Table 3-232. Function fmc_flag_clear .....	162
Table 3-233. Function fmc_interrupt_flag_get .....	163
Table 3-234. Function fmc_interrupt_flag_clear .....	163
Table 3-235. Function fmc_state_get .....	164
Table 3-236. Function fmc_ready_wait .....	164
Table 3-237. Function ob_parm_get.....	165
Table 3-238. Function ob_value_modify.....	165
Table 3-239. FWDGT Registers .....	166
Table 3-240. FWDGT firmware function.....	166
Table 3-241. Function fwdgt_write_ensable .....	167
Table 3-242. Function fwdgt_write_disable .....	167
Table 3-243. Function fwdgt_enable .....	168
Table 3-244. Function fwdgt_window_value_config .....	168
Table 3-245. Function fwdgt_counter_reload .....	169
Table 3-246. Function fwdgt_config .....	169
Table 3-247. Function fwdgt_flag_get.....	170
Table 3-248. GPIO Registers.....	170
Table 3-249. GPIO firmware function .....	171
Table 3-250. Function gpio_deinit .....	171
Table 3-251. Function gpio_mode_set.....	172
Table 3-252. Function gpio_output_options_set .....	173
Table 3-253. Function gpio_bit_set .....	174
Table 3-254. Function gpio_bit_reset .....	174
Table 3-255. Function gpio_bit_write.....	175
Table 3-256. Function gpio_port_write .....	176
Table 3-257. Function gpio_input_bit_get.....	176
Table 3-258. Function gpio_input_port_get .....	177

Table 3-259. Function gpio_output_bit_get .....	177
Table 3-260. Function gpio_output_port_get .....	178
Table 3-261. Function gpio_af_set .....	178
Table 3-262. Function gpio_pin_lock .....	179
Table 3-263. Function gpio_bit_toggle .....	180
Table 3-264. Function gpio_port_toggle .....	181
Table 3-265. I2C Registers .....	181
Table 3-266. I2C firmware function .....	182
Table 3-267. Enum i2c_flag_enum .....	183
Table 3-268. Enum i2c_interrupt_flag_enum .....	183
Table 3-269. Enum i2c_interrupt_enum .....	184
Table 3-270. Function i2c_deinit .....	184
Table 3-271. Function i2c_clock_config .....	184
Table 3-272. Function i2c_mode_addr_config .....	185
Table 3-273. Function i2c_smbus_type_config .....	186
Table 3-274. Function i2c_ack_config .....	187
Table 3-275. Function i2c_ackpos_config .....	187
Table 3-276. Function i2c_master_addressing .....	188
Table 3-277. Function i2c_dualaddr_enable .....	188
Table 3-278. Function i2c_dualaddr_disable .....	189
Table 3-279. Function i2c_enable .....	190
Table 3-280. Function i2c_disable .....	190
Table 3-281. Function i2c_start_on_bus .....	191
Table 3-282. Function i2c_stop_on_bus .....	191
Table 3-283. Function i2c_data_transmit .....	192
Table 3-284. Function i2c_data_receive .....	192
Table 3-285. Function i2c_dma_enable .....	193
Table 3-286. Function i2c_dma_last_transfer_config .....	193
Table 3-287. Function i2c_stretch_scl_low_config .....	194
Table 3-288. Function i2c_slave_response_to_gcall_config .....	194
Table 3-289. Function i2c_software_reset_config .....	195
Table 3-290. Function i2c_pec_enable .....	196
Table 3-291. Function i2c_pec_transfer_enable .....	196
Table 3-292. Function i2c_pec_value_get .....	197
Table 3-293. Function i2c_smbus_issue_alert .....	197
Table 3-294. Function i2c_smbus_arp_enable .....	198
Table 3-295. Function i2c_flag_get .....	199
Table 3-296. Function i2c_flag_clear .....	200
Table 3-297. Function i2c_interrupt_enable .....	201
Table 3-298. Function i2c_interrupt_disable .....	201
Table 3-299. Function i2c_interrupt_flag_get .....	202
Table 3-300. Function i2c_interrupt_flag_clear .....	203
Table 3-301. NVIC Registers .....	204
Table 3-302. SysTick Registers .....	205

Table 3-303. Enum IRQn_Type .....	205
Table 3-304. MISC firmware function .....	206
Table 3-305. Function nvic_priority_group_set .....	206
Table 3-306. Function nvic_irq_enable.....	207
Table 3-307. Function nvic_irq_disable.....	208
Table 3-308. Function nvic_vector_table_set.....	208
Table 3-309. Function system_lowpower_set .....	209
Table 3-310. Function system_lowpower_reset.....	209
Table 3-311. Function systick_clksource_set .....	210
Table 3-312. PMU Registers.....	211
Table 3-313. PMU firmware function .....	211
Table 3-314. Function pmu_deinit .....	212
Table 3-315. Function pmu_lvd_select .....	212
Table 3-316. Function pmu_ldo_output_select.....	213
Table 3-317. Function pmu_lvd_disable.....	213
Table 3-318. Function pmu_lowdriver_mode_enable .....	214
Table 3-319. Function pmu_lowdriver_mode_disable .....	214
Table 3-320. Function pmu_highdriver_mode_enable .....	215
Table 3-321. Function pmu_highdriver_mode_disable .....	215
Table 3-322. Function pmu_highdriver_switch_select .....	216
Table 3-323. Function pmu_lowpower_driver_config.....	216
Table 3-324. Function pmu_normalpower_driver_config .....	217
Table 3-325. Function pmu_to_sleepmode .....	217
Table 3-326. Function pmu_to_deepsleepmode .....	218
Table 3-327. Function pmu_to_standbymode .....	219
Table 3-328. Function pmu_wakeup_pin_enable .....	219
Table 3-329. Function pmu_wakeup_pin_disable .....	220
Table 3-330. Function pmu_backup_write_enable .....	220
Table 3-331. Function pmu_backup_write_disable .....	221
Table 3-332. Function pmu_flag_clear.....	221
Table 3-333. Function pmu_flag_get.....	222
Table 3-334. RCU Registers .....	223
Table 3-335. RCU firmware function .....	223
Table 3-336. Enum reg_idx .....	224
Table 3-337. Enum rcu_periph_enum .....	225
Table 3-338. Enum rcu_periph_sleep_enum .....	226
Table 3-339. Enum rcu_periph_reset_enum .....	226
Table 3-340. Enum rcu_flag_enum .....	227
Table 3-341. Enum rcu_int_flag_enum .....	227
Table 3-342. Enum rcu_int_flag_clear_enum .....	228
Table 3-343. Enum rcu_int_enum .....	228
Table 3-344. Enum rcu_adc_clock_enum .....	229
Table 3-345. Enum rcu_osc_type_enum .....	229
Table 3-346. Enum rcu_clock_freq_enum.....	230

---

Table 3-347. Function rcu_deinit .....	230
Table 3-348. Function rcu_periph_clock_enable .....	231
Table 3-349. Function rcu_periph_clock_disable .....	231
Table 3-350. Function rcu_periph_clock_sleep_enable .....	232
Table 3-351. Function rcu_periph_clock_sleep_disable .....	233
Table 3-352. Function rcu_periph_reset_enable .....	233
Table 3-353. Function rcu_periph_reset_disable .....	234
Table 3-354. Function rcu_bkp_reset_enable .....	235
Table 3-355. Function rcu_bkp_reset_disable .....	236
Table 3-356. Function rcu_system_clock_source_config .....	236
Table 3-357. Function rcu_system_clock_source_get .....	237
Table 3-358. Function rcu_ahb_clock_config .....	237
Table 3-359. Function rcu_apb1_clock_config .....	238
Table 3-360. Function rcu_apb2_clock_config .....	238
Table 3-361. Function rcu_adc_clock_config .....	239
Table 3-362. Function rcu_usvfs_clock_config .....	239
Table 3-363. Function rcu_ckout_config .....	240
Table 3-364. Function rcu_pll_preselection_config .....	241
Table 3-365. Function rcu_pll_config .....	242
Table 3-366. Function rcu_usart_clock_config .....	242
Table 3-367. Function rcu_cec_clock_config .....	243
Table 3-368. Function rcu_rtc_clock_config .....	244
Table 3-369. Function rcu_ck48m_clock_config .....	244
Table 3-370. Function rcu_hxtal预分频器配置 .....	245
Table 3-371. Function rcu_lxtal驱动能力配置 .....	245
Table 3-372. Function rcu_flag_get .....	246
Table 3-373. Function rcu_all_reset_flag_clear .....	247
Table 3-374. Function rcu_interrupt_flag_get .....	247
Table 3-375. Function rcu_interrupt_flag_clear .....	248
Table 3-376. Function rcu_interrupt_enable .....	249
Table 3-377. Function rcu_interrupt_disable .....	250
Table 3-378. Function rcu_osc_stab_wait .....	250
Table 3-379. Function rcu_osc_on .....	251
Table 3-380. Function rcu_osc_off .....	252
Table 3-381. Function rcu_osc_bypass_mode_enable .....	252
Table 3-382. Function rcu_osc_bypass_mode_disable .....	253
Table 3-383. Function rcu_hxtal_clock_monitor_enable .....	253
Table 3-384. Function rcu_hxtal_clock_monitor_disable .....	254
Table 3-385. Function rcu_irc8m_adjust_value_set .....	254
Table 3-386. Function rcu_irc28m_adjust_value_set .....	255
Table 3-387. Function rcu_voltage_key_unlock .....	255
Table 3-388. Function rcu_deepsleep_voltage_set .....	256
Table 3-389. Function rcu_clock_freq_get .....	257
Table 3-390. RTC Registers .....	257

Table 3-391. RTC firmware function .....	258
Table 3-392. Structure rtc_parameter_struct .....	259
Table 3-393. Structure rtc_alarm_struct .....	259
Table 3-394. Structure rtc_timestamp_struct .....	260
Table 3-395. Structure rtc_tamper_struct .....	260
Table 3-396. Function rtc_deinit .....	260
Table 3-397. Function rtc_init .....	261
Table 3-398. Function rtc_init_mode_enter .....	261
Table 3-399. Function rtc_init_mode_exit .....	262
Table 3-400. Function rtc_register_sync_wait .....	262
Table 3-401. Function rtc_current_time_get .....	263
Table 3-402. Function rtc_subsecond_get .....	263
Table 3-403. Function rtc_alarm_config .....	264
Table 3-404. Function rtc_alarm_subsecond_config .....	264
Table 3-405. Function rtc_alarm_get .....	266
Table 3-406. Function rtc_alarm_subsecond_get .....	266
Table 3-407. Function rtc_alarm_enable .....	267
Table 3-408. Function rtc_alarm_disable .....	267
Table 3-409. Function rtc_timestamp_enable .....	268
Table 3-410. Function rtc_timestamp_disable .....	268
Table 3-411. Function rtc_timestamp_get .....	269
Table 3-412. Function rtc_timestamp_subsecond_get .....	269
Table 3-413. Function rtc_tamper_enable .....	270
Table 3-414. Function rtc_tamper_disable .....	270
Table 3-415. Function rtc_interrupt_enable .....	271
Table 3-416. Function rtc_interrupt_disable .....	271
Table 3-417. Function rtc_flag_get .....	272
Table 3-418. Function rtc_flag_clear .....	272
Table 3-419. Function rtc_alter_output_config .....	273
Table 3-420. Function rtc_calibration_config .....	274
Table 3-421. Function rtc_hour_adjust .....	275
Table 3-422. Function rtc_second_adjust .....	275
Table 3-423. Function rtc_bypass_shadow_enable .....	276
Table 3-424. Function rtc_bypass_shadow_disable .....	276
Table 3-425. Function rtc_refclock_detection_enable .....	277
Table 3-426. Function rtc_refclock_detection_disable .....	277
Table 3-427. SPI/I2S registers .....	278
Table 3-428. SPI/I2S firmware function .....	278
Table 3-429. Structure spi_parameter_struct .....	279
Table 3-430. Function spi_i2s_deinit .....	280
Table 3-431. Function spi_struct_para_init .....	280
Table 3-432. Function spi_init .....	281
Table 3-433. Function spi_enable .....	282
Table 3-434. Function spi_disable .....	282

Table 3-435. Function i2s_init .....	283
Table 3-436. Function i2s_psc_config .....	284
Table 3-437. Function i2s_enable .....	285
Table 3-438. Function i2s_disable .....	286
Table 3-439. Function spi_nss_output_enable .....	286
Table 3-440. Function spi_nss_output_disable .....	287
Table 3-441. Function spi_nss_internal_high .....	287
Table 3-442. Function spi_nss_internal_low .....	288
Table 3-443. Function spi_dma_enable .....	288
Table 3-444. Function spi_dma_disable .....	289
Table 3-445. Function spi_i2s_data_frame_format_config .....	289
Table 3-446. Function spi_i2s_data_transmit .....	290
Table 3-447. Function spi_i2s_data_receive .....	290
Table 3-448. Function spi_bidirectional_transfer_config .....	291
Table 3-449. Function spi_crc_polynomial_set .....	292
Table 3-450. Function spi_crc_polynomial_get .....	292
Table 3-451. Function spi_crc_on .....	293
Table 3-452. Function spi_crc_off .....	293
Table 3-453. Function spi_crc_next .....	294
Table 3-454. Function spi_crc_get .....	294
Table 3-455. Function spi_ti_mode_enable .....	295
Table 3-456. Function spi_ti_mode_disable .....	295
Table 3-457. Function spi_nssp_mode_enable .....	296
Table 3-458. Function spi_nssp_mode_disable .....	296
Table 3-459. Function qspi_enable .....	297
Table 3-460. Function qspi_disable .....	297
Table 3-461. Function qspi_write_enable .....	298
Table 3-462. Function qspi_read_enable .....	298
Table 3-463. Function qspi_io23_output_enable .....	299
Table 3-464. Function qspi_io23_output_disable .....	299
Table 3-465. Function spi_i2s_interrupt_enable .....	300
Table 3-466. Function spi_i2s_interrupt_disable .....	301
Table 3-467. Function spi_i2s_interrupt_flag_get .....	301
Table 3-468. Function spi_i2s_flag_get .....	302
Table 3-469. Function spi_crc_error_clear .....	303
Table 3-470. SYSCFG Registers .....	304
Table 3-471. SYSCFG firmware function .....	304
Table 3-472. Function syscfg_deinit .....	305
Table 3-473. Function syscfg_dma_remap_enable .....	305
Table 3-474. Function syscfg_dma_remap_disable .....	306
Table 3-475. Function syscfg_high_current_enable .....	306
Table 3-476. Function syscfg_high_current_disable .....	307
Table 3-477. Function syscfg_exti_line_config .....	307
Table 3-478. Function syscfg_lock_config .....	308

---

Table 3-479. Function syscfg_flag_get.....	309
Table 3-480. Function syscfg_flag_clear.....	309
Table 3-481. Function syscfg_compensation_config .....	310
Table 3-482. Function syscfg_cps_rdy_flag_get .....	310
Table 3-483. TIMERx Registers .....	311
Table 3-484. TIMERx firmware function.....	312
Table 3-485. Structure timer_parameter_struct .....	314
Table 3-486. Structure timer_break_parameter_struct .....	315
Table 3-487. Structure timer_oc_parameter_struct.....	315
Table 3-488. Structure timer_ic_parameter_struct .....	315
Table 3-489. Function timer_deinit .....	316
Table 3-490. Function timer_struct_para_init.....	316
Table 3-491. Function timer_init .....	317
Table 3-492. Function timer_enable .....	318
Table 3-493. Function timer_disable .....	318
Table 3-494. Function timer_auto_reload_shadow_enable .....	319
Table 3-495. Function timer_auto_reload_shadow_disable .....	319
Table 3-496. Function timer_update_event_enable .....	320
Table 3-497. Function timer_update_event_disable .....	320
Table 3-498. Function timer_counter_alignment .....	321
Table 3-499. Function timer_counter_up_direction .....	322
Table 3-500. Function timer_counter_down_direction .....	322
Table 3-501. Function timer_prescaler_config.....	323
Table 3-502. Function timer_repetition_value_config .....	323
Table 3-503. Function timer_autoreload_value_config .....	324
Table 3-504. Function timer_counter_value_config .....	325
Table 3-505. Function timer_counter_read .....	325
Table 3-506. Function timer_prescaler_read .....	326
Table 3-507. Function timer_single_pulse_mode_config .....	326
Table 3-508. Function timer_update_source_config.....	327
Table 3-509. Function timer_ocpre_clear_source_config .....	328
Table 3-510. Function timer_interrupt_enable .....	328
Table 3-511. Function timer_interrupt_disable.....	329
Table 3-512. Function timer_interrupt_flag_get.....	330
Table 3-513. Function timer_interrupt_flag_clear .....	331
Table 3-514. Function timer_flag_get .....	332
Table 3-515. Function timer_flag_clear .....	332
Table 3-516. Function timer_dma_enable .....	333
Table 3-517. Function timer_dma_disable .....	334
Table 3-518. Function timer_channel_dma_request_source_select.....	335
Table 3-519. Function timer_dma_transfer_config.....	335
Table 3-520. Function timer_event_software_generate.....	337
Table 3-521. Function timer_break_struct_para_init .....	338
Table 3-522. Function timer_break_config .....	338

---

Table 3-523. Function timer_break_enable .....	339
Table 3-524. Function timer_break_disable .....	340
Table 3-525. Function timer_automatic_output_enable .....	340
Table 3-526. Function timer_automatic_output_disable .....	341
Table 3-527. Function timer_primary_output_config .....	341
Table 3-528. Function timer_channel_control_shadow_config .....	342
Table 3-529. Function timer_channel_control_shadow_update_config .....	343
Table 3-530. Function timer_channel_output_struct_para_init .....	343
Table 3-531. Function timer_channel_output_config .....	344
Table 3-532. Function timer_channel_output_mode_config .....	345
Table 3-533. Function timer_channel_output_pulse_value_config .....	346
Table 3-534. Function timer_channel_output_shadow_config .....	347
Table 3-535. Function timer_channel_output_fast_config .....	348
Table 3-536. Function timer_channel_output_clear_config .....	348
Table 3-537. Function timer_channel_output_polarity_config .....	349
Table 3-538. Function timer_channel_complementary_output_polarity_config .....	350
Table 3-539. Function timer_channel_output_state_config .....	351
Table 3-540. Function timer_channel_complementary_output_state_config .....	352
Table 3-541. Function timer_channel_input_struct_para_init .....	352
Table 3-542. Function timer_input_capture_config .....	353
Table 3-543. Function timer_channel_input_capture_prescaler_config .....	354
Table 3-544. Function timer_channel_capture_value_register_read .....	355
Table 3-545. Function timer_input_pwm_capture_config .....	355
Table 3-546. Function timer_hall_mode_config .....	356
Table 3-547. Function timer_input_trigger_source_select .....	357
Table 3-548. Function timer_master_output_trigger_source_select .....	358
Table 3-549. Function timer_slave_mode_select .....	359
Table 3-550. Function timer_master_slave_mode_config .....	360
Table 3-551. Function timer_external_trigger_config .....	361
Table 3-552. Function timer_quadrature_decoder_mode_config .....	362
Table 3-553. Function timer_internal_clock_config .....	363
Table 3-554. Function timer_internal_trigger_as_external_clock_config .....	363
Table 3-555. Function timer_external_trigger_as_external_clock_config .....	364
Table 3-556. Function timer_external_clock_mode0_config .....	365
Table 3-557. Function timer_external_clock_mode1_config .....	366
Table 3-558. Function timer_external_clock_mode1_disable .....	367
Table 3-559. Function timer_channel_remap_config .....	367
Table 3-560. Function timer_write_chxval_register_config .....	368
Table 3-561. Function timer_output_value_selection_config .....	368
Table 3-562. TSI Registers .....	369
Table 3-563. TSI firmware function .....	370
Table 3-564. Function tsi_deinit .....	371
Table 3-565. Function tsi_init .....	371
Table 3-566. Function tsi_enable .....	373

Table 3-567. Function tsi_disable .....	373
Table 3-568. Function tsi_sample_pin_enable .....	374
Table 3-569. Function tsi_sample_pin_disable .....	374
Table 3-570. Function tsi_channel_pin_enable .....	375
Table 3-571. Function tsi_channel_pin_disable .....	375
Table 3-572. Function tsi_sofeware_mode_config .....	376
Table 3-573. Function tsi_software_start .....	376
Table 3-574. Function tsi_software_stop .....	377
Table 3-575. Function tsi_hardware_mode_config .....	377
Table 3-576. Function tsi_pin_mode_config .....	378
Table 3-577. Function tsi_extend_charge_config .....	378
Table 3-578. Function tsi_plus_config .....	379
Table 3-579. Function tsi_max_number_config .....	380
Table 3-580. Function tsi_hysteresis_on .....	381
Table 3-581. Function tsi_hysteresis_off .....	382
Table 3-582. Function tsi_analog_on .....	382
Table 3-583. Function tsi_analog_off .....	383
Table 3-584. Function tsi_interrupt_enable .....	383
Table 3-585. Function tsi_interrupt_disable .....	384
Table 3-586. Function tsi_interrupt_flag_clear .....	384
Table 3-587. Function tsi_interrupt_flag_get .....	385
Table 3-588. Function tsi_flag_clear .....	385
Table 3-589. Function tsi_flag_get .....	386
Table 3-590. Function tsi_group_enable .....	386
Table 3-591. Function tsi_group_disable .....	387
Table 3-592. Function tsi_group_status_get .....	387
Table 3-593. Function tsi_group0_cycle_get .....	388
Table 3-594. Function tsi_group1_cycle_get .....	388
Table 3-595. Function tsi_group2_cycle_get .....	389
Table 3-596. Function tsi_group3_cycle_get .....	390
Table 3-597. Function tsi_group4_cycle_get .....	390
Table 3-598. Function tsi_group5_cycle_get .....	391
Table 3-599. USART Registers .....	391
Table 3-600. USART firmware function .....	392
Table 3-601. Enum usart_flag_enum .....	394
Table 3-602. Enum usart_interrupt_flag_enum .....	395
Table 3-603. Enum usart_interrupt_enum .....	395
Table 3-604. Enum usart_invert_enum .....	396
Table 3-605. Function usart_deinit .....	396
Table 3-606. Function usart_baudrate_set .....	396
Table 3-607. Function usart_parity_config .....	397
Table 3-608. Function usart_word_length_set .....	398
Table 3-609. Function usart_stop_bit_set .....	398
Table 3-610. Function usart_enable .....	399

---

Table 3-611. Function usart_disable .....	399
Table 3-612. Function usart_transmit_config.....	400
Table 3-613. Function usart_receive_config .....	400
Table 3-614. Function usart_data_first_config.....	401
Table 3-615. Function usart_invert_config .....	402
Table 3-616. Function usart_overrun_enable.....	403
Table 3-617. Function usart_overrun_disable.....	403
Table 3-618. Function usart_oversample_config .....	404
Table 3-619. Function usart_sample_bit_config.....	404
Table 3-620. Function usart_receiver_timeout_enable.....	405
Table 3-621. Function usart_receiver_timeout_disable.....	405
Table 3-622. Function usart_receiver_timeout_threshold_config .....	406
Table 3-623. Function usart_data_transmit .....	406
Table 3-624. Function usart_data_receive .....	407
Table 3-625. Function usart_autobaud_detection_enable .....	408
Table 3-626. Function usart_autobaud_detection_disable .....	408
Table 3-627. Function usart_autobaud_detection_mode_config .....	409
Table 3-628. Function usart_address_config .....	409
Table 3-629. Function usart_address_detection_mode_config .....	410
Table 3-630. Function usart_mute_mode_enable.....	410
Table 3-631. Function usart_mute_mode_disable.....	411
Table 3-632. Function usart_mute_mode_wakeup_config .....	411
Table 3-633. Function usart_lin_mode_enable .....	412
Table 3-634. Function usart_lin_mode_disable .....	413
Table 3-635. Function usart_lin_break_dection_length_config .....	413
Table 3-636. Function usart_halfduplex_enable .....	414
Table 3-637. Function usart_halfduplex_disable .....	414
Table 3-638. Function usart_clock_enable .....	415
Table 3-639. Function usart_clock_disable .....	415
Table 3-640. Function usart_synchronous_clock_config .....	416
Table 3-641. Function usart_guard_time_config .....	417
Table 3-642. Function usart_smartcard_mode_enable .....	417
Table 3-643. Function usart_smartcard_mode_disable .....	418
Table 3-644. Function usart_smartcard_mode_nack_enable.....	418
Table 3-645. Function usart_smartcard_mode_nack_disable.....	419
Table 3-646. Function usart_smartcard_mode_early_nack_enable.....	419
Table 3-647. Function usart_smartcard_mode_early_nack_disable.....	420
Table 3-648. Function usart_smartcard_autoretry_config.....	420
Table 3-649. Function usart_block_length_config .....	421
Table 3-650. Function usart_irda_mode_enable.....	421
Table 3-651. Function usart_irda_mode_disable.....	422
Table 3-652. Function usart_prescaler_config.....	422
Table 3-653. Function usart_irda_lowpower_config .....	423
Table 3-654. Function usart_hardware_flow_rts_config .....	424

Table 3-655. Function usart.hardware_flow_cts_config .....	424
Table 3-656. Function usart.rs485_driver_enable .....	425
Table 3-657. Function usart.rs485_driver_disable .....	425
Table 3-658. Function usart.driver_assertime_config .....	426
Table 3-659. Function usart.driver_deassertime_config.....	426
Table 3-660. Function usart.depolarity_config .....	427
Table 3-661. Function usart.dma_receive_config.....	428
Table 3-662. Function usart.dma_transmit_config.....	428
Table 3-663. Function usart.reception_error_dma_disable .....	429
Table 3-664. Function usart.reception_error_dma_enable .....	429
Table 3-665. Function usart.wakeup_enable .....	430
Table 3-666. Function usart.wakeup_disable .....	430
Table 3-667. Function usart.wakeup_mode_config .....	431
Table 3-668. Function usart.command_enable .....	432
Table 3-669. Function usart.receive_fifo_enable.....	432
Table 3-670. Function usart.receive_fifo_disable .....	433
Table 3-671. Function usart.receive_fifo_counter_number .....	433
Table 3-672. Function usart.flag_get .....	434
Table 3-673. Function usart.flag_clear .....	435
Table 3-674. Function usart.interrupt_enable .....	436
Table 3-675. Function usart.interrupt_disable .....	437
Table 3-676. Function usart.interrupt_flag_get .....	438
Table 3-677. Function usart.interrupt_flag_clear.....	439
Table 3-678. WWDGT Registers .....	440
Table 3-679. WWDGT firmware function .....	441
Table 3-680. Function wwdgt_deinit .....	441
Table 3-681. Function wwdgt_enable .....	441
Table 3-682. Function wwdgt_counter_update .....	442
Table 3-683. Function wwdgt_config .....	442
Table 3-684. Function wwdgt_interrupt_enable .....	443
Table 3-685. Function wwdgt_flag_get .....	444
Table 3-686. Function wwdgt_flag_clear .....	444
Table 4-1. Revision history .....	445

## 1. Introduction

This manual introduces firmware library of GD32F3x0 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 GD32F3x0 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
CEC	HDMI-CEC controller
CMP	Comparator
CRC	CRC calculation unit
CTC	Clock trim controller
DAC	Digital-to-analog converter

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

## 1.1.2. Naming rules

The firmware library naming rules are shown as below:

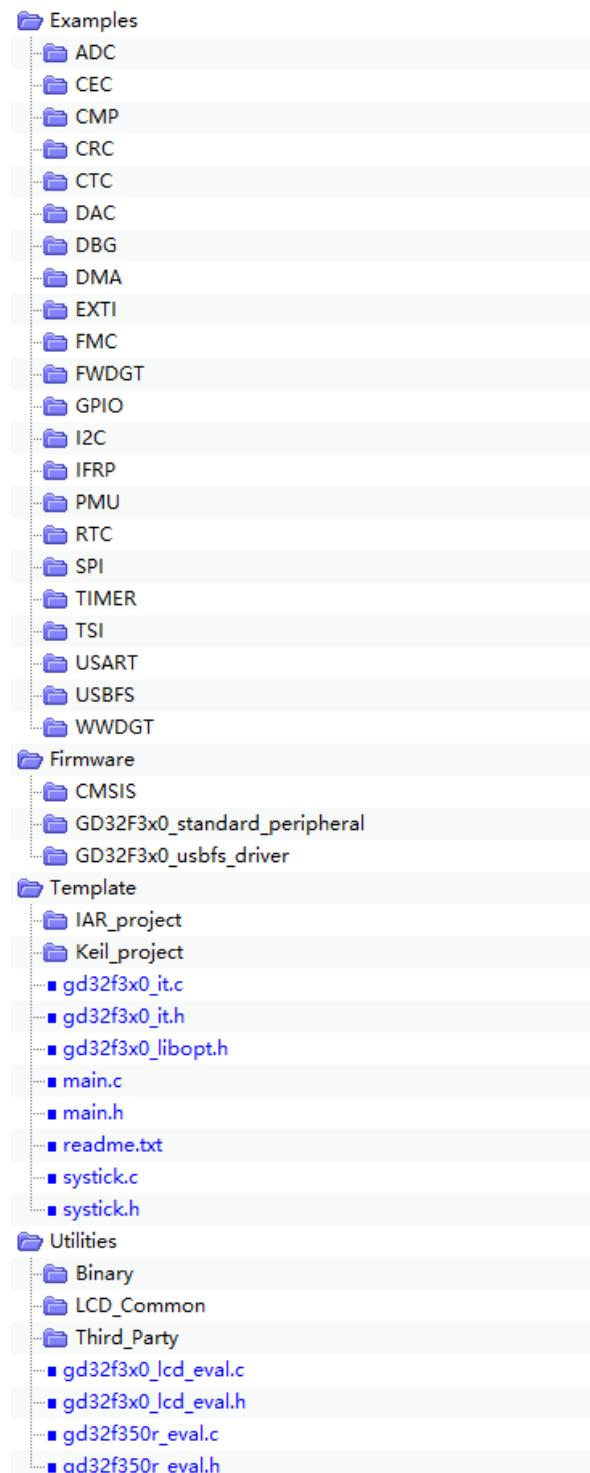
- The peripherals are shortened in XXX format, such as: ADC. More shorten information of peripherals refer to [Peripherals](#);
- The name of sourcefile and header file are started with “GD32f3x0\_”, such as: GD32f3x0\_adc.h;
- The constants used only in one file should be defined in the used file; the 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

GD32F3x0\_Firmware\_Library, the file structure is shown as below:

**Figure 2-1. File structure of firmware library of GD32F3x0**



### 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;
- GD32f3x0\_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);
- GD32f3x0\_it.c: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- GD32f3x0\_it.h: the header file include all the prototypes of the interrupt service routines;
- systick.c: the source file include the precise time delay functions by using systick;
- systick.h: the header file include the prototype of the precise time delay functions by using systick;
- main.c: example code. Note: all the examples are not influenced by software IDEs.

### 2.1.2. Firmware Folder

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

- CMSIS subfolder includes the Cortex M4 kernel support files, the startup file based on the Cortex M4 kernel processor, the global header file of GD32F3x0 and system configuration file;
- GD32F3x0\_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.

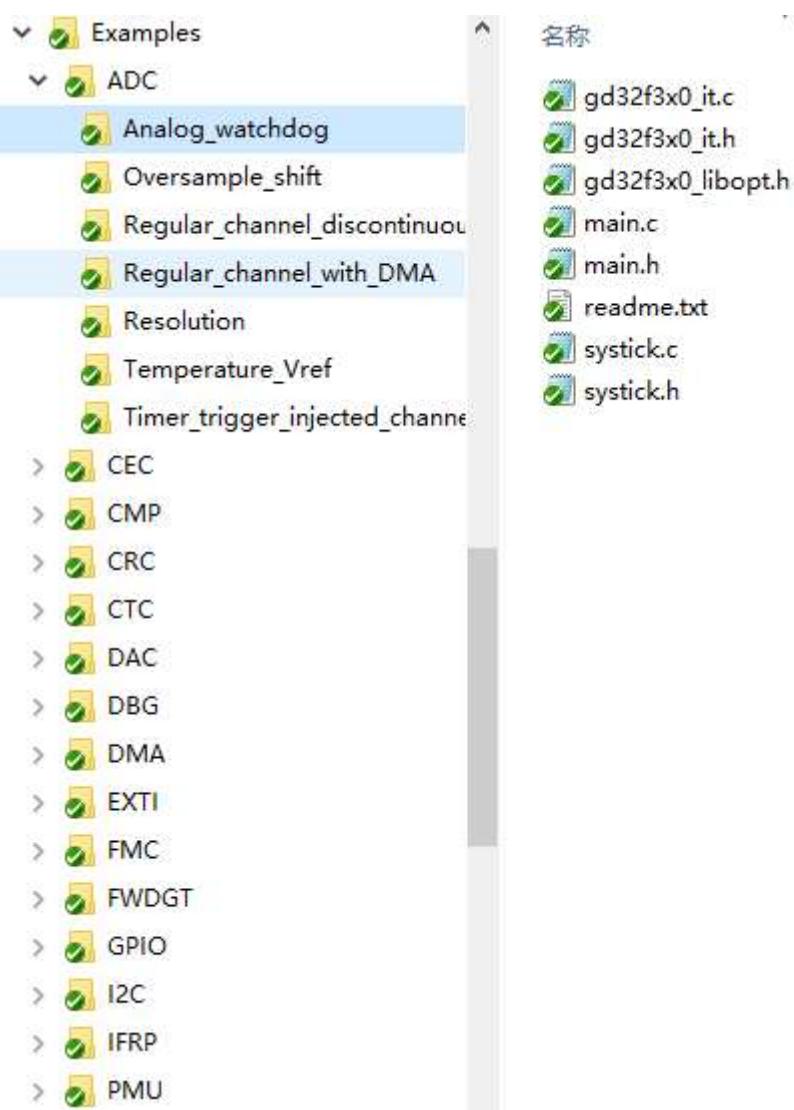
### 2.1.3. Template Folder

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 Keil4). User can use the project template to compile the firmware examples, the steps are shown as below:

#### Select files

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

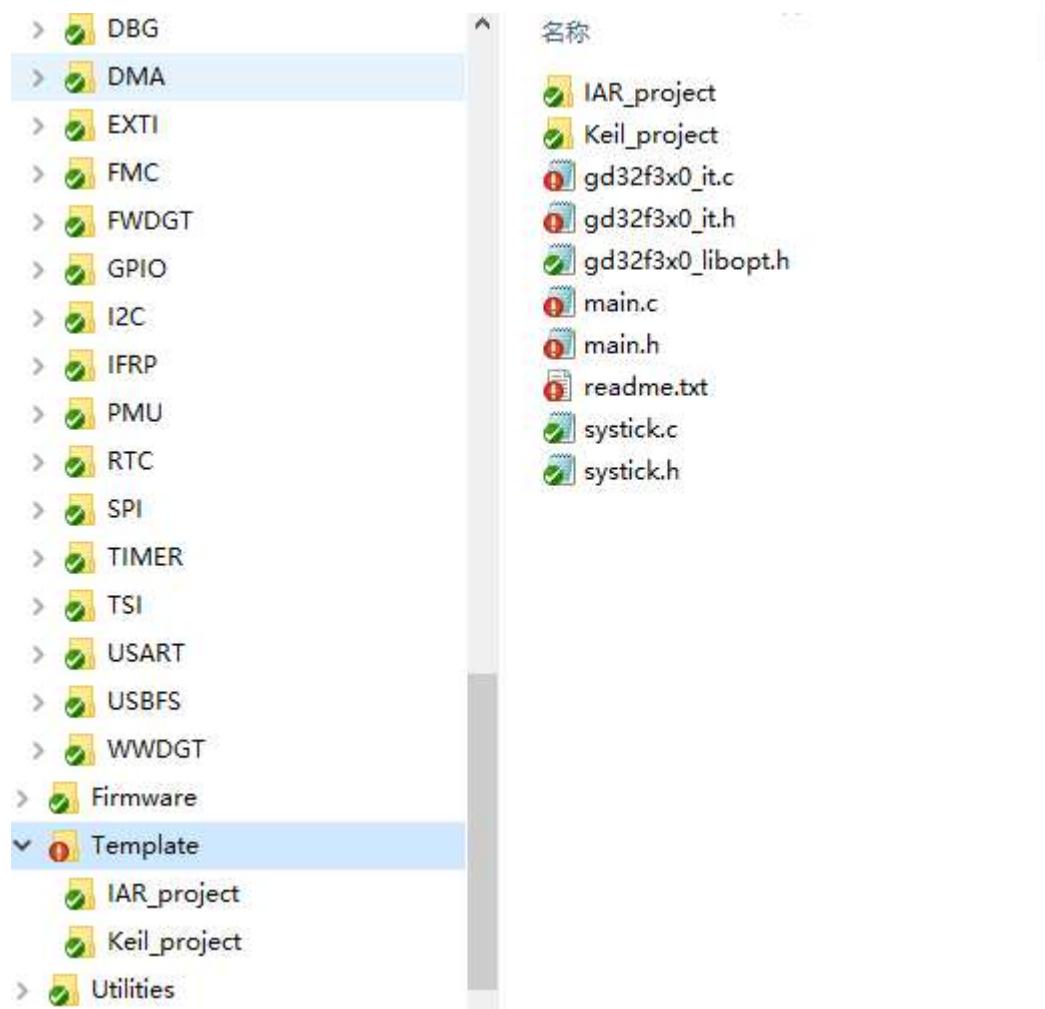
Figure 2-2. Select peripheral example files



### Copy files

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

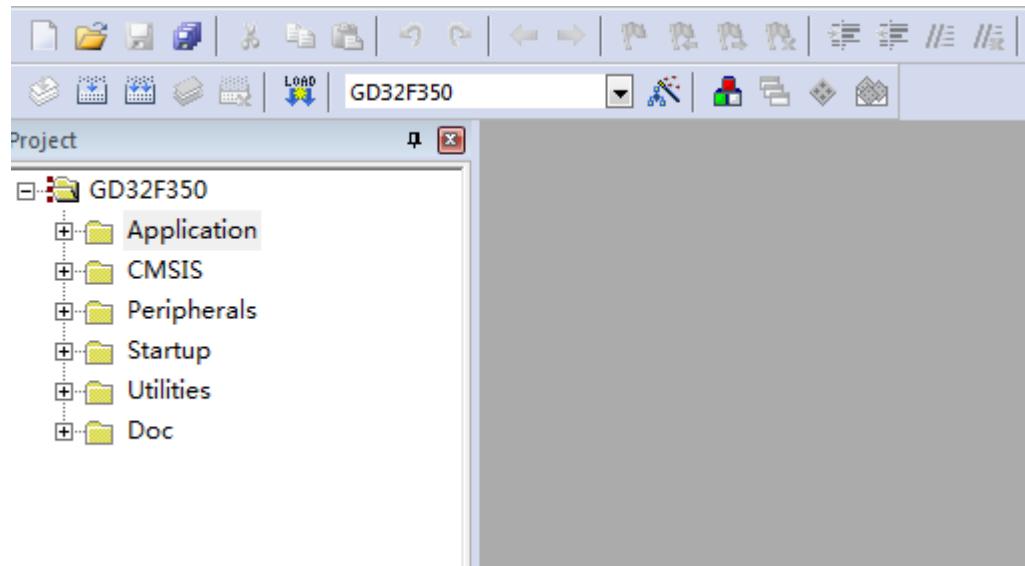
Figure 2-3. Copy the peripheral example files



### Open a project

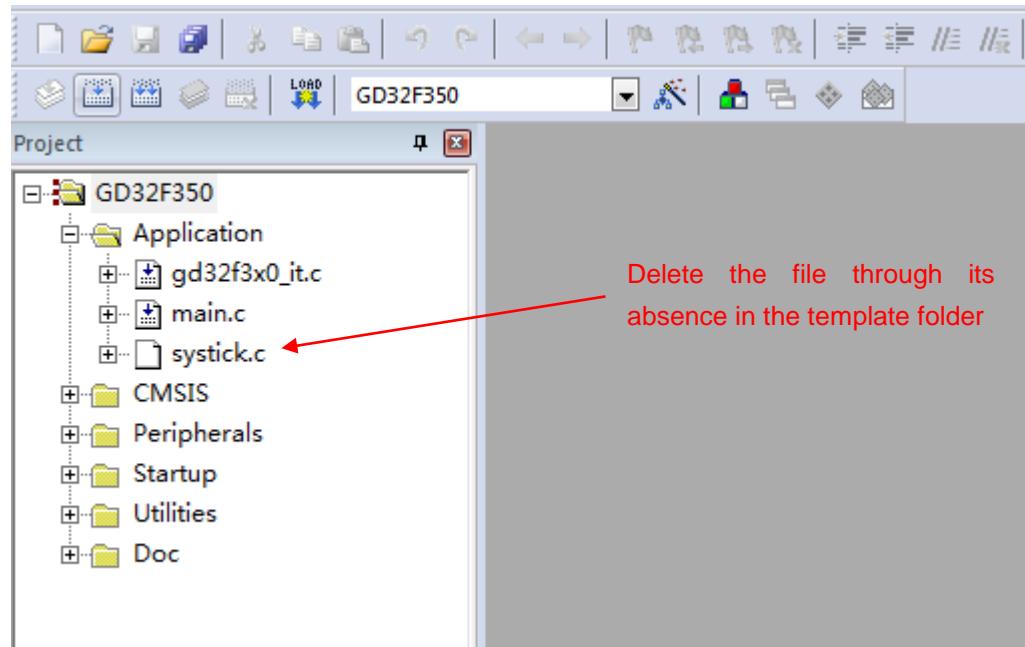
GD provides project in Keil and IAR, users can open project in different IDEs according to their need, such as "Keil\_project", open \Template\Keil\_project\Project.uvproj, shown as below:

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



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

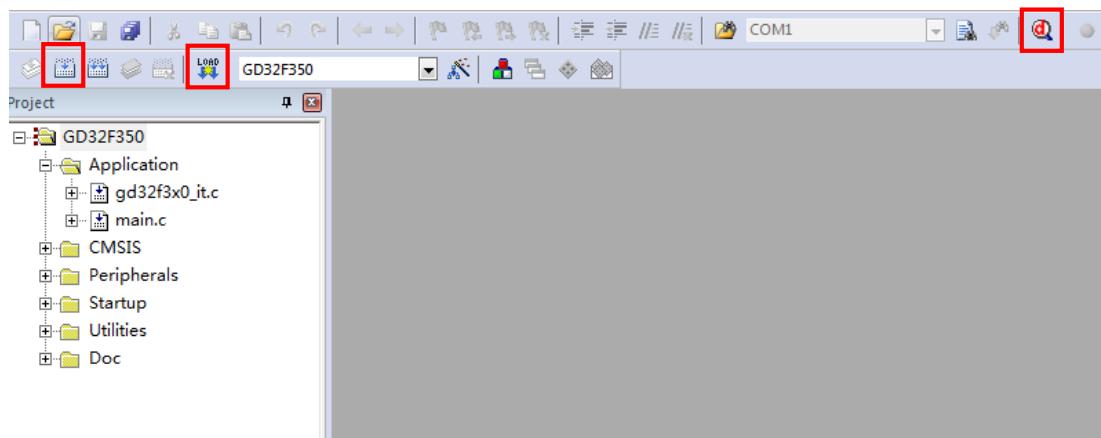
**Figure 2-5. Configure project files**



## Compile-Debug-Download

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

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



## 2.1.4. Utilities Folder

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

- Binary, LCD\_Common and Third\_Party subfolders;
- GD32f3x0r\_eval.h and GD32f3x0\_lcd\_eval.h are related header files of the evaluation board about running the firmware examples;
- GD32f3x0r\_eval.c and GD32f3x0\_lcd\_eval.c are related source files of the evaluation board about running the firmware examples.

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

## 2.2. File descriptions of Firmware Library

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

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

Files	Descriptions
gd32f3x0_libopt.h	The header file about all the header files of peripherals. It is the only one file which is necessity to be included in the user's application, to connect the firmware library and the application.
main.c	Example of main function.
gd32f3x0_it.h	Header file, including all the prototypes of interrupt service routines.
gd32f3x0_it.c	Source files about interrupt service routines of peripherals. User can written his own interrupt functions in this file. For the different interrupt service requests to the same interrupt vector, users can confirm the interrupt source by functions of judging interrupt flags of peripherals. The functions are included in the firmware library.
gd32f3x0_xxx.h	The header file of peripheral xxx, including functions about peripheral xxx, and the variables used for functions.



## GD32F3x0 Firmware Library User Guide

Files	Descriptions
gd32f3x0_xxx.c	The C source file for driving peripheral xxx.
systick.h	The header file of systick.c, including prototypes of systick configuration function and delay function.
systick.c	The source file about systick configuration function and delay function.
readme.txt	Description document about how to configure and how to use the firmware example.

### 3. Firmware Library of Standard Peripherals

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

The description format of firmware functions are shown as below:

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

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

#### 3.2. ADC

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

##### 3.2.1. Descriptions of Peripheral registers

ADC registers are listed in the table shown as below:

**Table 3-2. ADC Registers**

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

Registers	Descriptions
ADC_RSQ0	Regular sequence register 0
ADC_RSQ1	Regular sequence register 1
ADC_RSQ2	Regular sequence register 2
ADC_ISQ	Inserted sequence register
ADC_IDATAx	Inserted data register x(x=0..3)
ADC_RDATA	Regular data register
ADC_OVSAMPCTL	Oversample control register

### 3.2.2. Descriptions of Peripheral functions

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

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

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

Function name	Function description
adc_interrupt_disable	disable ADC interrupt
adc_watchdog_single_channel_enable	configure ADC analog watchdog single channel
adc_watchdog_group_channel_enable	configure ADC analog watchdog group channel
adc_watchdog_disable	disable ADC analog watchdog
adc_watchdog_threshold_config	configure ADC analog watchdog threshold
adc_resolution_config	configure ADC resolution
adc_oversample_mode_config	configure ADC oversample mode
adc_oversample_mode_enable	enable ADC oversample mode
adc_oversample_mode_disable	disable ADC oversample mode

## adc\_deinit

The description of adc\_deinit is shown as below:

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

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

Example:

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

## adc\_enable

The description of adc\_enable is shown as below:

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

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

Return value	
-	-

Example:

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

### **adc\_disable**

The description of adc\_disable is shown as below:

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

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

Example:

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

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

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

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

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

Example:

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

```
adc_calibration_enable();
```

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

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

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

<b>Function name</b>	adc_dma_mode_enable
<b>Function prototype</b>	void adc_dma_mode_enable(void);
<b>Function descriptions</b>	enable ADC DMA request
<b>Precondition</b>	-
<b>The called functions</b>	-
<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 3-9. Function adc\_dma\_mode\_disable**

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

Example:

```
/* disable ADC DMA request */

adc_dma_mode_disable();
```

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

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

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

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

Example:

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

adc_tempsensor_vrefint_enable();
```

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

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

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

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

Example:

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

adc_tempsensor_vrefint_disable();
```

### **adc\_vbat\_enable**

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

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

<b>Function name</b>	adc_vbat_enable
----------------------	-----------------

<b>Function prototype</b>	void adc_vbat_enable(void);
<b>Function descriptions</b>	enable the vbat channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the vbat channel */

adc_vbat_enable();
```

### **adc\_vbat\_disable**

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

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

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

Example:

```
/* disable the vbat channel */

adc_vbat_disable();
```

### **adc\_discontinuous\_mode\_config**

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

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

<b>Function name</b>	adc_discontinuous_mode_config
<b>Function prototype</b>	void adc_discontinuous_mode_config(uint8_t channel_group, uint8_t length);

<b>Function descriptions</b>	configure ADC discontinuous mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_channel_group</b>	select the channel group
<b>ADC_REGULAR_CHANNEL</b>	regular channel group
<b>ADC_INSERTED_CHANNEL</b>	inserted channel group
<b>ADC_CHANNEL_DISCON_DISABLE</b>	disable discontinuous mode of regular and inserted channel
<b>Input parameter{in}</b>	
<b>length</b>	number of conversions in discontinuous mode, the number can be 1..8 for regular channel, the number has no effect for inserted channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### adc\_special\_function\_config

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

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

<b>Function name</b>	adc_special_function_config
<b>Function prototype</b>	void adc_special_function_config(uint32_t function, ControlStatus newvalue);
<b>Function descriptions</b>	enable or disable ADC special function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>function</b>	the function to config
<b>ADC_SCAN_MODE</b>	scan mode select
<b>ADC_INSERTED_CHANNEL_AUTO</b>	inserted channel group convert automatically
<b>ADC_CONTINUOUS_MODE</b>	continuous mode select
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value

<b>ENABLE</b>	enable function
<b>DISABLE</b>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC scan mode */
adc_special_function_config(ADC_SCAN_MODE, ENABLE);
```

### adc\_data\_alignment\_config

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

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

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

Example:

```
/* configure ADC data alignment */
adc_data_alignment_config(ADC_DATAALIGN_RIGHT);
```

### adc\_channel\_length\_config

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

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

<b>Function name</b>	adc_channel_length_config
<b>Function prototype</b>	void adc_channel_length_config(uint8_t channel_group, uint32_t length);

<b>Function descriptions</b>	configure the length of regular channel group or inserted channel group
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channel_group</b>	select the channel group
<b>ADC_REGULAR_CHANNEL</b>	regular channel group
<b>ADC_INSERTED_CHANNEL</b>	inserted channel group
<b>Input parameter{in}</b>	
<b>length</b>	the length of the channel, regular channel 1-16, inserted channel 1-4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the length of ADC regular channel */
adc_channel_length_config(ADC_REGULAR_CHANNEL, 4);
```

### adc\_regular\_channel\_config

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

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

<b>Function name</b>	adc_regular_channel_config
<b>Function prototype</b>	void adc_regular_channel_config(uint8_t rank, uint8_t channel, uint32_t sample_time);
<b>Function descriptions</b>	configure ADC regular channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rank</b>	the regular group sequence rank, this parameter must be between 0 to 15
<b>Input parameter{in}</b>	
<b>channel</b>	the selected ADC channel
<b>ADC_CHANNEL_x</b>	ADC Channelx (x=0..9,16,17)
<b>Input parameter{in}</b>	
<b>sample_time</b>	the sample time value
<b>ADC_SAMPLETIME_1POINT5</b>	1.5 cycles
<b>ADC_SAMPLETIME_7POINT5</b>	7.5 cycles
<b>ADC_SAMPLETIME_</b>	13.5 cycles

<code>13POINT5</code>	
<code>ADC_SAMPLETIME_28POINT5</code>	28.5 cycles
<code>ADC_SAMPLETIME_41POINT5</code>	41.5 cycles
<code>ADC_SAMPLETIME_55POINT5</code>	55.5 cycles
<code>ADC_SAMPLETIME_71POINT5</code>	71.5 cycles
<code>ADC_SAMPLETIME_239POINT5</code>	239.5 cycles
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

The description of `adc_inserted_channel_config` is shown as below:

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

<b>Function name</b>	<code>adc_inserted_channel_config</code>
<b>Function prototype</b>	<code>void adc_inserted_channel_config(uint8_t rank, uint8_t channel, uint32_t sample_time);</code>
<b>Function descriptions</b>	configure ADC inserted channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rank</b>	the inserted group sequencer rank, this parameter must be between 0 to 3
<b>Input parameter{in}</b>	
<b>channel</b>	the selected ADC channel
<code>ADC_CHANNEL_X</code>	ADC Channelx (x=0..9,16,17)
<b>Input parameter{in}</b>	
<b>sample_time</b>	the sample time value
<code>ADC_SAMPLETIME_1POINT5</code>	1.5 cycles
<code>ADC_SAMPLETIME_7POINT5</code>	7.5 cycles
<code>ADC_SAMPLETIME_13POINT5</code>	13.5 cycles

13POINT5	
ADC_SAMPLETIME_28POINT5	28.5 cycles
ADC_SAMPLETIME_41POINT5	41.5 cycles
ADC_SAMPLETIME_55POINT5	55.5 cycles
ADC_SAMPLETIME_71POINT5	71.5 cycles
ADC_SAMPLETIME_239POINT5	239.5 cycles
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

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

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

Example:

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

adc_inserted_channel_offset_config(ADC_INSERTED_CHANNEL_0, 100);
```

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

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

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

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

Example:

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

adc_external_trigger_config(ADC_INSERTED_CHANNEL_0, ENABLE);
```

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

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

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

<b>Function name</b>	adc_external_trigger_source_config
<b>Function prototype</b>	void adc_external_trigger_source_config(uint8_t channel_group, uint32_t external_trigger_source);
<b>Function descriptions</b>	configure ADC external trigger source
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>channel_group</b>	select the channel group
<b>ADC_REGULAR_CHANNEL</b>	regular channel group
<b>ADC_INSERTED_CHANNEL</b>	inserted channel group
Input parameter{in}	
<b>external_trigger_source</b>	regular or inserted group trigger source
<b>ADC_EXTTRIG_REGULAR_T0_CH0</b>	TIMER0 CH0 event select for regular channel
<b>ADC_EXTTRIG_REGULAR_T0_CH1</b>	TIMER0 CH1 event select for regular channel
<b>ADC1_EXTTRIG_REGULAR_T0_CH2</b>	TIMER0 CH2 event select for regular channel
<b>ADC_EXTTRIG_REGULAR_T1_CH1</b>	TIMER1 CH1 event select for regular channel
<b>ADC_EXTTRIG_REGULAR_T2_TRGO</b>	TIMER2 TRGO event select for regular channel
<b>ADC_EXTTRIG_REGULAR_T14_CH0</b>	TIMER14 CH0 event select for regular channel
<b>ADC_EXTTRIG_REGULAR EXTI_11</b>	external interrupt line 11 for regular channel
<b>ADC_EXTTRIG_REGULAR_NONE</b>	software trigger for regular channel
<b>ADC_EXTTRIG_INSERTED_T0_TRGO</b>	TIMER0 TRGO event select for inserted channel
<b>ADC_EXTTRIG_INSERTED_T0_CH3</b>	TIMER0 CH3 event select for inserted channel
<b>ADC_EXTTRIG_INSERTED_T1_TRGO</b>	TIMER1 TRGO event select for inserted channel
<b>ADC_EXTTRIG_INSERTED_T1_CH0</b>	TIMER1 CH0 event select for inserted channel
<b>ADC_EXTTRIG_INSERTED_T2_CH3</b>	TIMER2 CH3 event select for inserted channel
<b>ADC_EXTTRIG_INSERTED_T14_TRGO</b>	TIMER14 TRGO event select for inserted channel
<b>ADC_EXTTRIG_INSERTED_EXTI_15</b>	external interrupt line 15 for inserted channel
<b>ADC_EXTTRIG_INSERTED_NONE</b>	software trigger for inserted channel
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* configure ADC regular channel external trigger source */
adc_external_trigger_source_config(ADC_REGULAR_CHANNEL,
ADC_EXTTRIG_REGULAR_T0_CH0);
```

### adc\_software\_trigger\_enable

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

**Table 3-23. Function adc\_software\_trigger\_enable**

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

Example:

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

### adc\_regular\_data\_read

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

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

<b>Function name</b>	adc_regular_data_read
<b>Function prototype</b>	uint16_t adc_regular_data_read(void);
<b>Function descriptions</b>	read ADC regular group data register
<b>Precondition</b>	-
<b>The called functions</b>	-

Output parameter{out}	
-	-
Return value	
uint16_t	ADC conversion value (0-0xFFFF)

Example:

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

uint16_t adc_value = 0;

adc_value = adc_regular_data_read();
```

### adc\_inserted\_data\_read

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

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

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

Example:

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

uint16_t adc_value = 0;

adc_value = adc_inserted_data_read (ADC_INSERTED_CHANNEL_0);
```

### adc\_flag\_get

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

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

Function name	adc_flag_get
Function prototype	FlagStatus adc_flag_get(uint32_t flag);
Function descriptions	get the ADC flag bits
Precondition	-

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

Example:

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

### adc\_flag\_clear

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

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

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

Example:

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

```
adc_flag_clear(ADC_FLAG_WDE);
```

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

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

**Table 3-28. Function adc\_interrupt\_flag\_get**

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

Example:

```
/* get the ADC analog watchdog interrupt bits*/
FlagStatus flag_value;
flag_value = adc_interrupt_flag_get(ADC_INT_FLAG_WDE);
```

### **adc\_interrupt\_flag\_clear**

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

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

<b>Function name</b>	adc_interrupt_flag_clear
<b>Function prototype</b>	void adc_interrupt_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear the ADC interrupt bits
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	the adc interrupt bits
<b>ADC_INT_FLAG_WDE</b>	analog watchdog interrupt
<b>ADC_INT_FLAG_EOC</b>	end of group conversion interrupt
<b>ADC_INT_FLAG_EOIC</b>	end of inserted group conversion interrupt
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

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

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

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

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

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

Example:

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

### **adc\_interrupt\_disable**

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

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

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

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

Example:

```
/* disable ADC interrupt */

adc_interrupt_disable(ADC_INT_WDE);
```

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

The description of `adc_watchdog_single_channel_enable` is shown as below:

**Table 3-32. Function `adc_watchdog_single_channel_enable`**

<b>Function name</b>	adc_watchdog_single_channel_enable
<b>Function prototype</b>	void adc_watchdog_single_channel_enable(uint8_t channel);
<b>Function descriptions</b>	configure ADC analog watchdog single channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channel</b>	the selected ADC channel
<i>ADC_CHANNEL_x</i>	ADC Channelx(x=0..9,16,17)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

adc_watchdog_single_channel_enable(ADC_CHANNEL_1);
```

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

The description of `adc_watchdog_group_channel_enable` is shown as below:

**Table 3-33. Function `adc_watchdog_group_channel_enable`**

<b>Function name</b>	adc_watchdog_group_channel_enable
<b>Function prototype</b>	void adc_watchdog_group_channel_enable(uint8_t channel_group);
<b>Function descriptions</b>	configure ADC analog watchdog group channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

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

Example:

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

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

The description of `adc_watchdog_disable` is shown as below:

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

<b>Function name</b>	adc_watchdog_disable
<b>Function prototype</b>	void adc_watchdog_disable(void);
<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 ADC0 analog watchdog */
adc_watchdog_disable(ADC0);
```

### **adc\_watchdog\_threshold\_config**

The description of `adc_watchdog_threshold_config` is shown as below:

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

<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>	
low_threshold	analog watchdog low threshold, 0..4095
<b>Input parameter{in}</b>	
high_threshold	analog watchdog high threshold, 0..4095
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC analog watchdog threshold */
adc_watchdog_threshold_config(0x0400, 0xA00);
```

### adc\_resolution\_config

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

**Table 3-36. 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>	
resolution	ADC resolution
ADC_RESOLUTION_12B	12-bit ADC resolution
ADC_RESOLUTION_10B	10-bit ADC resolution
ADC_RESOLUTION_8B	8-bit ADC resolution
ADC_RESOLUTION_6B	6-bit ADC resolution
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure ADC resolution */

adc_resolution_config(ADC_RESOLUTION_12B);

```

### **adc\_oversample\_mode\_config**

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

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

<b>Function name</b>	adc_oversample_mode_config
<b>Function prototype</b>	void adc_oversample_mode_config(uint32_t 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
<i>ADC_OVERSAMPLING_ALL_CONVERT</i>	all oversampled conversions for a channel are done consecutively after a trigger
<i>ADC_OVERSAMPLING_ONE_CONVERT</i>	each oversampled conversion for a channel needs a trigger
<b>Input parameter{in}</b>	
<b>shift</b>	ADC oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_NONE</i>	no oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_1B</i>	1-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_2B</i>	2-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_3B</i>	3-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_4B</i>	4-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_5B</i>	5-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_6B</i>	6-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_7B</i>	7-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_8B</i>	8-bit oversampling shift
<b>Input parameter{in}</b>	
<b>ratio</b>	ADC oversampling ratio
<i>ADC_OVERSAMPLING</i>	oversampling ratio multiple 2

<code>_RATIO_MUL2</code>	
<code>ADC_OVERSAMPLING_RATIO_MUL4</code>	oversampling ratio multiple 4
<code>ADC_OVERSAMPLING_RATIO_MUL8</code>	oversampling ratio multiple 8
<code>ADC_OVERSAMPLING_RATIO_MUL16</code>	oversampling ratio multiple 16
<code>ADC_OVERSAMPLING_RATIO_MUL32</code>	oversampling ratio multiple 32
<code>ADC_OVERSAMPLING_RATIO_MUL64</code>	oversampling ratio multiple 64
<code>ADC_OVERSAMPLING_RATIO_MUL128</code>	oversampling ratio multiple 128
<code>ADC_OVERSAMPLING_RATIO_MUL256</code>	oversampling ratio multiple 256
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

The description of `adc_oversample_mode_enable` is shown as below:

**Table 3-38. Function `adc_oversample_mode_enable`**

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

Example:

```
/* enable ADC oversample mode */
```

```
adc_oversample_mode_enable();
```

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

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

**Table 3-39. 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>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable ADC oversample mode */
```

```
adc_oversample_mode_disable();
```

## **3.3. CEC**

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

### **3.3.1. Descriptions of Peripheral registers**

CEC registers are listed in the table shown as below:

**Table 3-40. CEC Registers**

<b>Registers</b>	<b>Descriptions</b>
CEC_CTL	Control register
CEC_CFG	Configuration register
CEC_TDATA	Transmit data register
CEC_RDATA	Receive data register
CEC_INTF	Interrupt Flag Register

Registers	Descriptions
CEC_INTEN	Interrupt enable register

### 3.3.2. Descriptions of Peripheral functions

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

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

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

#### **cec\_deinit**

The description of cec\_deinit is shown as below:

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

Function name	cec_deinit
Function prototype	void cec_deinit(void);

<b>Function descriptions</b>	reset HDMI-CEC 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 CEC */
```

```
cec_deinit();
```

### cec\_init

The description of cec\_init is shown as below:

**Table 3-43. Function cec\_init**

<b>Function name</b>	cec_init
<b>Function prototype</b>	void cec_init(uint32_t sftmopt, uint32_t sft, uint32_t address);
<b>Function descriptions</b>	configure signal free time,the signal free time counter start option,own address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sftmopt</b>	signal free time counter start option
<b>CEC_SFT_START_ST AOM</b>	signal free time counter starts counting when STAOM is asserted
<b>CEC_SFT_START_LA ST</b>	signal free time counter starts automatically after transmission/reception end
<b>Input parameter{in}</b>	
<b>sft</b>	signal free time
<b>CEC_SFT_PROTOCOL _PERIOD</b>	the signal free time will perform as HDMI-CEC protocol description
<b>CEC_SFT_1POINT5_P ERIOD</b>	1.5 nominal data bit periods
<b>CEC_SFT_2POINT5_P ERIOD</b>	2.5 nominal data bit periods
<b>CEC_SFT_3POINT5_P ERIOD</b>	3.5 nominal data bit periods
<b>CEC_SFT_4POINT5_P ERIOD</b>	4.5 nominal data bit periods

<i>CEC_SFT_5POINT5_P ERIOD</i>	5.5 nominal data bit periods
<i>CEC_SFT_6POINT5_P ERIOD</i>	6.5 nominal data bit periods
<i>CEC_SFT_7POINT5_P ERIOD</i>	7.5 nominal data bit periods
<b>Input parameter{in}</b>	
<b>address</b>	own address
<i>CEC_OWN_ADDRESS _CLEAR</i>	own address is cleared
<i>CEC_OWN_ADDRESS x</i>	own address is x
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* init CEC*/
cec_init(CEC_SFT_START_STAOM, CEC_SFT_PROTOCOL_PERIOD,
CEC_OWN_ADDRESS0);
```

### cec\_error\_config

The description of `cec_error_config` is shown as below:

**Table 3-44. Function `cec_error_config`**

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

<i>ERROR_BIT_ON</i>	
<i>CEC_LONG_PERIOD_ERROR_BIT_OFF</i>	do not generate Error-bit on long bit period error
<b>Input parameter{in}</b>	
<i>singlecast_bre</i>	generate an Error-bit when detected BRE in singlecast
<i>CEC_RISING_PERIOD_ERROR_BIT_ON</i>	generate Error-bit on bit rising error
<i>CEC_RISING_PERIOD_ERROR_BIT_OFF</i>	do not generate Error-bit on bit rising error
<b>Input parameter{in}</b>	
<i>rxbrestp</i>	whether stop receive message when detected BRE
<i>CEC_STOP_RISING_ERROR_BIT_ON</i>	stop reception when detected bit rising error
<i>CEC_STOP_RISING_ERROR_BIT_OFF</i>	do not stop reception when detected bit rising error
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### **cec\_enable**

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

**Table 3-45. Function **cec\_enable****

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

Example:

```
/* enable HDMI-CEC controller */

cec_enable();
```

### **cec\_disable**

The description of `cec_disable` is shown as below:

**Table 3-46. Function `cec_disable`**

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

Example:

```
/* disable HDMI-CEC controller */

cec_disable();
```

### **cec\_transmission\_start**

The description of `cec_transmission_start` is shown as below:

**Table 3-47. Function `cec_transmission_start`**

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

Example:

```
/* start CEC message transmission */
```

```
cec_transmission_start();
```

### **cec\_transmission\_end**

The description of `cec_transmission_end` is shown as below:

**Table 3-48. Function `cec_transmission_end`**

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

Example:

```
/* end CEC message transmission */
```

```
cec_transmission_end();
```

### **cec\_listen\_mode\_enable**

The description of `cec_listen_mode_enable` is shown as below:

**Table 3-49. Function `cec_listen_mode_enable`**

<b>Function name</b>	cec_listen_mode_enable
<b>Function prototype</b>	void cec_listen_mode_enable (void);
<b>Function descriptions</b>	enable CEC listen 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 CEC listen mode */
```

---

```
cec_listen_mode_enable();
```

### **cec\_listen\_mode\_disable**

The description of `cec_listen_mode_disable` is shown as below:

**Table 3-50. Function `cec_listen_mode_disable`**

<b>Function name</b>	cec_listen_mode_disable
<b>Function prototype</b>	void cec_listen_mode_disable (void);
<b>Function descriptions</b>	disable CEC listen 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 CEC listen mode */

cec_listen_mode_disable();
```

### **cec\_own\_address\_config**

The description of `cec_own_address_config` is shown as below:

**Table 3-51. Function `cec_own_address_config`**

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

Example:

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

```
cec_own_address_config (CEC_OWN_ADDRESS_CLEAR);
```

### **cec\_sft\_config**

The description of `cec_sft_config` is shown as below:

**Table 3-52. Function `cec_sft_config`**

<b>Function name</b>	cec_sft_config
<b>Function prototype</b>	void cec_sft_config(uint32_t sftmopt, uint32_t sft);
<b>Function descriptions</b>	configure signal free time and the signal free time counter start option
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>sftmopt</code>	signal free time counter start option
<code>CEC_SFT_START_ST_AOM</code>	signal free time counter starts counting when STAOM is asserted
<code>CEC_SFT_START_LA_ST</code>	signal free time counter starts automatically after transmission/reception end
<b>Input parameter{in}</b>	
<code>sft</code>	signal free time
<code>CEC_SFT_PROTOCOL_PERIOD</code>	the signal free time will perform as HDMI-CEC protocol description
<code>CEC_SFT_1POINT5_PERIOD</code>	1.5 nominal data bit periods
<code>CEC_SFT_2POINT5_PERIOD</code>	2.5 nominal data bit periods
<code>CEC_SFT_3POINT5_PERIOD</code>	3.5 nominal data bit periods
<code>CEC_SFT_4POINT5_PERIOD</code>	4.5 nominal data bit periods
<code>CEC_SFT_5POINT5_PERIOD</code>	5.5 nominal data bit periods
<code>CEC_SFT_6POINT5_PERIOD</code>	6.5 nominal data bit periods
<code>CEC_SFT_7POINT5_PERIOD</code>	7.5 nominal data bit periods
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

cec_sft_config (CEC_SFT_START_STAOM, CEC_SFT_PROTOCOL_PERIOD);
```

### **cec\_generate\_errorbit\_config**

The description of `cec_generate_errorbit_config` is shown as below:

**Table 3-53. Function `cec_generate_errorbit_config`**

<b>Function name</b>	cec_generate_errorbit_config
<b>Function prototype</b>	void cec_generate_errorbit_config(uint32_t broadcast, uint32_t singlecast_lbpe, uint32_t singlecast_bre);
<b>Function descriptions</b>	configure generate Error-bit when detected some abnormal situation or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>broadcast</b>	generate Error-bit in broadcast or not
<i>CEC_BROADCAST_ERROR_BIT_ON</i>	generate Error-bit in broadcast
<i>CEC_BROADCAST_ERROR_BIT_OFF</i>	do not generate Error-bit in broadcast
<b>Input parameter{in}</b>	
<b>singlecast_lbpe</b>	generate Error-bit on long bit period error or not
<i>CEC_LONG_PERIOD_ERROR_BIT_ON</i>	generate Error-bit on long bit period error
<i>CEC_LONG_PERIOD_ERROR_BIT_OFF</i>	do not generate Error-bit on long bit period error
<b>Input parameter{in}</b>	
<b>singlecast_bre</b>	generate Error-bit on bit rising error or not
<i>CEC_RISING_PERIOD_ERROR_BIT_ON</i>	generate Error-bit on bit rising error
<i>CEC_RISING_PERIOD_ERROR_BIT_OFF</i>	do not generate Error-bit on bit rising error
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure generate Error-bit when detected some abnormal situation or not */

cec_generate_errorbit_config (CEC_BROADCAST_ERROR_BIT_ON,
CEC_LONG_PERIOD_ERROR_BIT_ON, CEC_RISING_PERIOD_ERROR_BIT_ON);
```

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

The description of `cec_stop_receive_bre_config` is shown as below:

**Table 3-54. Function `cec_stop_receive_bre_config`**

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

Example:

```
/* whether stop receive message when detected bit rising error */
cec_stop_receive_bre_config (CEC_STOP_RISING_ERROR_BIT_ON);
```

### **cec\_reception\_tolerance\_enable**

The description of `cec_reception_tolerance_enable` is shown as below:

**Table 3-55. Function `cec_reception_tolerance_enable`**

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

Example:

```
/* enable reception bit timing tolerance */
```

---

```
cec_reception_tolerance_enable();
```

### **cec\_reception\_tolerance\_disable**

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

**Table 3-56. Function cec\_reception\_tolerance\_disable**

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

Example:

```
/* disable reception bit timing tolerance */

cec_reception_tolerance_disable();
```

### **cec\_data\_send**

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

**Table 3-57. Function cec\_data\_send**

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

Example:

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

cec_data_send (0x55);
```

### **cec\_data\_receive**

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

**Table 3-58. Function cec\_data\_receive**

<b>Function name</b>	cec_data_receive
<b>Function prototype</b>	uint8_t cec_data_receive(void);
<b>Function descriptions</b>	receive a data by the CEC peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>Uint8_t</b>	the data to receive

Example:

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

uint8_t data = 0;

data = cec_data_receive();
```

### **cec\_interrupt\_enable**

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

**Table 3-59. Function cec\_interrupt\_enable**

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

<i>CEC_INT_TEND</i>	enable transmission successfully end interrupt
<i>CEC_INT_TU</i>	enable Tx data buffer underrun interrupt
<i>CEC_INT_TERR</i>	enable Tx-error interrupt
<i>CEC_INT_TAERR</i>	enable Tx ACK error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### cec\_interrupt\_disable

The description of `cec_interrupt_disable` is shown as below:

**Table 3-60. Function `cec_interrupt_disable`**

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

Example:

```
/* disable CEC_INT_BR interrupt */
```

```
cec_interrupt_disable (CEC_INT_BR);
```

### **cec\_flag\_get**

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

**Table 3-61. Function cec\_flag\_get**

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

Example:

```
/* get CEC_FLAG_BR status */

FlagStatus flag = reset;

flag = cec_flag_get (CEC_INT_BR);
```

### **cec\_flag\_clear**

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

**Table 3-62. Function cec\_flag\_clear**

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

Example:

```
/* clear CEC_FLAG_BR status */
cec_flag_get (CEC_INT_BR);
```

### **cec\_interrupt\_flag\_get**

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

**Table 3-63. Function cec\_interrupt\_flag\_get**

<b>Function name</b>	cec_interrupt_flag_get
<b>Function prototype</b>	FlagStatus cec_interrupt_flag_get(uint32_t flag);
<b>Function descriptions</b>	get CEC int flag and status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag
<i>CEC_INT_FLAG_BR</i>	Rx-byte data received

<i>CEC_INT_FLAG_REN</i> <i>D</i>	end of reception
<i>CEC_INT_FLAG_RO</i>	RX overrun
<i>CEC_INT_FLAG_BRE</i>	bit rising error
<i>CEC_INT_FLAG_BPSE</i>	short bit period error
<i>CEC_INT_FLAG_BPLE</i>	long bit period error
<i>CEC_INT_FLAG_RAE</i>	Rx ACK error
<i>CEC_INT_FLAG_ARBF</i>	arbitration lost
<i>CEC_INT_FLAG_TBR</i>	Tx-byte data request
<i>CEC_INT_FLAG_TEND</i>	transmission successfully end
<i>CEC_INT_FLAG_TU</i>	Tx data buffer underrun
<i>CEC_INT_FLAG_TERR</i>	Tx-error
<i>CEC_INT_FLAG_TAER</i> <i>R</i>	Tx ACK error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get CEC_INT_FLAG_BR status */

FlagStatus flag = reset;

flag = cec_interrupt_flag_get (CEC_INT_FLAG_BR);
```

### **cec\_interrupt\_flag\_clear**

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

**Table 3-64. Function `cec_interrupt_flag_clear`**

<b>Function name</b>	<code>cec_interrupt_flag_clear</code>
<b>Function prototype</b>	<code>void cec_interrupt_flag_clear(uint32_t flag);</code>
<b>Function descriptions</b>	clear CEC int flag and status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag
<i>CEC_INT_FLAG_BR</i>	Rx-byte data received
<i>CEC_INT_FLAG_REN</i> <i>D</i>	end of reception
<i>CEC_INT_FLAG_RO</i>	RX overrun
<i>CEC_INT_FLAG_BRE</i>	bit rising error
<i>CEC_INT_FLAG_BPSE</i>	short bit period error
<i>CEC_INT_FLAG_BPLE</i>	long bit period error

<i>CEC_INT_FLAG_RAE</i>	Rx ACK error
<i>CEC_INT_FLAG_ARBF</i>	arbitration lost
<i>CEC_INT_FLAG_TBR</i>	Tx-byte data request
<i>CEC_INT_FLAG_TEND</i>	transmission successfully end
<i>CEC_INT_FLAG_TU</i>	Tx data buffer underrun
<i>CEC_INT_FLAG_TERR</i>	Tx-error
<i>CEC_INT_FLAG_TAER</i> <i>R</i>	Tx ACK error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear CEC_INT_FLAG_BR status */
cec_interrupt_flag_clear(CEC_INT_FLAG_BR);
```

## 3.4. CMP

The general purpose comparator can work either standalone(all terminal are available on I/Os) or together with the timers. It could 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, achieves some current control by working together with a PWM output of a TIMER.The CMP registers are listed in chapter [3.4.1](#), the CMP firmware functions are introduced in chapter [3.4.2](#).

### 3.4.1. Descriptions of Peripheral registers

CMP registers are listed in the table shown as below:

**Table 3-65. CMP Registers**

Registers	Descriptions
CMP_CS	Control/Status register

### 3.4.2. Descriptions of Peripheral functions

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

**Table 3-66. CMP firmware function**

Function name	Function description
cmp_deinit	deinitialize comparator
cmp_mode_init	initialize comparator mode
cmp_output_init	initialize comparator output

Function name	Function description
cmp_enable	enable comparator
cmp_disable	disable comparator
cmp_switch_enable	enable comparator switch
cmp_switch_disable	disable comparator switch
cmp_window_enable	enable the window mode
cmp_window_disable	disable the window mode
cmp_lock_enable	lock the comparator
cmp_output_level_get	get output level

### Enum operating\_mode\_enum

**Table 3-67. Enum operating\_mode\_enum**

Member name	Function description
CMP_HIGHSPEED	high speed mode
CMP_MIDDLE SPEED	medium speed mode
CMP_LOWSPEED	low speed mode
CMP_VERYLOWSPEED	very-low speed mode

### Enum inverting\_input\_enum

**Table 3-68. Enum inverting\_input\_enum**

Member name	Function description
CMP_1_4VREFINT	VREFINT /4 input
CMP_1_2VREFINT	VREFINT /2 input
CMP_3_4VREFINT	VREFINT *3/4 input
CMP_VREFINT	VREFINT input
CMP_DAC	PA4 (DAC) input
CMP_PA5	PA5 input
CMP_PA_0_2	PA0 or PA2 input

### Enum cmp\_hysteresis\_enum

**Table 3-69. Enum cmp\_hysteresis\_enum**

Member name	Function description
CMP_HYSTERESIS_NO	output no hysteresis
CMP_HYSTERESIS_LOW	output low hysteresis
CMP_HYSTERESIS_MIDDLE	output middle hysteresis
CMP_HYSTERESIS_HIGH	output high hysteresis

### Enum cmp\_output\_enum

**Table 3-70. Enum cmp\_output\_enum**

Member name	Function description
CMP_OUTPUT_NONE	output no selection
CMP_OUTPUT_TIMER0BKin	TIMER 0 break input
CMP_OUTPUT_TIMER0IC0	TIMER 0 channel0 input capture
CMP_OUTPUT_TIMER0OCPREC LR	TIMER 0 OCPRE_CLR input
CMP_OUTPUT_TIMER1IC3	TIMER 1 channel3 input capture
CMP_OUTPUT_TIMER1OCPREC LR	TIMER 1 OCPRE_CLR input
CMP_OUTPUT_TIMER2IC0	TIMER 2 channel0 input capture
CMP_OUTPUT_TIMER2OCPREC LR	TIMER 2 OCPRE_CLR input

### cmp\_deinit

The description of cmp\_deinit is shown as below:

**Table 3-71. Function cmp\_deinit**

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

Example:

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

### cmp\_mode\_init

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

**Table 3-72. Function cmp\_mode\_init**

Function name	cmp_mode_init
Function prototype	void cmp_mode_init(uint32_t cmp_periph, operating_mode_enum

	operating_mode, inverting_input_enum inverting_input, cmp_hysteresis_enum output_hysteresis)
<b>Function descriptions</b>	initialize comparator mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	comparator periph
<b>CMP0</b>	comparator 0
<b>CMP1</b>	comparator 1
<b>Input parameter{in}</b>	
<b>operating_mode</b>	operating_mode, refer to <a href="#">Table 3-67. Enum operating_mode enum</a>
<b>CMP_HIGHSPEED</b>	high speed mode
<b>CMP_MIDDLESPEED</b>	medium speed mode
<b>CMP_LOWSPEED</b>	low speed mode
<b>CMP_VERYLOWSPEED</b> <i>D</i>	very-low speed mode
<b>Input parameter{in}</b>	
<b>inverting_input</b>	inverting_input, refer to <a href="#">Table 3-68. Enum inverting_input enum</a>
<b>CMP_1_4VREFINT</b>	VREFINT *1/4 input
<b>CMP_1_2VREFINT</b>	VREFINT *1/2 input
<b>CMP_3_4VREFINT</b>	VREFINT *3/4 input
<b>CMP_VREFINT</b>	VREFINT input
<b>CMP_DAC</b>	PA4 (DAC) input
<b>CMP_PA5</b>	PA5 input
<b>CMP_PA_0_2</b>	PA0 or PA2 input
<b>Input parameter{in}</b>	
<b>output_hysteresis</b>	hysteresis, refer to <a href="#">Table 3-69. Enum cmp_hysteresis enum</a>
<b>CMP_HYSTERESIS_NO</b>	output no hysteresis
<b>CMP_HYSTERESIS_LOW</b>	output low hysteresis
<b>CMP_HYSTERESIS_MIDDLE</b>	output middle hysteresis
<b>CMP_HYSTERESIS_HIGH</b>	output high hysteresis
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

cmp\_mode\_init(CMP0,CMP\_HIGHSPD,CMP\_1\_4VREFINT, CMP\_HYSTESIS\_NO);

## cmp\_output\_init

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

**Table 3-73. Function cmp\_output\_init**

<b>Function name</b>	cmp_output_init
<b>Function prototype</b>	void cmp_output_init(uint32_t cmp_periph, cmp_output_enum output_selection, uint32_t output_polarity);
<b>Function descriptions</b>	initialize comparator output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	comparator periph
CMP0	comparator 0
CMP1	comparator 1
<b>Input parameter{in}</b>	
<b>output_selection</b>	output_selection, refer to <a href="#">Table 3-70. Enum cmp_output_enum</a>
CMP_OUTPUT_NONE	output no selection
CMP_OUTPUT_TIMER_OBKIN	TIMER 0 break input
CMP_OUTPUT_TIMER_OIC0	TIMER 0 channel0 input capture
CMP_OUTPUT_TIMER_OOCPRECLR	TIMER 0 OCPRE_CLR input
CMP_OUTPUT_TIMER_1IC3	TIMER 1 channel3 input capture
CMP_OUTPUT_TIMER_1OCPRECLR	TIMER 1 OCPRE_CLR input
CMP_OUTPUT_TIMER_2IC0	TIMER 2 channel0 input capture
CMP_OUTPUT_TIMER_2OCPRECLR	TIMER 2 OCPRE_CLR input
<b>Input parameter{in}</b>	
<b>output_polarity</b>	output_polarity
CMP_OUTPUT_POLARITY_INVERTED	output is inverted
CMP_OUTPUT_POLARITY_NOINVERTED	output is not inverted
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* CMP0 output initialize*/
cmp_output_init(CMP0,CMP_OUTPUT_TIMER0BKIN,
CMP_OUTPUT_POLARITY_NOINVERTED);
```

### **cmp\_enable**

The description of cmp\_enable is shown as below:

**Table 3-74. Function cmp\_enable**

<b>Function name</b>	cmp_enable
<b>Function prototype</b>	void cmp_enable(uint32_t cmp_periph);
<b>Function descriptions</b>	enable comparator
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
cmp_periph	comparator periph
CMP0	comparator 0
CMP1	comparator 1
<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 3-75. Function cmp\_disable**

<b>Function name</b>	cmp_disable
<b>Function prototype</b>	void cmp_disable(uint32_t cmp_periph);
<b>Function descriptions</b>	disable comparator
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
cmp_periph	comparator periph
CMP0	comparator 0
CMP1	comparator 1
<b>Output parameter{out}</b>	

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

Example:

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

### **cmp\_switch\_enable**

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

**Table 3-76. Function cmp\_switch\_enable**

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

Example:

```
/* enable CMP switch */
cmp_switch_enable();
```

### **cmp\_switch\_disable**

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

**Table 3-77. Function cmp\_switch\_disable**

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

-	-
---	---

Example:

```
/* disable CMP switch */

cmp_switch_disable();
```

### **cmp\_window\_enable**

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

**Table 3-78. Function cmp\_window\_enable**

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

Example:

```
/* enable the window mode */

cmp_window_enable ();
```

### **cmp\_window\_disable**

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

**Table 3-79. Function cmp\_window\_disable**

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

Example:

```
/* disable the window mode */

cmp_window_disable();
```

### **cmp\_lock\_enable**

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

**Table 3-80. Function cmp\_lock\_enable**

<b>Function name</b>	cmp_lock_enable
<b>Function prototype</b>	void cmp_lock_enable(uint32_t cmp_periph);
<b>Function descriptions</b>	lock the comparator
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
cmp_periph	comparator periph
CMP0	comparator 0
CMP1	comparator 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* lock CMP0 register */

cmp_lock_enable(CMP0);
```

### **cmp\_output\_level\_get**

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

**Table 3-81. Function cmp\_output\_level\_get**

<b>Function name</b>	cmp_output_level_get
<b>Function prototype</b>	uint32_t cmp_output_level_get(uint32_t cmp_periph);
<b>Function descriptions</b>	get output level
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
cmp_periph	comparator periph
CMP0	comparator 0
CMP1	comparator 1
<b>Output parameter{out}</b>	
-	-

Return value	
uint32_t	the output level
CMP_OUTPUTLEVEL_HIGH	Non-inverting input above inverting input and the output is high
CMP_OUTPUTLEVEL_LOW	Non-inverting input below inverting input and the output is low

Example:

```
/* get CMP0 output level */
cmp_output_level_get (CMP0);
```

## 3.5. CRC

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

### 3.5.1. Descriptions of Peripheral registers

CRC registers are listed in the table shown as below:

**Table 3-82. 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

### 3.5.2. Descriptions of Peripheral functions

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

**Table 3-83. CRC firmware function**

Function name	Function description
crc_deinit	deinit CRC calculation unit
crc_reverse_output_data_enable	enable the reverse operation of output data
crc_reverse_output_data_disable	disable the reverse operation of output data
crc_data_register_reset	reset data register to the value of initializaiton data register
crc_data_register_read	read the data register
crc_free_data_register_read	read the free data register
crc_free_data_register_write	write the free data register
crc_init_data_register_write	write the initial value register

Function name	Function description
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 a 32-bit data
crc_block_data_calculate	CRC calculate a 32-bit data array

## crc\_deinit

The description of crc\_deinit is shown as below:

**Table 3-84. 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 3-85. Function crc\_reverse\_output\_data\_enable**

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

Example:

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

crc_reverse_output_data_enable();
```

### **crc\_reverse\_output\_data\_disable**

The description of `crc_reverse_output_data_disable` is shown as below:

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

<b>Function name</b>	<code>crc_reverse_output_data_disable</code>
<b>Function prototype</b>	<code>void crc_reverse_output_data_disable (void);</code>
<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 3-87. Function `crc_data_register_reset`**

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

Example:

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

```
crc_data_register_reset();
```

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

The description of `crc_data_register_read` is shown as below:

**Table 3-88. Function `crc_data_register_read`**

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

Example:

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

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

The description of `crc_free_data_register_read` is shown as below:

**Table 3-89. Function `crc_free_data_register_read`**

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

Example:

---

```

/* read crc free data register */

uint8_t crc_value = 0;

crc_value = crc_free_data_register_read();

```

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

The description of `crc_free_data_register_write` is shown as below:

**Table 3-90. Function `crc_free_data_register_write`**

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

Example:

```

/* write the free data register */

crc_free_data_register_write(0x11);

```

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

The description of `crc_init_data_register_write` is shown as below:

**Table 3-91. Function `crc_init_data_register_write`**

<b>Function name</b>	crc_init_data_register_write	
<b>Function prototype</b>	void crc_init_data_register_write(uint32_t init_data)	
<b>Function descriptions</b>	write the initialization 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 initialzaiton 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 3-92. 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>	
<code>data_reverse</code>	specify input data reverse function
<code>CRC_INPUT_DATA_N OT</code>	input data is not reversed
<code>CRC_INPUT_DATA_B YTE</code>	input data is reversed on 8 bits
<code>CRC_INPUT_DATA_H ALFWORD</code>	input data is reversed on 16 bits
<code>CRC_INPUT_DATA_W 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 3-93. Function `crc_polynomial_size_set`**

<b>Function name</b>	<code>crc_polynomial_size_set</code>
<b>Function prototype</b>	<code>void crc_polynomial_size_set(uint32_t poly_size)</code>
<b>Function descriptions</b>	configure the CRC size of polynomial function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>poly_size</code>	size of polynomial

<b>CRC_CTL_PS_32</b>	32-bit polynomial for CRC calculation
<b>CRC_CTL_PS_16</b>	16-bit polynomial for CRC calculation
<b>CRC_CTL_PS_8</b>	8-bit polynomial for CRC calculation
<b>CRC_CTL_PS_7</b>	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 3-94. 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 3-95. Function crc\_single\_data\_calculate**

<b>Function name</b>	crc_single_data_calculate
<b>Function prototype</b>	uint32_t crc_single_data_calculate(uint32_t sdata);
<b>Function descriptions</b>	CRC calculate a 32-bit data
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
sdata	specify 32-bit data
Output parameter{out}	
-	-
Return value	
uint32_t	32-bit CRC calculate value (0-0xFFFFFFFF)

Example:

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

uint32_t val = 0, valcrc = 0;

val = (uint32_t)0xabcd1234;

rcu_periph_clock_enable(RCU_CRC);

valcrc = crc_single_data_calculate(val);
```

### crc\_block\_data\_calculate

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

**Table 3-96. Function crc\_block\_data\_calculate**

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

Example:

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

#define BUFFER_SIZE      6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {

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

rcu_periph_clock_enable(RCU_CRC);
```

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

## 3.6. CTC

The Clock Trim Controller (CTC) is used to trim internal 48MHz RC oscillator (IRC48M) automatically by hardware. The CTC registers are listed in chapter [3.6.1](#), the CTC firmware functions are introduced in chapter [3.6.2](#).

### 3.6.1. Descriptions of Peripheral registers

CTC registers are listed in the table shown as below:

**Table 3-97. CTC Registers**

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

### 3.6.2. Descriptions of Peripheral functions

CTC registers are listed in the table shown as below:

**Table 3-98. CTC firmware function**

Function name	Function description
ctc_deinit	reset ctc clock trim controller
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_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_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

Function name	Function description
ctc_interrupt_enable	enable the CTC interrupt
ctc_interrupt_disable	disable the CTC interrupt
ctc_flag_get	get CTC flag
ctc_flag_clear	clear CTC flag
ctc_interrupt_flag_get	get CTC interrupt flag
ctc_interrupt_flag_clear	clear CTC interrupt flag

## ctc\_deinit

The description of ctc\_deinit is shown as below:

**Table 3-99. Function ctc\_deinit**

Function name	ctc_deinit
Function prototype	void ctc_deinit (void)
Function descriptions	reset ctc clock trim controller
Precondition	-
The called functions	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\_refsource\_polarity\_config

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

**Table 3-100. Function ctc\_refsource\_polarity\_config**

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

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

Example:

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

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

The description of `ctc_refresource_signal_select` is shown as below:

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

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

Example:

```
/* reference signal selection */
ctc_refresource_signal_select(CTC_REFRESOURCE_LXTAL);
```

### **ctc\_refresource\_prescaler\_config**

The description of `ctc_refresource_prescaler_config` is shown as below:

**Table 3-102. Function `ctc_refresource_prescaler_config`**

<b>Function name</b>	<code>ctc_refresource_prescaler_config</code>
<b>Function prototype</b>	<code>void ctc_refresource_prescaler_config(uint32_t prescaler);</code>

<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
<code>CTC_REFRESOURCE_P SC_OFF</code>	reference signal not divided
<code>CTC_REFRESOURCE_P SC_DIV2</code>	reference signal divided by 2
<code>CTC_REFRESOURCE_P SC_DIV4</code>	reference signal divided by 4
<code>CTC_REFRESOURCE_P SC_DIV8</code>	reference signal divided by 8
<code>CTC_REFRESOURCE_P SC_DIV16</code>	reference signal divided by 16
<code>CTC_REFRESOURCE_P SC_DIV32</code>	reference signal divided by 32
<code>CTC_REFRESOURCE_P SC_DIV64</code>	reference signal divided by 64
<code>CTC_REFRESOURCE_P SC_DIV128</code>	reference signal divided by 128
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure reference signal source prescaler */
ctc_refresource_prescaler_config(CTC_REFRESOURCE_PSC_DIV2);
```

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

The description of `ctc_clock_limit_value_config` is shown as below:

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

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

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

Example:

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

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

The description of `ctc_counter_reload_value_config` is shown as below:

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

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

Example:

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

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

The description of `ctc_counter_enable` is shown as below:

**Table 3-105. 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 3-106. 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\_ir48m\_trim\_value\_config**

The description of `ctc_ir48m_trim_value_config` is shown as below:

**Table 3-107. Function `ctc_ir48m_trim_value_config`**

<b>Function name</b>	ctc_ir48m_trim_value_config
<b>Function prototype</b>	void ctc_ir48m_trim_value_config(uint8_t trim_value);
<b>Function descriptions</b>	configure the IRC48M trim value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
trim_value	0~63
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* IRC48M trim value configuration */

ctc_irc48m_trim_value_config (0x01);
```

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

The description of `ctc_software_refsource_pulse_generate` is shown as below:

**Table 3-108. 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 3-109. Function `ctc_hardware_trim_mode_config`**

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

Return value	
-	-

Example:

```
/* enable CTC hardware trim */
ctc.hardware_trim_mode_config(CTC_HARDWARE_TRIM_MODE_ENABLE);
```

### ctc\_counter\_capture\_value\_read

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

**Table 3-110. Function ctc\_counter\_capture\_value\_read**

<b>Function name</b>	ctc_counter_capture_value_read
<b>Function prototype</b>	uint16_t ctc_counter_capture_value_read(void);
<b>Function descriptions</b>	read CTC counter capture value
<b>Precondition</b>	-
<b>The called functions</b>	-
<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

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 3-111. Function ctc\_counter\_direction\_read**

<b>Function name</b>	ctc_counter_direction_read
<b>Function prototype</b>	FlagStatus ctc_counter_direction_read(void);
<b>Function descriptions</b>	read CTC trim counter direction
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

Return value	
FlagStatus	SET(down-counting) / RESET(up-counting)

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 3-112. Function `ctc_counter_reload_value_read`**

Function name	ctc_counter_reload_value_read
Function prototype	uint16_t ctc_counter_reload_value_read(void);
Function descriptions	read CTC counter reload value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	the 16-bit CTC 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\_ir48m\_trim\_value\_read**

The description of `ctc_ir48m_trim_value_read` is shown as below:

**Table 3-113. Function `ctc_ir48m_trim_value_read`**

Function name	ctc_ir48m_trim_value_read
Function prototype	uint8_t ctc_ir48m_trim_value_read(void);
Function descriptions	read the IRC48M trim value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
<b>Return value</b>	
uint8_t	the 6-bit IRC48M trim value (0-63)

Example:

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

uint8_t ctc_trim_value = 0;

ctc_trim_value = ctc_irc48m_trim_value_read();
```

### ctc\_interrupt\_enable

The description of `ctc_interrupt_enable` is shown as below:

**Table 3-114. 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
<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_EREF</code>	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 3-115. Function `ctc_interrupt_disable`**

<b>Function name</b>	ctc_interrupt_disable
<b>Function prototype</b>	void ctc_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable the CTC interrupt
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	CTC interrupt
<b>CTC_INT_CKOK</b>	clock trim OK interrupt
<b>CTC_INT_CKWARN</b>	clock trim warning interrupt
<b>CTC_INT_ERR</b>	error interrupt
<b>CTC_INT_EREF</b>	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\_flag\_get**

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

**Table 3-116. Function ctc\_flag\_get**

<b>Function name</b>	ctc_flag_get
<b>Function prototype</b>	FlagStatus ctc_flag_get(uint32_t flag);
<b>Function descriptions</b>	get CTC status flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	CTC status flag
<b>CTC_FLAG_CKOK</b>	clock trim OK interrupt flag
<b>CTC_FLAG_CKWARN</b>	clock trim warning interrupt flag
<b>CTC_FLAG_ERR</b>	error interrupt flag
<b>CTC_FLAG_EREF</b>	expect reference interrupt flag
<b>CTC_FLAG_CKERR</b>	clock trim error bit
<b>CTC_FLAG_REFMISS</b>	reference sync pulse miss flag
<b>CTC_FLAG_TRIMERR</b>	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 3-117. Function ctc\_flag\_clear**

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

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

**Table 3-118. Function ctc\_interrupt\_flag\_get**

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

<i>CTC_INT_FLAG_ERR</i>	error interrupt
<i>CTC_INT_FLAG_EREF</i>	expect reference interrupt
<i>CTC_INT_FLAG_CKE_RR</i>	clock trim error bit interrupt
<i>CTC_INT_FLAG_REF_MISS</i>	reference sync pulse miss interrupt
<i>CTC_INT_FLAG_TRIM_ERR</i>	trim value error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<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 3-119. Function `ctc_interrupt_flag_clear`**

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

Return value	
-	-

Example:

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

## 3.7. DAC

The Digital-to-analog converter converts 12-bit digital data to a voltage on the external pins.

The DAC registers are listed in chapter [3.7.1](#), the DAC firmware functions are introduced in chapter [3.7.2](#). GD32F330xxxx do not have DAC peripheral.

### 3.7.1. Descriptions of Peripheral registers

DAC registers are listed in the table shown as below:

**Table 3-120. DAC Registers**

Registers	Descriptions
DAC_CTL	DAC control register
DAC_SWT	DAC software trigger register
DAC_R12DH	DAC 12-bit right-aligned data holding register
DAC_L12DH	DAC 12-bit left-aligned data holding register
DAC_R8DH	DAC 8-bit right-aligned data holding register
DAC_DO	DAC data output register
DAC_STAT	DAC Status register

### 3.7.2. Descriptions of Peripheral functions

DAC registers are listed in the table shown as below:

**Table 3-121. DAC firmware function**

Function name	Function description
dac_deinit	deinitialize DAC
dac_enable	enable DAC
dac_disable	disable DAC
dac_dma_enable	dac_dma_enable
dac_dma_disable	dac_dma_disable
dac_output_buffer_enable	enable DAC output buffer
dac_output_buffer_disable	disable DAC output buffer
dac_trigger_enable	enable DAC trigger
dac_trigger_disable	disable DAC trigger
dac_software_trigger_enable	enable DAC software trigger

Function name	Function description
dac_software_trigger_disable	disable DAC software trigger
dac_interrupt_enable	enable DAC interrupt(DAC DMA underrun interrupt)
dac_interrupt_disable	disable DAC interrupt(DAC DMA underrun interrupt)
dac_trigger_source_config	configure DAC trigger source
dac_wave_mode_config	configure DAC wave mode
dac_wave_bit_width_config	configure DAC wave bit width
dac_lfsr_noise_config	configure DAC LFSR noise mode
dac_triangle_noise_config	configure DAC triangle noise mode
dac_output_value_get	get the last data output value
dac_flag_get	get the specified DAC flag(DAC DMA underrun flag)
dac_flag_clear	clear the specified DAC flag(DAC DMA underrun flag)
dac_interrupt_flag_get	get the specified DAC interrupt flag(DAC DMA underrun interrupt flag)
dac_interrupt_flag_clear	clear the specified DAC interrupt flag(DAC DMA underrun interrupt flag)
dac_data_set	set DAC data holding register value

## dac\_deinit

The description of dac\_deinit is shown as below:

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

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

Example:

```
/* deinitialize DAC */

dac_deinit();
```

## dac\_enable

The description of dac\_enable is shown as below:

**Table 3-123. Function dac\_enable**

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

Example:

```
/* enable DAC */

dac_enable();
```

### **dac\_disable**

The description of dac\_disable is shown as below:

**Table 3-124. Function dac\_disable**

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

Example:

```
/* disable DAC */

dac_disable();
```

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

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

**Table 3-125. Function dac\_dma\_enable**

<b>Function name</b>	dac_dma_enable
----------------------	----------------

<b>Function prototype</b>	void dac_dma_enable(void)
<b>Function descriptions</b>	enable DAC DMA function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DAC DMA function */

dac_dma_enable();
```

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

The description of `dac_dma_disable` is shown as below:

**Table 3-126. Function `dac_dma_disable`**

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

Example:

```
/* disable DAC DMA function */

dac_dma_disable();
```

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

The description of `dac_output_buffer_enable` is shown as below:

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

<b>Function name</b>	dac_output_buffer_enable
<b>Function prototype</b>	void dac_output_buffer_enable(void)
<b>Function descriptions</b>	enable DAC output buffer

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

Example:

```
/* enable DAC output buffer */
dac_output_buffer_enable();
```

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

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

**Table 3-128. Function dac\_output\_buffer\_disable**

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

Example:

```
/* disable DAC output buffer */
dac_output_buffer_disable();
```

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

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

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

<b>Function name</b>	dac_trigger_enable
<b>Function prototype</b>	void dac_trigger_enable(void)
<b>Function descriptions</b>	enable DAC trigger
<b>Precondition</b>	-
<b>The called functions</b>	-

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

Example:

```
/* enable DAC trigger */
```

```
dac_trigger_enable();
```

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

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

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

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

Example:

```
/* disable DAC trigger */
```

```
dac_trigger_disable();
```

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

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

**Table 3-131. Function dac\_software\_trigger\_enable**

<b>Function name</b>	dac_software_trigger_enable
<b>Function prototype</b>	void dac_software_trigger_enable(void)
<b>Function descriptions</b>	enable DAC software trigger
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

The description of `dac_software_trigger_disable` is shown as below:

**Table 3-132. Function `dac_software_trigger_disable`**

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

Example:

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

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

The description of `dac_interrupt_enable` is shown as below:

**Table 3-133. Function `dac_interrupt_enable`**

<b>Function name</b>	dac_interrupt_enable
<b>Function prototype</b>	void dac_interrupt_enable (void)
<b>Function descriptions</b>	enable DAC interrupt(DAC DMA underrun interrupt)
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* enable DAC interrupt(DAC DMA underrun interrupt) */

dac_interrupt_enable();
```

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

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

**Table 3-134. Function dac\_interrupt\_disable**

<b>Function name</b>	dac_interrupt_disable
<b>Function prototype</b>	void dac_interrupt_disable (void)
<b>Function descriptions</b>	disable DAC interrupt(DAC DMA underrun 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 DAC interrupt(DAC DMA underrun interrupt) */

dac_interrupt_disable();
```

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

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

**Table 3-135. Function dac\_trigger\_source\_config**

<b>Function name</b>	dac_trigger_source_config
<b>Function prototype</b>	void dac_trigger_source_config(uint32_t triggersource)
<b>Function descriptions</b>	set DAC trigger source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Input parameter{in}</b>	
<b>triggersource</b>	external triggers of DAC
<b>DAC_TRIGGER_T1_T RGO</b>	TIMER1 TRGO

<i>DAC_TRIGGER_T2_T RGO</i>	TIMER2 TRGO
<i>DAC_TRIGGER_T5_T RGO</i>	TIMER5 TRGO
<i>DAC_TRIGGER_T14_T RGO</i>	TIMER14 TRGO
<i>DAC_TRIGGER_EXTI_9</i>	EXTI interrupt line9 event
<i>DAC_TRIGGER_SOFT_WARE</i>	software trigger
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set DAC trigger source*/
dac_trigger_source_config(DAC_TRIGGER_T1_TRGO);
```

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

The description of **dac\_wave\_mode\_config** is shown as below:

**Table 3-136. Function dac\_wave\_mode\_config**

<b>Function name</b>	dac_wave_mode_config
<b>Function prototype</b>	void dac_wave_mode_config(uint32_t wave_mode)
<b>Function descriptions</b>	configure DAC wave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Input parameter{in}</b>	
<i>wave_mode</i>	wave_mode
<i>DAC_WAVE_DISABLE</i>	wave disable
<i>DAC_WAVE_MODE_L_FSR</i>	LFSR noise mode
<i>DAC_WAVE_MODE_T_RIANGLE</i>	triangle noise mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DAC wave mode */

dac_wave_mode_config(DAC_WAVE_DISABLE);
```

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

The description of `dac_wave_bit_width_config` is shown as below:

**Table 3-137. Function `dac_wave_bit_width_config`**

<b>Function name</b>	<code>dac_wave_bit_width_config</code>
<b>Function prototype</b>	<code>void dac_wave_bit_width_config(uint32_t bit_width)</code>
<b>Function descriptions</b>	configure DAC wave bit width
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Input parameter{in}</b>	
<b>bit_width</b>	noise wave bit width
<b>DAC_WAVE_BIT_WIDTH_1</b>	$x = 1..12$
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DAC wave bit width */

dac_wave_bit_width_config(DAC_WAVE_BIT_WIDTH_1);
```

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

The description of `dac_lfsr_noise_config` is shown as below:

**Table 3-138. 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 unmask_bits)</code>
<b>Function descriptions</b>	configure DAC LFSR noise mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Input parameter{in}</b>	
<b>unmask_bits</b>	noise wave unmask bit width
<b>DAC_LFSR_BIT0</b>	unmask the LFSR bit0
<b>DAC_LFSR_BITSx_0</b>	unmask the LFSR bits[x:0],x=1..11

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### dac\_triangle\_noise\_config

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

**Table 3-139. Function dac\_triangle\_noise\_config**

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

Example:

```
/* configure DAC triangle noise mode */
dac_triangle_noise_config(DAC_TRIANGLE_AMPLITUDE_1);
```

### dac\_output\_value\_get

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

**Table 3-140. Function dac\_output\_value\_get**

<b>Function name</b>	dac_output_value_get
<b>Function prototype</b>	uint16_t dac_output_value_get(void)
<b>Function descriptions</b>	get DAC output value
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
<b>uint16_t</b>	DAC output data (0x0000 – 0x07FF)

Example:

```
/* get DAC output value */

uint16_t data = 0;

data = dac_output_value_get();
```

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

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

**Table 3-141. Function dac\_flag\_get**

<b>Function name</b>	dac_flag_get
<b>Function prototype</b>	FlagStatus dac_flag_get(void)
<b>Function descriptions</b>	get the specified DAC flag(DAC DMA underrun flag)
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	the state of DAC
<i>SET</i>	DMA underrun error condition occurred
<i>RESET</i>	No DMA underrun error condition occurred

Example:

```
/* get the specified DAC flag(DAC DMA underrun flag) */

FlagStatus dac_falg = RESET;

dac_falg = dac_flag_get();
```

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

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

**Table 3-142. Function dac\_flag\_clear**

<b>Function name</b>	dac_flag_clear
<b>Function prototype</b>	void dac_flag_clear(void)

<b>Function descriptions</b>	clear the specified DAC flag(DAC DMA underrun 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 the specified DAC flag(DAC DMA underrun flag)*/  
  
dac_flag_clear();
```

### dac\_interrupt\_flag\_get

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

**Table 3-143. Function dac\_interrupt\_flag\_get**

<b>Function name</b>	dac_interrupt_flag_get
<b>Function prototype</b>	FlagStatus dac_interrupt_flag_get(void)
<b>Function descriptions</b>	get the specified DAC interrupt flag(DAC DMA underrun interrupt flag)
<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>	the state of DAC
<b>SET</b>	DMA underrun error interrupt occurred
<b>RESET</b>	No DMA underrun error interrupt occurred

Example:

```
/* get the specified DAC flag(DAC DMA underrun flag) */
```

```
FlagStatus dac_falg = RESET;  
  
dac_falg = dac_interrupt_flag_get();
```

### dac\_interrupt\_flag\_clear

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

**Table 3-144. Function dac\_interrupt\_flag\_clear**

<b>Function name</b>	dac_interrupt_flag_clear
<b>Function prototype</b>	void dac_interrupt_flag_clear(void)
<b>Function descriptions</b>	clear the specified DAC interrupt flag(DAC DMA underrun interrupt 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 the specified DAC interrupt flag(DAC DMA underrun interrupt flag)*/  
dac_interrupt_flag_clear();
```

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

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

**Table 3-145. Function dac\_data\_set**

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

Example:

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

dac\_data\_set(DAC\_ALIGN\_8B\_R,0xff);

## 3.8. DBG

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

### 3.8.1. Descriptions of Peripheral registers

DBG registers are listed in the table shown as below:

**Table 3-146. 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 3-147. 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 3-148. Enum dbg\_periph\_enum**

Member name	Function description
DBG_FWDGT_HOLD	debug FWDGT kept when core is halted
DBG_WWDGT_HOLD	debug WWDGT kept when core is halted
DBG_TIMER0_HOLD	hold TIMER0 counter when core is halted
DBG_TIMER1_HOLD	hold TIMER1 counter when core is halted
DBG_TIMER2_HOLD	hold TIMER2 counter when core is halted
DBG_TIMER5_HOLD	hold TIMER5 counter when core is halted
DBG_TIMER13_HOLD	hold TIMER13 counter when core is halted
DBG_TIMER14_HOLD	hold TIMER14 counter when core is halted
DBG_TIMER15_HOLD	hold TIMER15 counter when core is halted
DBG_TIMER16_HOLD	hold TIMER16 counter when core is halted

Member name	Function description
DBG_I2C0_HOLD	hold I2C0 smbus when core is halted
DBG_I2C1_HOLD	hold I2C1 smbus when core is halted
DBG_RTC_HOLD	hold RTC counter when core is halted

### dbg\_deinit

The description of dbg\_deinit is shown as below:

**Table 3-149. Function dbg\_deinit**

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

Example:

```
/* deinitialize the DBG*/
dbg_deinit();
```

### dbg\_id\_get

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

**Table 3-150. Function dbg\_id\_get**

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

Example:

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

```
uint32_t id_value = 0;
```

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

### **dbg\_low\_power\_enable**

The description of `dbg_low_power_enable` is shown as below:

**Table 3-151. 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>	
<code>dbg_low_power</code>	low power mode
<code>DBG_LOW_POWER_SLEEP</code>	keep debugger connection during sleep mode
<code>DBG_LOW_POWER_DEEPSLEEP</code>	keep debugger connection during deepsleep mode
<code>DBG_LOW_POWER_STANDBY</code>	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 3-152. 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>	
<code>dbg_low_power</code>	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 3-153. 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 3-148. Enum <code>dbg_periph_enum</code></a>
<i>DBG_FWDGT_HOLD</i>	debug FWDGT kept when core is halted
<i>DBG_WWDGT_HOLD</i>	debug WWDGT kept when core is halted
<i>DBG_TIMERx_HOLD</i>	x=0,1,2,5,13,14,15,16, hold TIMERx counter when core is halted
<i>DBG_I2Cx_HOLD</i>	x=0,1, hold I2Cx smbus when core is halted
<i>DBG_RTC_HOLD</i>	hold RTC counter when core is halted
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable peripheral behavior when the mcu is in debug mode */
dbg_periph_enable(DBG_TIMER0_HOLD);
```

### dbg\_periph\_disable

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

**Table 3-154. 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 3-148. Enum dbg_periph_enum</a>
<i>DBG_FWDGT_HOLD</i>	debug FWDGT kept when core is halted
<i>DBG_WWDGT_HOLD</i>	debug WWDGT kept when core is halted
<i>DBG_TIMERx_HOLD</i>	x=0,1,2,5,13,14,15,16, hold TIMERx counter when core is halted
<i>DBG_I2Cx_HOLD</i>	x=0,1, hold I2Cx smbus when core is halted
<i>DBG_RTC_HOLD</i>	hold RTC counter when core is halted
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

dbg_periph_disable(DBG_TIMER0_HOLD);
```

## 3.9. DMA

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

### 3.9.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

**Table 3-155. DMA Registers**

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

Registers	Descriptions
DMA_CHxCNT (x=0..6)	Channel x counter register
DMA_CHxPADDR (x=0..6)	Channel x peripheral base address register
DMA_CHxMADDR (x=0..6)	Channel x memory base address register

### 3.9.2. Descriptions of Peripheral functions

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

**Table 3-156. DMA firmware function**

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

### **dma\_channel\_enum**

**Table 3-157. Enum dma\_channel\_enum**

Member name	Function description
DMA_CH0	DMA Channel0
DMA_CH1	DMA Channel1
DMA_CH2	DMA Channel2
DMA_CH3	DMA Channel3
DMA_CH4	DMA Channel4
DMA_CH5	DMA Channel5
DMA_CH6	DMA Channel6

### **Structure dma\_parameter\_struct**

**Table 3-158. Structure dma\_parameter\_struct**

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

### **dma\_deinit**

The description of dma\_deinit is shown as below:

**Table 3-159. Function dma\_deinit**

<b>Function name</b>	dma_deinit
<b>Function prototype</b>	void dma_deinit(dma_channel_enum channelx);
<b>Function descriptions</b>	deinitialize DMA a channel registers
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
DMA_CHx(x=0..6)	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* DMA channel0 initialize */

dma_deinit(DMA_CH0);
```

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

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

**Table 3-160. Function dma\_struct\_para\_init**

<b>Function name</b>	dma_struct_para_init
<b>Function prototype</b>	void dma_struct_para_init(dma_parameter_struct* init_struct);
<b>Function descriptions</b>	initialize the parameters of DMA struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
init_struct	Structure for initialization, the structure members can refer to <a href="#">Table 3-158. Structure dma_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the parameters of DMA */

dma_parameter_struct dma_init_struct;

dma_struct_para_init(&dma_init_struct);
```

### **dma\_init**

The description of dma\_init is shown as below:

**Table 3-161. Function dma\_init**

<b>Function name</b>	dma_init
<b>Function prototype</b>	void dma_init(dma_channel_enum channelx, dma_parameter_struct init_struct);
<b>Function descriptions</b>	initialize DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
DMA_CHx( x=0..6)	DMA channel selection
<b>Input parameter{in}</b>	

<b>init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-158.</a> <a href="#"><u>Structure dma_parameter_struct.</u></a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* DMA channel0 initialize */

dma_parameter_struct dma_init_struct;

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

## dma\_circulation\_enable

The description of `dma_circulation_enable` is shown as below:

**Table 3-162. 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>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum <code>dma_channel_enum</code>.</a>
DMA_CHx( x=0..6)	DMA channel selection
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

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

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

The description of `dma_circulation_disable` is shown as below:

**Table 3-163. Function `dma_circulation_disable`**

<b>Function name</b>	dma_circulation_disable
<b>Function prototype</b>	void dma_circulation_disable(dma_channel_enum channelx);
<b>Function descriptions</b>	disable DMA circulation mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
DMA_CHx( x=0..6)	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA channel0 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 3-164. Function `dma_memory_to_memory_enable`**

<b>Function name</b>	dma_memory_to_memory_enable
<b>Function prototype</b>	void dma_memory_to_memory_enable(dma_channel_enum channelx);
<b>Function descriptions</b>	enable memory to memory mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
DMA_CHx( x=0..6)	DMA channel selection

Output parameter{out}	
-	-
Return value	
-	-

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 3-165. 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>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum <code>dma_channel_enum</code>.</a>
<b>DMA_CHx( x=0..6)</b>	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

The description of `dma_channel_enable` is shown as below:

**Table 3-166. 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>	-
Input parameter{in}	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum <code>dma_channel_enum</code>.</a>

	<b><a href="#">dma_channel_enum</a>.</b>
DMA_CHx(x=0..6)	DMA channel selection
<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 3-167. 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>	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
DMA_CHx( x=0..6)	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA channel0 */

dma_channel_disable(DMA_CH0);
```

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

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

**Table 3-168. 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>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
<b>DMA_CHx(x=0..6)</b>	DMA channel selection
<b>Input parameter{in}</b>	
<b>address</b>	peripheral base address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

#define BANK0_WRITE_START_ADDR ((uint32_t)0x08004000)

dma_periph_address_config(DMA_CH0, BANK0_WRITE_START_ADDR);
```

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

The description of `dma_memory_address_config` is shown as below:

**Table 3-169. Function `dma_memory_address_config`**

<b>Function name</b>	dma_memory_address_config
<b>Function prototype</b>	void dma_memory_address_config(dma_channel_enum channelx, uint32_t address);
<b>Function descriptions</b>	set DMA memory base address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
<b>DMA_CHx(x=0..6)</b>	DMA channel selection
<b>Input parameter{in}</b>	
<b>address</b>	memory base address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

uint8_t g_destbuf[TRANSFER_NUM];
```

---

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

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

The description of `dma_transfer_number_config` is shown as below:

**Table 3-170. Function `dma_transfer_number_config`**

<b>Function name</b>	dma_transfer_number_config	
<b>Function prototype</b>	void dma_transfer_number_config( dma_channel_enum channelx, uint32_t number);	
<b>Function descriptions</b>	set the number of remaining data to be transferred by the DMA	
<b>Precondition</b>	-	
<b>The called functions</b>	-	
<b>Input parameter{in}</b>		
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .	
DMA_CHx( x=0..6)	DMA channel selection	
<b>Input parameter{in}</b>		
number	data transfer number(0x00000000-0x0000FFFF)	
<b>Output parameter{out}</b>		
-	-	
<b>Return value</b>		
-	-	

Example:

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

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

The description of `dma_transfer_number_get` is shown as below:

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

<b>Function name</b>	dma_transfer_number_get	
<b>Function prototype</b>	uint32_t dma_transfer_number_get(dma_channel_enum channelx);	
<b>Function descriptions</b>	get the number of remaining data to be transferred by the DMA	
<b>Precondition</b>	-	
<b>The called functions</b>	-	
<b>Input parameter{in}</b>		
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .	
DMA_CHx( x=0..6)	DMA channel selection	
<b>Output parameter{out}</b>		

-	-
<b>Return value</b>	
<b>uint32_t</b>	DMA data transmission remaining quantity (0x00000000-0x0000FFFF)

Example:

```
/* get DMA channel0 transfer number */

uint32_t number = 0;

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

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

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

**Table 3-172. Function `dma_priority_config`**

<b>Function name</b>	dma_priority_config
<b>Function prototype</b>	void dma_priority_config(dma_channel_enum channelx, uint32_t priority);
<b>Function descriptions</b>	configure priority level of DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
<b>DMA_CHx(x=0..6)</b>	DMA channel selection
<b>Input parameter{in}</b>	
<b>priority</b>	priority Level of this channel
<b>DMA_PRIORITY_LOW</b>	low priority
<b>DMA_PRIORITY_MEDIUM</b>	medium priority
<b>DMA_PRIORITY_HIGH</b>	high priority
<b>DMA_PRIORITY_ULTRA_HIGH</b>	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 3-173. 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>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
DMA_CHx( x=0..6)	DMA channel selection
<b>Input parameter{in}</b>	
mwidth	transfer data width of memory
DMA_MEMORY_WIDTH_8BIT	transfer data width of memory is 8-bit
DMA_MEMORY_WIDTH_16BIT	transfer data width of memory is 16-bit
DMA_MEMORY_WIDTH_32BIT	transfer data width of memory is 32-bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-174. Function dma\_periph\_width\_config**

<b>Function name</b>	dma_periph_width_config
<b>Function prototype</b>	void dma_periph_width_config(dma_channel_enum channelx, uint32_t pwidth);
<b>Function descriptions</b>	configure transfer data width of peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
DMA_CHx( x=0..6)	DMA channel selection

Input parameter{in}	
<b>pwidth</b>	transfer data width of peripheral
<b>DMA_PERIPHERAL_W_IDTH_8BIT</b>	transfer data width of peripheral is 8-bit
<b>DMA_PERIPHERAL_W_IDTH_16BIT</b>	transfer data width of peripheral is 16-bit
<b>DMA_PERIPHERAL_W_IDTH_32BIT</b>	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 3-175. Function `dma_memory_increase_enable`**

<b>Function name</b>	dma_memory_increase_enable
<b>Function prototype</b>	void dma_memory_increase_enable(dma_channel_enum channelx);
<b>Function descriptions</b>	enable next address increasement algorithm of memory
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
<b>DMA_CHx(x=0..6)</b>	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

The description of `dma_memory_increase_disable` is shown as below:

**Table 3-176. Function dma\_memory\_increase\_disable**

<b>Function name</b>	dma_memory_increase_disable
<b>Function prototype</b>	void dma_memory_increase_disable(dma_channel_enum channelx);
<b>Function descriptions</b>	disable next address increasement algorithm of memory
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
DMA_CHx(x=0..6)	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### dma\_periph\_increase\_enable

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

**Table 3-177. Function dma\_periph\_increase\_enable**

<b>Function name</b>	dma_periph_increase_enable
<b>Function prototype</b>	void dma_periph_increase_enable(dma_channel_enum channelx);
<b>Function descriptions</b>	enable next address increasement algorithm of peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
DMA_CHx(x=0..6)	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

The description of `dma_periph_increase_disable` is shown as below:

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

<b>Function name</b>	dma_periph_increase_disable
<b>Function prototype</b>	void dma_periph_increase_disable(dma_channel_enum channelx);
<b>Function descriptions</b>	disable next address increasement algorithm of peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
<b>DMA_CHx(x=0..6)</b>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

The description of `dma_transfer_direction_config` is shown as below:

**Table 3-179. 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>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
<b>DMA_CHx( x=0..6)</b>	DMA channel selection
<b>Input parameter{in}</b>	
<b>direction</b>	specify the direction of data transfer
<b>DMA_PERIPHERAL_TO_MEMORY</b>	read from peripheral and write to memory
<b>DMA_MEMORY_TO_PERIPHERAL</b>	read from memory and write to peripheral

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

The description of `dma_flag_get` is shown as below:

**Table 3-180. 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>	-
Input parameter{in}	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum <code>dma_channel_enum</code>.</a>
<b>DMA_CHx( x=0..6)</b>	DMA channel selection
Input parameter{in}	
<b>flag</b>	specify get which flag
<b>DMA_FLAG_G</b>	global interrupt flag of channel
<b>DMA_FLAG_FTF</b>	full transfer finish flag of channel
<b>DMA_FLAG_HTF</b>	half transfer finish flag of channel
<b>DMA_FLAG_ERR</b>	error flag of channel
Output parameter{out}	
-	-
Return value	
<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 3-181. 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>	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
DMA_CHx(x=0..6)	DMA channel selection
<b>Input parameter{in}</b>	
flag	specify get which flag
DMA_FLAG_G	global interrupt flag of channel
DMA_FLAG_FTF	full transfer finish flag of channel
DMA_FLAG_HTF	half transfer finish flag of channel
DMA_FLAG_ERR	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\_flag\_get

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

**Table 3-182. Function dma\_interrupt\_flag\_get**

<b>Function name</b>	dma_interrupt_flag_get
<b>Function prototype</b>	FlagStatus dma_interrupt_flag_get(dma_channel_enum channelx, uint32_t flag);
<b>Function descriptions</b>	check DMA flag and interrupt enable bit is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
DMA_CHx(x=0..6)	DMA channel selection
<b>Input parameter{in}</b>	
flag	specify get which flag
DMA_INT_FLAG_FTF	full transfer finish interrupt flag of channel

DMA_INT_FLAGHTF	half transfer finish interrupt flag of channel
DMA_INT_FLAGERR	error interrupt flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
FlagStatus	SET or RESET

Example:

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

### **dma\_interrupt\_flag\_clear**

The description of `dma_interrupt_flag_clear` is shown as below:

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

<b>Function name</b>	dma_interrupt_flag_clear
<b>Function prototype</b>	void dma_interrupt_flag_clear(dma_channel_enum channelx, uint32_t flag);
<b>Function descriptions</b>	clear DMA a channel flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
channelx	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
DMA_CHx( x=0..6)	DMA channel selection
<b>Input parameter{in}</b>	
flag	specify get which flag
DMA_INT_FLAG_G	global interrupt flag of channel
DMA_INT_FLAG_FTF	full transfer finish interrupt flag of channel
DMA_INT_FLAG_HTF	half transfer finish interrupt flag of channel
DMA_INT_FLAG_ERR	error interrupt flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

}

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

The description of `dma_interrupt_enable` is shown as below:

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

<b>Function name</b>	dma_interrupt_enable
<b>Function prototype</b>	void dma_interrupt_enable(dma_channel_enum channelx, uint32_t source);
<b>Function descriptions</b>	enable DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
<b>DMA_CHx( x=0..6)</b>	DMA channel selection
<b>Input parameter{in}</b>	
<b>source</b>	DMA interrupt source
<b>DMA_INT_FTF</b>	full transfer finish interrupt of channel
<b>DMA_INT_HTF</b>	half transfer finish interrupt of channel
<b>DMA_INT_ERR</b>	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 3-185. 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>	specify which DMA channel to set, refer to <a href="#">Table 3-157. Enum dma_channel_enum</a> .
<b>DMA_CHx( x=0..6)</b>	DMA channel selection

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

Example:

```
/* disable DMA channel0 interrupt */

dma_interrupt_disable(DMA_CH0, DMA_INT_FTF);
```

## 3.10. EXTI

EXTI is the interrupt/event controller in the MCU. It contains up to 24 independent edge detectors and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter [3.10.1](#), the EXTI firmware functions are introduced in chapter [3.10.2](#).

### 3.10.1. Descriptions of Peripheral registers

EXTI registers are listed in the table shown as below:

**Table 3-186. EXTI Registers**

Registers	Descriptions
EXTI_INTEN	Interrupt enable register
EXTI_EVENT	Event enable register
EXTI_RTEN	Rising edge trigger enable register
EXTI_FTEN	Falling edge trigger enable register
EXTI_SWIEV	Software interrupt event register
EXTI_PD	Pending register

### 3.10.2. Descriptions of Peripheral functions

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

#### exti\_line\_enum

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

Member name	Function description
EXTI_0	EXTI line 0
EXTI_1	EXTI line 1

EXTI_2	EXTI line 2
EXTI_3	EXTI line 3
EXTI_4	EXTI line 4
EXTI_5	EXTI line 5
EXTI_6	EXTI line 6
EXTI_7	EXTI line 7
EXTI_8	EXTI line 8
EXTI_9	EXTI line 9
EXTI_10	EXTI line 10
EXTI_11	EXTI line 11
EXTI_12	EXTI line 12
EXTI_13	EXTI line 13
EXTI_14	EXTI line 14
EXTI_15	EXTI line 15
EXTI_16	EXTI line 16
EXTI_17	EXTI line 17
EXTI_18	EXTI line 18
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\_mode\_enum**

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

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

### **exti\_trig\_type\_enum**

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

Member name	Function description
EXTI_TRIG_RISING	EXTI rising edge trigger
EXTI_TRIG_FALLING	EXTI falling edge trigger
EXTI_TRIG_BOTH	EXTI rising and falling edge trigger

**Table 3-190. EXTI firmware function**

Function name	Function description
exti_deinit	reset EXTI, reset the value of all EXTI registers into initial values
exti_init	initialize the EXTI, enable the configuration of EXTI initialize
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 line x software interrupt
exti_software_interrupt_disable	disable EXTI line x software interrupt
exti_flag_get	get EXTI line x pending flag
exti_flag_clear	clear EXTI line x pending flag
exti_interrupt_flag_get	get EXTI line x flag when the interrupt flag is set
exti_interrupt_flag_clear	clear EXTI line x pending flag

## exti\_deinit

The description of exti\_deinit is shown as below:

**Table 3-191. Function exti\_deinit**

Function name	exti_deinit
Function prototype	void exti_deinit(void);
Function descriptions	reset EXTI, reset the value of all EXTI registers into initial values
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 3-192. 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);
<b>Function descriptions</b>	initialize the EXTI, enable the configuration of EXTI initialize
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x
<i>EXTI_x</i>	x=0..19,21,22
<b>Input parameter{in}</b>	
<b>mode</b>	EXTI mode
<i>EXTI_INTERRUPT</i>	interrupt mode
<i>EXTI_EVENT</i>	event mode
<b>Input parameter{in}</b>	
<b>trig_type</b>	trigger type
<i>EXTI_TRIG_RISING</i>	rising edge trigger
<i>EXTI_TRIG_FALLING</i>	falling edge trigger
<i>EXTI_TRIG_BOTH</i>	rising edge and falling edge trigger
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **exti\_interrupt\_enable**

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

**Table 3-193. 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
<i>EXTI_x</i>	x=0,1,2..27
<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 3-194. 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 interrupt from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x
<i>EXTI_x</i>	x=0,1,2..27
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
exti_interrupt_disable(EXTI_0);
```

### **exti\_event\_enable**

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

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

<b>Function name</b>	exti_event_enable
<b>Function prototype</b>	void exti_event_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable the events from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x
<i>EXTI_x</i>	x=0,1,2..27
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

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

```
exti_event_enable(EXTI_0);
```

### **exti\_event\_disable**

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

**Table 3-196. Function exti\_event\_disable**

<b>Function name</b>	exti_event_disable
<b>Function prototype</b>	void exti_event_disable(exti_line_enum linex);
<b>Function descriptions</b>	disable the events from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
linex	EXTI line x
EXTI_x	x=0,1,2..27
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
exti_event_disable(EXTI_0);
```

### **exti\_software\_interrupt\_enable**

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

**Table 3-197. Function exti\_software\_interrupt\_enable**

<b>Function name</b>	exti_software_interrupt_enable
<b>Function prototype</b>	void exti_software_interrupt_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable EXTI software interrupt event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
linex	EXTI line x
EXTI_x	0..19,21,22
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

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

```
exti_software_interrupt_enable(EXTI_0);
```

### **exti\_software\_interrupt\_disable**

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

**Table 3-198. Function exti\_software\_interrupt\_disable**

<b>Function name</b>	exti_software_interrupt_disable
<b>Function prototype</b>	void exti_software_interrupt_disable(exti_line_enum linex);
<b>Function descriptions</b>	disable EXTI software interrupt event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x
<i>EXTI_x</i>	x=0..19,21,22
<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 3-199. 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 pending flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x
<i>EXTI_x</i>	x=0..19,21,22
<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 3-200. 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 pending flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
linex	EXTI line x
EXTI_x	x=0..19,21,22
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* clear EXTI line 0 flag status */

exti_flag_clear(EXTI_0);

```

### **exti\_interrupt\_flag\_get**

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

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

<b>Function name</b>	exti_interrupt_flag_get
<b>Function prototype</b>	FlagStatus exti_interrupt_flag_get(exti_line_enum linex);
<b>Function descriptions</b>	get EXTI line x flag when the interrupt flag is set
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
linex	EXTI line x
EXTI_x	x=0..19,21,22
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```

/* get EXTI line 0 interrupt flag status */

FlagStatus state = exti_interrupt_flag_get(EXTI_0);

```

### **exti\_interrupt\_flag\_clear**

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

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

<b>Function name</b>	exti_interrupt_flag_clear
<b>Function prototype</b>	void exti_interrupt_flag_clear(exti_line_enum linex);
<b>Function descriptions</b>	clear EXTI line x pending flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
linex	EXTI line x
EXTI_x	x=0..19,21,22
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* clear EXTI line 0 interrupt flag status */

exti_interrupt_flag_clear(EXTI_0);

```

## **3.11. FMC**

There is flash controller and option byte for GD32F3x0 series. The FMC registers are listed in chapter [3.11.1](#) the FMC firmware functions are introduced in chapter [3.11.2](#).

### **3.11.1. Descriptions of Peripheral registers**

FMC registers are listed in the table shown as below:

**Table 3-203. FMC Registers**

Registers	Descriptions
FMC_WS	FMC wait state register
FMC_KEY	FMC unlock key register
FMC_OBKEY	FMC option bytes unlock key register
FMC_STAT	FMC status register
FMC_CTL	FMC control register
FMC_ADDR	FMC address register
FMC_OBSTAT	FMC option bytes status register

Registers	Descriptions
FMC_WP	FMC write protection register
FMC_PID	FMC product ID register

### 3.11.2. Descriptions of Peripheral functions

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

**Table 3-204. FMC firmware function**

Function name	Function description
fmc_unlock	unlock the main FMC operation
fmc_lock	lock the main FMC operation
fmc_wscnt_set	set the wait state counter value
fmc_wait_state_enable	fmc wait state enable
fmc_wait_state_disable	fmc_wait_state_enable
fmc_page_erase	erase FMC page
fmc_mass_erase	erase FMC whole chip
fmc_halfword_program	FMC program half word at the corresponding address
fmc_word_program	FMC program a word at the corresponding address
fmc_word_reprogram	FMC program a word at the corresponding address without erasing
ob_unlock	unlock the option byte operation
ob_lock	lock the option byte operation
ob_reset	reload the option byte and generate a system reset
ob_erase	erase the option byte
ob_write_protection_enable	enable option byte write protection (OB_WP)
ob_security_protection_config	configure read out protect
ob_user_write	write the FMC option byte user
ob_data_program	write the FMC option byte data
ob_user_get	get the FMC option byte OB_USER
ob_data_get	get the FMC option byte OB_DATA
ob_write_protection_get	get the FMC option byte write protection
ob_obstat_plevel_get	get the value of FMC option byte security protection level (PLEVEL) in FMC_OBSTAT register
fmc_interrupt_enable	enable FMC interrupt
fmc_interrupt_disable	disable FMC interrupt
fmc_flag_get	get flag set or reset
fmc_flag_clear	clear the FMC pending flag
fmc_interrupt_flag_get	get interrupt flag set or reset
fmc_interrupt_flag_clear	clear the FMC interrupt pending flag by writing 1
fmc_state_get	return the FMC state
fmc_ready_wait	check FMC ready or not
ob_parm_get	get current option byte value

Function name	Function description
ob_value_modify	modify the target option byte depending on the original value

## Enum fmc\_state\_enum

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

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

## Structure ob\_parm\_struct

**Table 3-206. Structure ob\_parm\_struct**

Member name	Function description
spc	option byte parameter spc
user	option byte parameter user
data0	option byte parameter data0
data1	option byte parameter data1
wp0	option byte parameter wp0
wp1	option byte parameter wp1

## fmc\_unlock

The description of fmc\_unlock is shown as below:

**Table 3-207. Function fmc\_unlock**

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

Example:

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

`fmc_unlock( );`

### **fmc\_lock**

The description of `fmc_lock` is shown as below:

**Table 3-208. Function fmc\_lock**

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

Example:

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

fmc_lock();
```

### **fmc\_wscnt\_set**

The description of `fmc_wscnt_set` is shown as below:

**Table 3-209. Function fmc\_wscnt\_set**

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

Example:

---

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

```
fmc_wscnt_set (WS_WSCNT_1);
```

### **fmc\_wait\_state\_enable**

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

**Table 3-210. Function fmc\_wscnt\_set**

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

Example:

```
/* fmc wait state enable */
fmc_wait_state_enable ( );
```

### **fmc\_wait\_state\_disable**

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

**Table 3-211. Function fmc\_wscnt\_set**

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

Example:

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

```
fmc_wait_state_disable ( );
```

### **fmc\_page\_erase**

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

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

<b>Function name</b>	fmc_page_erase
<b>Function prototype</b>	fmc_state_enum fmc_page_erase(uint32_t page_address);
<b>Function descriptions</b>	erase page
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
page_address	the page address to be erased
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-205. Enum fmc_state_enum</a>

Example:

```
/* erase page */
fmc_state_enum state = fmc_page_erase ( 0x08004000);
```

### **fmc\_mass\_erase**

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

**Table 3-213. Function fmc\_mass\_erase**

<b>Function name</b>	fmc_mass_erase
<b>Function prototype</b>	fmc_state_enum fmc_mass_erase(void );
<b>Function descriptions</b>	erase whole chip
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-205. Enum fmc_state_enum</a>

Example:

```
/* erase whole chip */
```

```
fmc_state_enum state = fmc_mass_erase ( );
```

## **fmc\_word\_program**

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

**Table 3-214. Function fmc\_word\_program**

<b>Function name</b>	fmc_word_program
<b>Function prototype</b>	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
<b>Function descriptions</b>	program a word at the corresponding address
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
address	the address to program
data	the data to program
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-205. Enum fmc_state_enum</a>

Example:

```
/* program a word at the corresponding address */
fmc_state_enum fmc_state = fmc_word_program ( 0x08004000,0xaabbccdd);
```

## **fmc\_halfword\_program**

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

**Table 3-215. Function fmc\_halfword\_program**

<b>Function name</b>	fmc_halfword_program
<b>Function prototype</b>	fmc_state_enum fmc_halfword_program(uint32_t address, uint16_t data);
<b>Function descriptions</b>	program a half word at the corresponding address
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
address	the address to program
data	the data to program
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-205. Enum fmc_state_enum</a>

Example:

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

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

## **fmc\_word\_reprogram**

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

**Table 3-216. Function fmc\_word\_reprogram**

<b>Function name</b>	fmc_word_reprogram
<b>Function prototype</b>	fmc_state_enum fmc_word_reprogram(uint32_t address, uint32_t data);
<b>Function descriptions</b>	program a word at the corresponding address without erasing
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
address	the address to program
data	the data to program
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-205. Enum fmc_state_enum</a>

Example:

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

fmc_state_enum fmc_state ;

fmc_state = fmc_word_program ( 0x08004000, 0x01234567);

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

## **ob\_unlock**

The description of ob\_unlock is shown as below:

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

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

-	-
---	---

Example:

```
/* unlock the option byte operation */

ob_unlock( );
```

### ob\_lock

The description of ob\_lock is shown as below:

**Table 3-218. 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:

```
/* lock the option byte operation */

ob_lock( );
```

### ob\_reset

The description of ob\_reset is shown as below:

**Table 3-219. Function ob\_reset**

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

Example:

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

ob_reset();
```

### **ob\_erase**

The description of ob\_erase is shown as below:

**Table 3-220. Function ob\_erase**

<b>Function name</b>	ob_erase
<b>Function prototype</b>	void ob_erase(void);
<b>Function descriptions</b>	erase the option byte
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-205. Enum fmc_state_enum</a>

Example:

```
/* 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 3-221. Function ob\_write\_protection\_enable**

<b>Function name</b>	ob_write_protection_enable
<b>Function prototype</b>	fmc_state_enum ob_write_protection_enable(uint16_t ob_wp);
<b>Function descriptions</b>	enable option byte write protection (OB_WP)
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
ob_wp	write protection configuration data
<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 3-205. Enum fmc_state_enum</a>

Example:

```
/* enable write protection */

fmc_state_enum fmc_state = ob_write_protection_enable (0x7C);
```

### **ob\_security\_protection\_config**

The description of `ob_security_protection_config` is shown as below:

**Table 3-222. Function `ob_security_protection_config`**

<b>Function name</b>	ob_security_protection_config
<b>Function prototype</b>	fmc_state_enum ob_security_protection_config (uint16_t ob_spc);
<b>Function descriptions</b>	configure security protection
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>ob_spc</b>	specify security protection
<i>FMC_NSPC</i>	no security protection
<i>FMC_LSPC</i>	low security protection
<i>FMC_HSPC</i>	high security protection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-205. Enum <code>fmc_state_enum</code></a>

Example:

```
/* enable security protection */

fmc_state_enum fmc_state;

fmc_state = ob_security_protection_config (FMC_USPC);
```

### **ob\_user\_write**

The description of `ob_user_write` is shown as below:

**Table 3-223. Function `ob_user_write`**

<b>Function name</b>	ob_user_write
<b>Function prototype</b>	fmc_state_enum ob_user_write(uint8_t ob_user);
<b>Function descriptions</b>	program the FMC user option byte
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>ob_user</b>	user option byte
<i>OB_FWDGT_HW</i>	hardware free watchdog timer

<code>OB_DEEPSLEEP_RST</code>	no reset when entering deepsleep mode
<code>OB_STDBY_RST</code>	no reset when entering deepsleep mode
<code>OB_BOOT1_SET_1</code>	BOOT1 bit is 1
<code>OB_VDDA_DISABLE</code>	disable VDDA monitor
<code>OB_SRAM_PARITY_ENABLE</code>	enable SRAM parity check
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<code>fmc_state_enum</code>	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-205. Enum fmc_state enum</a>

Example:

```
/* program the FMC user option byte */

fmc_state_enum fmc_state = ob_user_write(OB_FWDGT_HW);
```

## ob\_data\_program

The description of `ob_data_program` is shown as below:

**Table 3-224. Function `ob_data_program`**

<b>Function name</b>	<code>ob_data_program</code>
<b>Function prototype</b>	<code>fmc_state_enum ob_data_program(uint32_t address, uint8_t data);</code>
<b>Function descriptions</b>	program the FMC data option byte
<b>Precondition</b>	<code>ob_unlock</code>
<b>The called functions</b>	<code>fmc_ready_wait</code>
<b>Input parameter{in}</b>	
<code>address</code>	OB_DATA_ADDR0 or OB_DATA_ADDR1
<b>Input parameter{in}</b>	
<code>data</code>	the byte to be programmed
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<code>fmc_state_enum</code>	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-205. Enum fmc_state enum</a>

Example:

```
/* program option bytes data */

fmc_state_enum fmc_state = ob_data_program (OB_DATA_ADDR0, 0x56);
```

## ob\_user\_get

The description of `ob_user_get` is shown as below:

**Table 3-225. Function ob\_user\_get**

<b>Function name</b>	ob_user_get
<b>Function prototype</b>	uint8_t ob_user_get(void);
<b>Function descriptions</b>	get OB_USER in register FMC_OBSTAT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	the FMC user option byte values(0x00 – 0xFF)

Example:

```
/* get the FMC user option byte */

uint8_t user = ob_user_get();
```

### **ob\_data\_get**

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

**Table 3-226. Function ob\_data\_get**

<b>Function name</b>	ob_data_get
<b>Function prototype</b>	uint16_t ob_data_get(void);
<b>Function descriptions</b>	get OB_DATA in register FMC_OBSTAT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	the FMC data option byte values(0x0 – 0xFFFF)

Example:

```
/* get the FMC data option byte */

uint16_t data = ob_data_get();
```

### **ob\_write\_protection\_get**

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

**Table 3-227. Function ob\_write\_protection\_get**

<b>Function name</b>	ob_write_protection_get
----------------------	-------------------------

<b>Function prototype</b>	uint16_t ob_write_protection_get(void);
<b>Function descriptions</b>	get the FMC option byte write protection (OB_WP) in register FMC_WP
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	the FMC write protection option byte value(0x0 – 0xFFFF)

Example:

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

uint16_t wp = ob_write_protection_get();
```

### ob\_obstat\_plevel\_get

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

**Table 3-228. Function ob\_obstat\_plevel\_get**

<b>Function name</b>	ob_obstat_plevel_get
<b>Function prototype</b>	uint32_t ob_obstat_plevel_get(void);
<b>Function descriptions</b>	get the value of FMC option byte security protection level (PLEVEL) in FMC_OBSTAT register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	the value of PLEVEL(0x0,0x01,0x03)

Example:

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

uint32_t obstat_plevel = ob_obstat_plevel_get();
```

### fmc\_interrupt\_enable

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

**Table 3-229. Function fmc\_interrupt\_enable**

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

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

Example:

```
/* enable FMC interrupt */
fmc_interrupt_enable(FMC_INT_END);
```

### fmc\_interrupt\_disable

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

**Table 3-230. Function fmc\_interrupt\_disable**

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

Example:

```
/* disable FMC interrupt */
fmc_interrupt_disable(FMC_INT_END);
```

### fmc\_flag\_get

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

**Table 3-231. Function fmc\_flag\_get**

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

Example:

```
/* get FMC flag */

FlagStatus flag = fmc_flag_get(FMC_FLAG_END);
```

### **fmc\_flag\_clear**

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

**Table 3-232. Function fmc\_flag\_clear**

<b>Function name</b>	fmc_flag_clear
<b>Function prototype</b>	void fmc_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear the FMC flag by writing 1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
flag	clear FMC flag
FMC_FLAG_PGERR	FMC operation error flag
FMC_FLAG_WPERR	FMC erase/program protection error flag
FMC_FLAG_END	FMC end of operation flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
fmc_flag_clear(FMC_FLAG_END);
```

### **fmc\_interrupt\_flag\_get**

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

**Table 3-233. Function fmc\_interrupt\_flag\_get**

<b>Function name</b>	fmc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus fmc_interrupt_flag_get(uint32_t int_flag);
<b>Function descriptions</b>	get interrupt flag set or reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>flag</i>	FMC flag
<i>FMC_FLAG_PGERR</i>	FMC operation error flag
<i>FMC_FLAG_WPERR</i>	FMC erase/program protection error flag
<i>FMC_FLAG_END</i>	FMC end of operation flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get FMC interrupt flag */
FlagStatus flag = fmc_interrupt_flag_get (FMC_FLAG_PGERR);
```

### **fmc\_interrupt\_flag\_clear**

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

**Table 3-234. Function fmc\_interrupt\_flag\_clear**

<b>Function name</b>	fmc_interrupt_flag_clear
<b>Function prototype</b>	void fmc_interrupt_flag_clear(uint32_t int_flag);
<b>Function descriptions</b>	clear the FMC interrupt pending flag by writing 1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>flag</i>	clear FMC flag
<i>FMC_FLAG_PGERR</i>	FMC operation error flag
<i>FMC_FLAG_WPERR</i>	FMC erase/program protection error flag
<i>FMC_FLAG_END</i>	FMC end of operation flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

--	--

Example:

```
/* clear FMC interrupt flag */

fmc_interrupt_flag_clear (FMC_FLAG_PGERR);
```

### fmc\_state\_get

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

**Table 3-235. Function fmc\_state\_get**

<b>Function name</b>	fmc_state_get
<b>Function prototype</b>	fmc_state_enum fmc_state_get(void);
<b>Function descriptions</b>	get the FMC state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-205. Enum fmc_state_enum</a>

Example:

```
/* get the FMC state */

fmc_state_enum state = fmc_state_get( );
```

### fmc\_ready\_wait

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

**Table 3-236. Function fmc\_ready\_wait**

<b>Function name</b>	fmc_ready_wait
<b>Function prototype</b>	fmc_state_enum fmc_ready_wait(uint32_t timeout);
<b>Function descriptions</b>	check whether FMC is ready or not
<b>Precondition</b>	-
<b>The called functions</b>	fmc_state_get()
<b>Input parameter{in}</b>	
timeout	timeout count
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-205. Enum fmc_state_enum</a>

Example:

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

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

### **ob\_parm\_get**

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

**Table 3-237. Function ob\_parm\_get**

<b>Function name</b>	ob_parm_get
<b>Function prototype</b>	void ob_parm_get(ob_parm_struct *ob_parm);
<b>Function descriptions</b>	get current option byte value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ob_parm</b>	pointer to option byte parameter struct <a href="#">Table 3-206. Structure ob_parm_struct</a>
<b>Output parameter{out}</b>	
<b>ob_parm</b>	pointer to option byte parameter struct <a href="#">Table 3-206. Structure ob_parm_struct</a>
<b>Return value</b>	
-	-

Example:

```
/* get current option byte value */

ob_parm_struct ob_parm;

ob_parm_get(&ob_parm);
```

### **ob\_value\_modify**

The description of ob\_value\_modify is shown as below:

**Table 3-238. Function ob\_value\_modify**

<b>Function name</b>	ob_value_modify
<b>Function prototype</b>	void ob_value_modify(uint32_t address, uint16_t value,ob_parm_struct *ob_parm);
<b>Function descriptions</b>	modify the target option byte depending on the original value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>address</b>	target option byte address

Input parameter{in}	
value	target option byte value
Input parameter{in}	
ob_parm	pointer to option byte parameter struct <a href="#">Table 3-206. Structure ob_parm_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

ob_parm_struct ob_parm;

ob_value_modify(OB_SPC_ADDR, (uint16_t)ob_spc ,&ob_parm);
```

## 3.12. 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 [3.12.1](#) the FWDGT firmware functions are introduced in chapter [3.12.2](#).

### 3.12.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

**Table 3-239. FWDGT Registers**

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

### 3.12.2. Descriptions of Peripheral functions

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

**Table 3-240. FWDGT firmware function**

Function name	Function description
fwdgt_write_enable	enable write access to FWDGT_PSC and FWDGT_RLD
fwdgt_write_disable	disable write access to FWDGT_PSC and FWDGT_RLD

Function name	Function description
fwdgt_enable	start the FWDGT counter
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 3-241. Function fwdgt\_write\_enable**

Function name	fwdgt_write_enable
Function prototype	void fwdgt_write_enable(void);
Function descriptions	enable write access to FWDGT_PSC and FWDGT_RLD
Precondition	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable write access to FWDGT_PSC and FWDGT_RLD */
fwdgt_write_enable();
```

### fwdgt\_write\_disable

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

**Table 3-242. Function fwdgt\_write\_disable**

Function name	fwdgt_write_disable
Function prototype	void fwdgt_write_disable(void);
Function descriptions	disable write access to FWDGT_PSC and FWDGT_RLD
Precondition	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

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

```
fwdgt_write_disable ( );
```

### **fwdgt\_enable**

The description of fwdgt\_enable is shown as below:

**Table 3-243. Function fwdgt\_enable**

<b>Function name</b>	fwdgt_enable
<b>Function prototype</b>	void fwdgt_enable(void);
<b>Function descriptions</b>	start the FWDGT counter
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* start the free watchdog timer counter */

fwdgt_enable ( );
```

### **fwdgt\_window\_value\_config**

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

**Table 3-244. Function fwdgt\_window\_value\_config**

<b>Function name</b>	fwdgt_window_value_config
<b>Function prototype</b>	ErrStatus fwdgt_window_value_config(uint16_t window_value);
<b>Function descriptions</b>	configure the FWDGT counter window value
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
window_value	window_value: specify window value(0x0000 - 0xFFFF)
<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 3-245. Function fwdgt\_counter\_reload**

<b>Function name</b>	fwdgt_counter_reload
<b>Function prototype</b>	void fwdgt_counter_reload(void);
<b>Function descriptions</b>	reload the counter of FWDGT
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### fwdgt\_config

The description of fwdgt\_config is shown as below:

**Table 3-246. Function fwdgt\_config**

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

Example:

```
/* config 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 3-247. Function fwdgt\_flag\_get**

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

Example:

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

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

## 3.13. GPIO/AFIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter [3.13.1](#), the GPIO firmware functions are introduced in chapter [3.13.2](#).

### 3.13.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

**Table 3-248. GPIO Registers**

Registers	Descriptions
GPIOx_CTL	GPIO port control register
GPIOx_OMODE	GPIO port output mode register
GPIOx_OSPEED0	GPIO port output speed register 0

Registers	Descriptions
GPIOx_PUD	GPIO port pull-up/pull-down register
GPIOx_ISTAT	GPIO port input status register
GPIOx_OCTL	GPIO port output control register
GPIOx_BOP	GPIO port bit operation register
GPIOx_LOCK	GPIO port configuration lock register
GPIO_AFSEL0	GPIO alternate function selected register 0
GPIO_AFSEL1	GPIO alternate function selected register 1
GPIO_BC	GPIO bit clear register
GPIO_TG	GPIO port bit toggle register
GPIO_OSPEED1	GPIO port output speed register 1

### 3.13.2. Descriptions of Peripheral functions

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

**Table 3-249. GPIO firmware function**

Function name	Function description
gpio_deinit	reset GPIO port
gpio_mode_set	set GPIO mode
gpio_output_options_set	set GPIO output type and speed
gpio_bit_set	set GPIO pin bit
gpio_bit_reset	reset GPIO pin bit
gpio_bit_write	write data to the specified GPIO pin
gpio_port_write	write data to the specified GPIO port
gpio_input_bit_get	get GPIO pin input status
gpio_input_port_get	get GPIO port input status
gpio_output_bit_get	get GPIO pin output status
gpio_output_port_get	get GPIO port output status
gpio_af_set	set GPIO alternate function
gpio_pin_lock	lock GPIO pin
gpio_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 3-250. 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 3-251. 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	rcu_periph_reset_enable / rcu_periph_reset_disable
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
GPIO_PUPD_PULLUP	with pull-up resistor
GPIO_PUPD_PULLDOWN	with pull-down resistor
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	All pins

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config PA0 as 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 3-252. Function gpio\_output\_options\_set**

Function name	gpio_output_options_set
Function prototype	void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t pin);
Function descriptions	set GPIO output type and speed
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C,D,F)
Input parameter{in}	
otype	gpio pin output mode
GPIO_OTYPE_PP	push pull mode
GPIO_OTYPE_OD	open drain mode
Input parameter{in}	
speed	gpio pin output max speed
GPIO_OSPEED_2MHZ	output max speed 2MHz
GPIO_OSPEED_10MH Z	output max speed 10MHz
GPIO_OSPEED_50MH Z	output max speed 50MHz
GPIO_OSPEED_MAX	GPIO very high output speed, max speed more than 50MHz
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

### gpio\_bit\_set

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

**Table 3-253. 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>	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C,D,F)
<b>Input parameter{in}</b>	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set PA0*/

gpio_bit_set(GPIOA, GPIO_PIN_0);
```

### gpio\_bit\_reset

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

**Table 3-254. 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>	
gpio_periph	GPIO port
GPIOx	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	
-	-

Example:

```
/* reset PA0*/
gpio_bit_set (GPIOA, GPIO_PIN_0);
```

### gpio\_bit\_write

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

**Table 3-255. 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>	-
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
Input parameter{in}	
<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 3-256. Function gpio\_port\_write**

<b>Function name</b>	gpio_port_write
<b>Function prototype</b>	void gpio_port_write(uint32_t gpio_periph,uint16_t data);
<b>Function descriptions</b>	write data to the specified GPIO port
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C,D,F)
<b>Input parameter{in}</b>	
data	specify the value to be written to the port output data register
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*write 1010 0101 1010 0101 to Port A */
gpio_port_write (GPIOA, 0xA5A5);
```

### gpio\_input\_bit\_get

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

**Table 3-257. Function gpio\_input\_bit\_get**

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

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

Example:

```
/* get status of PA0 */
FlagStatus bit_state = gpio_input_bit_get(GPIOA, GPIO_PIN_0);

gpio_input_port_get
```

The description of `gpio_input_port_get` is shown as below:

**Table 3-258. Function `gpio_input_port_get`**

<b>Function name</b>	gpio_input_port_get
<b>Function prototype</b>	uint16_t gpio_input_port_get(uint32_t gpio_periph);
<b>Function descriptions</b>	get GPIO all pins input status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C,D,F)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	0x0000-0xFFFF

Example:

```
/* get input value of Port A */
uint16_t port_state;
port_state = gpio_input_bit_get(GPIOA);
```

### **gpio\_output\_bit\_get**

The description of `gpio_output_bit_get` is shown as below:

**Table 3-259. 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>	
gpio_periph	GPIO port
GPIOx	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 3-260. 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 3-261. Function gpio\_af\_set**

<b>Function name</b>	gpio_af_set
<b>Function prototype</b>	void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);

<b>Function descriptions</b>	set GPIO alternate function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<b>GPIOx</b>	GPIOx(x = A,B,C)
<b>Input parameter{in}</b>	
<b>alt_func_num</b>	GPIO pin af function, please refer to specific device datasheet
<b>GPIO_AF_0</b>	TIMER2, TIMER13, TIMER14, TIMER16, SPI0, SPI1, I2S0, CK_OUT, USART0, CEC, IFRP, TSI, CTC, I2C0, I2C1, SWDIO, SWCLK
<b>GPIO_AF_1</b>	USART0, USART1, TIMER2, TIMER14, I2C0, I2C1, IFRP, CEC
<b>GPIO_AF_2</b>	TIMER0, TIMER1, TIMER15, TIMER16, I2S0
<b>GPIO_AF_3</b>	TSI, I2C0, TIMER14
<b>GPIO_AF_4 (port A,B only)</b>	USART1, I2C0, I2C1, TIMER13
<b>GPIO_AF_5 (port A,B only)</b>	TIMER15, TIMER16, USBFS, I2S0
<b>GPIO_AF_6 (port A,B only)</b>	CTC, SPI1
<b>GPIO_AF_7 (port A,B only)</b>	CMP0, CMP1
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<b>GPIO_PIN_x</b>	GPIO_PIN_x(x=0..15)
<b>GPIO_PIN_ALL</b>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*set PA0 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 3-262. Function gpio\_pin\_lock**

<b>Function name</b>	gpio_pin_lock
<b>Function prototype</b>	void gpio_pin_lock(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	lock GPIO pin bit
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<b>GPIOx</b>	GPIOx(x = A,B)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<b>GPIO_PIN_x</b>	GPIO_PIN_x(x=0..15)
<b>GPIO_PIN_ALL</b>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* lock PA0*/
gpio_pin_lock (GPIOA, GPIO_PIN_0);
```

### gpio\_bit\_toggle

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

**Table 3-263. Function gpio\_bit\_toggle**

<b>Function name</b>	gpio_bit_toggle
<b>Function prototype</b>	void gpio_bit_toggle(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	toggle GPIO pin status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<b>GPIOx</b>	GPIOx(x = A,B,C,D,F)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<b>GPIO_PIN_x</b>	GPIO_PIN_x(x=0..15)
<b>GPIO_PIN_ALL</b>	GPIO_PIN_ALL
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* toggle PA0 */
gpio_bit_toggle (GPIOA, GPIO_PIN_0);
```

### gpio\_port\_toggle

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

**Table 3-264. 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>	-
<b>Input parameter{in}</b>	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C,D,F)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* toggle GPIOA*/
gpio_port_toggle (GPIOA);
```

## 3.14. 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 [3.14.1](#), the I2C firmware functions are introduced in chapter [3.14.2](#)

### 3.14.1. Descriptions of Peripheral registers

I2C registers are listed in the table shown as below:

**Table 3-265. I2C Registers**

Registers	Descriptions
I2C_CTL0	Control register 0
I2C_CTL1	Control register 1
I2C_SADDR0	Slave address register 0
I2C_SADDR1	Slave address register 1
I2C_DATA	Transfer buffer register
I2C_STAT0	Transfer status register 0
I2C_STAT1	Transfer status register 1
I2C_CKCFG	Clock configure register

Registers	Descriptions
I2C_RT	Rise time register
I2C_FMPCFG	Fast mode plus configure register

### 3.14.2. Descriptions of Peripheral functions

I2C firmware functions are listed in the table shown as below:

**Table 3-266. I2C firmware function**

Function name	Function description
i2c_deinit	reset I2C
i2c_clock_config	configure I2C clock
i2c_mode_addr_config	configure I2C address
i2c_smbus_type_config	SMBus type selection
i2c_ack_config	whether or not to send an ACK
i2c_ackpos_config	configure I2C position of ACK and PEC when receiving
i2c_master_addressing	master sends slave address
i2c_dualaddr_enable	enable dual-address mode
i2c_dualaddr_disable	disable dual-address mode
i2c_enable	enable I2C
i2c_disable	disable I2C
i2c_start_on_bus	generate a START condition on I2C bus
i2c_stop_on_bus	generate a STOP condition on I2C bus
i2c_data_transmit	I2C transmit data function
i2c_data_receive	I2C receive data function
i2c_dma_enable	enable I2C DMA mode
i2c_dma_last_transfer_config	configure whether next DMA EOT is DMA last transfer or not
i2c_stretch_scl_low_config	whether to stretch SCL low when data is not ready in slave mode
i2c_slave_response_to_gcall_config	whether or not to response to a general call
i2c_software_reset_config	software reset I2C
i2c_pec_enable	whether to enable I2C PEC calculation or not
i2c_pec_transfer_enable	I2C whether to transfer PEC value
i2c_pec_value_get	packet error checking value
i2c_smbus_issue_alert	I2C issue alert through SMBA pin
i2c_smbus_arp_enable	whether ARP is enabled under SMBus
i2c_flag_get	check I2C flag is set or not
i2c_flag_clear	clear I2C flag
i2c_interrupt_enable	enable I2C interrupt
i2c_interrupt_disable	disable I2C interrupt
i2c_interrupt_flag_get	check I2C interrupt flag
i2c_interrupt_flag_clear	clear I2C interrupt flag

**Enum i2c\_flag\_enum**
**Table 3-267. Enum i2c\_flag\_enum**

<b>Member name</b>	<b>Function description</b>
I2C_FLAG_SBSEND	start condition sent out in master mode
I2C_FLAG_ADDSEND	address is sent in master mode or received and matches in slave mode
I2C_FLAG_BTC	byte transmission finishes
I2C_FLAG_ADD10SEND	header of 10-bit address is sent in master mode
I2C_FLAG_STPDET	stop condition detected in slave mode
I2C_FLAG_RBNE	I2C_DATA is not empty during receiving
I2C_FLAG_TBE	I2C_DATA is empty during transmitting
I2C_FLAG_BERR	a bus error occurs indication a unexpected start or stop condition on I2C bus
I2C_FLAG_LOSTARB	arbitration lost in master mode
I2C_FLAG_AERR	acknowledge error
I2C_FLAG_OUERR	over-run or under-run situation occurs in slave mode
I2C_FLAG_PECERR	PEC error when receiving data
I2C_FLAG_SMBTO	timeout signal in SMBus mode
I2C_FLAG_SMBALT	SMBus alert status
I2C_FLAG_MASTER	a flag indicating whether I2C block is in master or slave mode
I2C_FLAG_I2CBSY	busy flag
I2C_FLAG_TR	whether the I2C is a transmitter or a receiver
I2C_FLAG_RXGC	general call address (00h) received
I2C_FLAG_DEFSMB	default address of SMBus device
I2C_FLAG_HSTSMB	SMBus host header detected in slave mode
I2C_FLAG_DUMOD	dual flag in slave mode indicating which address is matched in dual-address mode

**Enum i2c\_interrupt\_flag\_enum**
**Table 3-268. Enum i2c\_interrupt\_flag\_enum**

<b>Member name</b>	<b>Function description</b>
I2C_INT_FLAG_SBSEND	start condition sent out in master mode interrupt flag
I2C_INT_FLAG_ADDSEND	address is sent in master mode or received and matches in slave mode interrupt flag
I2C_INT_FLAG_BTC	byte transmission finishes
I2C_INT_FLAG_ADD10SEND	header of 10-bit address is sent in master mode interrupt flag
I2C_INT_FLAG_STPDET	stop condition detected in slave mode interrupt flag
I2C_INT_FLAG_RBNE	I2C_DATA is not empty during receiving interrupt flag
I2C_INT_FLAG_TBE	I2C_DATA is empty during transmitting interrupt flag
I2C_INT_FLAG_BERR	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag

Member name	Function description
I2C_INT_FLAG_LOSTARB	arbitration lost in master mode interrupt flag
I2C_INT_FLAG_AERR	acknowledge error interrupt flag
I2C_INT_FLAG_OUERR	over-run or under-run situation occurs in slave mode interrupt flag
I2C_INT_FLAG_PECERR	PEC error when receiving data interrupt flag
I2C_INT_FLAG_SMBTO	timeout signal in SMBus mode interrupt flag
I2C_INT_FLAG_SMBALT	SMBus Alert status interrupt flag

### Enum i2c\_interrupt\_enum

**Table 3-269. Enum i2c\_interrupt\_enum**

Member name	Function description
I2C_INT_ERR	error interrupt enable
I2C_INT_EV	event interrupt enable
I2C_INT_BUF	buffer interrupt enable

### i2c\_deinit

The description of i2c\_deinit is shown as below:

**Table 3-270. Function i2c\_deinit**

Function name	i2c_deinit	
Function prototype	void i2c_deinit(uint32_t i2c_periph);	
Function descriptions	reset I2C	
Precondition	-	
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable	
Input parameter{in}		
i2c_periph	I2C peripheral	
I2Cx	(x=0,1)	
Output parameter{out}		
-	-	
Return value		
-	-	

Example:

```
/* reset I2C0 */
i2c_deinit(I2C0);
```

### i2c\_clock\_config

The description of i2c\_clock\_config is shown as below:

**Table 3-271. Function i2c\_clock\_config**

Function name	i2c_clock_config
---------------	------------------

<b>Function prototype</b>	void i2c_clock_config(uint32_t i2c_periph, uint32_t clkspeed, uint32_t dutycyc);
<b>Function descriptions</b>	configure I2C clock
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
clkspeed	i2c clock speed
<b>Input parameter{in}</b>	
dutycyc	duty cycle in fast mode
I2C_DTCY_2	T_low/T_high=2
I2C_DTCY_16_9	T_low/T_high=16/9
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure I2C0 clock speed as 100KHz */
i2c_clock_config(I2C0, 100000, I2C_DTCY_2);
```

### **i2c\_mode\_addr\_config**

The description of i2c\_mode\_addr\_config is shown as below:

**Table 3-272. Function i2c\_mode\_addr\_config**

<b>Function name</b>	i2c_mode_addr_config
<b>Function prototype</b>	void i2c_mode_addr_config(uint32_t i2c_periph, uint32_t mode, uint32_t addformat, uint32_t addr);
<b>Function descriptions</b>	configure I2C address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
mode	I2C mode select
I2C_I2CMODE_ENABL E	I2C mode
I2C_SMBUSMODE_EN ABLE	SMBus mode

Input parameter{in}	
<b>addformat</b>	7bits or 10bits
<i>I2C_ADDFORMAT_7BITS</i>	7bits
<i>I2C_ADDFORMAT_10BITS</i>	10bits
Input parameter{in}	
<b>addr</b>	I2C address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure I2C0 address as 0x82, using 7 bits */

i2c_mode_addr_config(I2C0, I2C_I2CMODE_ENABLE, I2C_ADDFORMAT_7BITS, 0x82);
```

### i2c\_smbus\_type\_config

The description of i2c\_smbus\_type\_config is shown as below:

**Table 3-273. Function i2c\_smbus\_type\_config**

<b>Function name</b>	i2c_smbus_type_config
<b>Function prototype</b>	void i2c_smbus_type_config(uint32_t i2c_periph, uint32_t type);
<b>Function descriptions</b>	SMBus type selection
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
<b>type</b>	Device or host
<i>I2C_SMBUS_DEVICE</i>	device
<i>I2C_SMBUS_HOST</i>	host
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config I2C0 as SMBUS host type*/

i2c_smbus_type_config (I2C0, I2C_SMBUS_HOST);
```

### i2c\_ack\_config

The description of i2c\_ack\_config is shown as below:

**Table 3-274. Function i2c\_ack\_config**

<b>Function name</b>	i2c_ack_config
<b>Function prototype</b>	void i2c_ack_config(uint32_t i2c_periph, uint32_t ack);
<b>Function descriptions</b>	whether or not to send an ACK
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
ack	whether or not to send an ACK
I2C_ACK_ENABLE	ACK will be sent
I2C_ACK_DISABLE	ACK will not be sent
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 will send ACK */
i2c_ack_config (I2C0, I2C_ACK_ENABLE);
```

### i2c\_ackpos\_config

The description of i2c\_ackpos\_config is shown as below:

**Table 3-275. Function i2c\_ackpos\_config**

<b>Function name</b>	i2c_ackpos_config
<b>Function prototype</b>	void i2c_ackpos_config(uint32_t i2c_periph, uint32_t pos);
<b>Function descriptions</b>	configure I2C position of ACK and PEC when receiving
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
pos	ACK position
I2C_ACKPOS_CURRENT	whether to send ACK or not for the current
I2C_ACKPOS_NEXT	whether to send ACK or not for the next byte

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* The ACK of I2C0 is send for the current frame */

i2c_ackpos_config(I2C0, I2C_ACKPOS_CURRENT);
```

### i2c\_master\_addressing

The description of i2c\_master\_addressing is shown as below:

**Table 3-276. Function i2c\_master\_addressing**

Function name	i2c_master_addressing
Function prototype	void i2c_master_addressing(uint32_t i2c_periph, uint32_t addr, uint32_t trandirection);
Function descriptions	master sends slave address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
addr	slave address
Input parameter{in}	
trandirection	transmitter or receiver
I2C_TRANSMITTER	transmitter
I2C_RECEIVER	receiver
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* send slave address to I2C bus and I2C0 act as receiver */

i2c_master_addressing(I2C0, 0x82, I2C_RECEIVER);
```

### i2c\_dualaddr\_enable

The description of i2c\_dualaddr\_enable is shown as below:

**Table 3-277. Function i2c\_dualaddr\_enable**

Function name	i2c_dualaddr_enable
---------------	---------------------

<b>Function prototype</b>	void i2c_dualaddr_enable(uint32_t i2c_periph, uint32_t addr)
<b>Function descriptions</b>	enable dual-address mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
addr	second address in dual-address mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 dual-address*/
i2c_dualaddr_enable (I2C0, 0x80);
```

### i2c\_dualaddr\_disable

The description of i2c\_dualaddr\_disable is shown as below:

**Table 3-278. Function i2c\_dualaddr\_disable**

<b>Function name</b>	i2c_dualaddr_disable
<b>Function prototype</b>	void i2c_dualaddr_disable(uint32_t i2c_periph)
<b>Function descriptions</b>	disable dual-address mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C0 dual-address */
i2c_dualaddr_disable (I2C0);
```

### i2c\_enable

The description of i2c\_enable is shown as below:

**Table 3-279. 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>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 */
i2c_enable (I2C0);
```

### **i2c\_disable**

The description of i2c\_disable is shown as below:

**Table 3-280. Function i2c\_disable**

<b>Function name</b>	i2c_disable
<b>Function prototype</b>	void i2c_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C0 */
i2c_disable (I2C0);
```

### **i2c\_start\_on\_bus**

The description of i2c\_start\_on\_bus is shown as below:

**Table 3-281. Function i2c\_start\_on\_bus**

<b>Function name</b>	i2c_start_on_bus
<b>Function prototype</b>	void i2c_start_on_bus(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a START condition on I2C bus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 send a start condition to I2C bus */
i2c_start_on_bus (I2C0);
```

### **i2c\_stop\_on\_bus**

The description of i2c\_stop\_on\_bus is shown as below:

**Table 3-282. Function i2c\_stop\_on\_bus**

<b>Function name</b>	i2c_stop_on_bus
<b>Function prototype</b>	void i2c_stop_on_bus(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a STOP condition on I2C bus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 generate a STOP condition to I2C bus */
i2c_stop_on_bus (I2C0);
```

### **i2c\_data\_transmit**

The description of i2c\_data\_transmit is shown as below:

**Table 3-283. Function i2c\_data\_transmit**

<b>Function name</b>	i2c_data_transmit
<b>Function prototype</b>	void i2c_data_transmit(uint32_t i2c_periph, uint8_t data);
<b>Function descriptions</b>	I2C transmit data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
data	transmit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 transmit data */

i2c_data_transmit (I2C0, 0x80);
```

### i2c\_data\_receive

The description of i2c\_data\_receive is shown as below:

**Table 3-284. Function i2c\_data\_receive**

<b>Function name</b>	i2c_data_receive
<b>Function prototype</b>	uint8_t i2c_data_receive(uint32_t i2c_periph);
<b>Function descriptions</b>	I2C receive data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint8_t	0x00..0xFF

Example:

```
/* I2C0 receive data */

uint8_t i2c_receiver;

i2c_receiver = i2c_data_receive(I2C0);
```

### i2c\_dma\_enable

The description of i2c\_dma\_enable is shown as below:

**Table 3-285. Function i2c\_dma\_enable**

<b>Function name</b>	i2c_dma_enable
<b>Function prototype</b>	void i2c_dma_enable(uint32_t i2c_periph, uint32_t dmastate);
<b>Function descriptions</b>	enable I2C DMA mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
dmastate	on or off
I2C_DMA_ON	DMA mode enable
I2C_DMA_OFF	DMA mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 DMA mode enable */
i2c_dma_enable (I2C0, I2C_DMA_ON);
```

### i2c\_dma\_last\_transfer\_config

The description of i2c\_dma\_last\_transfer\_config is shown as below:

**Table 3-286. Function i2c\_dma\_last\_transfer\_config**

<b>Function name</b>	i2c_dma_last_transfer_config
<b>Function prototype</b>	void i2c_dma_last_transfer_config(uint32_t i2c_periph, uint32_t dmalast);
<b>Function descriptions</b>	configure whether next DMA EOT is DMA last transfer or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
dmalast	next DMA EOT is the last transfer or not
I2C_DMALST_ON	next DMA EOT is the last transfer
I2C_DMALST_OFF	next DMA EOT is not the last transfer
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* next DMA EOT is the last transfer */

i2c_dma_last_transfer_config (I2C0, I2C_DMALST_ON);
```

### i2c\_stretch\_scl\_low\_config

The description of i2c\_stretch\_scl\_low\_config is shown as below:

**Table 3-287. Function i2c\_stretch\_scl\_low\_config**

<b>Function name</b>	i2c_stretch_scl_low_config
<b>Function prototype</b>	void i2c_stretch_scl_low_config(uint32_t i2c_periph, uint32_t stretchpara);
<b>Function descriptions</b>	whether to stretch SCL low when data is not ready in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
stretchpara	SCL stretching enable or disable
I2C_SCLSTRETCH_ENABLE	SCL stretching is enabled
I2C_SCLSTRETCH_DISABLE	SCL stretching is disabled
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* stretch SCL low when data is not ready in slave mode */

i2c_stretch_scl_low_config (I2C0, I2C_SCLSTRETCH_ENABLE);
```

### i2c\_slave\_response\_to\_gcall\_config

The description of i2c\_slave\_response\_to\_gcall\_config is shown as below:

**Table 3-288. Function i2c\_slave\_response\_to\_gcall\_config**

<b>Function name</b>	i2c_slave_response_to_gcall_config
<b>Function prototype</b>	void i2c_slave_response_to_gcall_config(uint32_t i2c_periph, uint32_t gcallpara);

<b>Function descriptions</b>	whether or not to response to a general call
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<b>I2Cx</b>	(x=0,1)
<b>Input parameter{in}</b>	
<b>gcallpara</b>	response to a general call or not
<b>I2C_GCEN_ENABLE</b>	slave will response to a general call
<b>I2C_GCEN_DISABLE</b>	slave will not response to a general call
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 will response to a general call */

i2c_slave_response_to_gcall_config (I2C0, I2C_GCEN_ENABLE);
```

### i2c\_software\_reset\_config

The description of i2c\_software\_reset\_config is shown as below:

**Table 3-289. Function i2c\_software\_reset\_config**

<b>Function name</b>	i2c_software_reset_config
<b>Function prototype</b>	void i2c_software_reset_config(uint32_t i2c_periph, uint32_t sreset);
<b>Function descriptions</b>	software reset I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<b>I2Cx</b>	(x=0,1)
<b>Input parameter{in}</b>	
<b>sreset</b>	reset or not
<b>I2C_SRESET_SET</b>	I2C is under reset
<b>I2C_SRESET_RESET</b>	I2C is not under reset
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* software reset I2C0 */
```

i2c\_software\_reset\_config (I2C0, I2C\_SRESET\_SET);

## i2c\_pec\_enable

The description of i2c\_pec\_enable is shown as below:

**Table 3-290. Function i2c\_pec\_enable**

<b>Function name</b>	i2c_pec_enable
<b>Function prototype</b>	void i2c_pec_enable(uint32_t i2c_periph, uint32_t pecstate);
<b>Function descriptions</b>	I2C PEC calculation on or off
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>pecstate</b>	on or off
<i>I2C_PEC_ENABLE</i>	PEC calculation on
<i>I2C_PEC_DISABLE</i>	PEC calculation off
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* Enable I2C PEC calculation */
i2c_pec_enable (I2C0, I2C_PEC_ENABLE);
```

## i2c\_pec\_transfer\_enable

The description of i2c\_pec\_transfer\_enable is shown as below:

**Table 3-291. Function i2c\_pec\_transfer\_enable**

<b>Function name</b>	i2c_pec_transfer_enable
<b>Function prototype</b>	void i2c_pec_transfer_enable(uint32_t i2c_periph, uint32_t pecpara);
<b>Function descriptions</b>	I2C whether to transfer PEC value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>pecpara</b>	Transfer PEC or not
<i>I2C_PECTRANS_ENA</i>	transfer PEC

<i>BLE</i>	
<i>I2C_PECTRANS_DISA</i> <i>BLE</i>	not transfer PEC
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 transfer PEC */

i2c_pec_transfer_enable (I2C0, I2C_PECTRANS_ENABLE);
```

### i2c\_pec\_value\_get

The description of i2c\_pec\_value\_get is shown as below:

**Table 3-292. Function i2c\_pec\_value\_get**

<b>Function name</b>	i2c_pec_value_get
<b>Function prototype</b>	uint8_t i2c_pec_value_get(uint32_t i2c_periph);
<b>Function descriptions</b>	get packet error checking value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	PEC value

Example:

```
/* I2C0 get packet error checking value */

uint8_t pec_value;

pec_value = i2c_pec_value_get (I2C0);
```

### i2c\_smbus\_issue\_alert

The description of i2c\_smbus\_issue\_alert is shown as below:

**Table 3-293. Function i2c\_smbus\_issue\_alert**

<b>Function name</b>	i2c_smbus_issue_alert
<b>Function prototype</b>	void i2c_smbus_issue_alert(uint32_t i2c_periph, uint32_t smbuspara);
<b>Function descriptions</b>	I2C issue alert through SMBA pin

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<i>smbuspara</i>	issue alert through SMBA pin or not
<i>I2C_SALTSEND_ENAB LE</i>	issue alert through SMBA pin
<i>I2C_SALTSEND_DISA BLE</i>	not issue alert through SMBA pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 issue alert through SMBA pin enable*/
i2c_smbus_issue_alert (I2C0, I2C_SALTSEND_ENABLE);
```

### **i2c\_smbus\_arp\_enable**

The description of i2c\_smbus\_arp\_enable is shown as below:

**Table 3-294. Function i2c\_smbus\_arp\_enable**

<b>Function name</b>	i2c_smbus_arp_enable
<b>Function prototype</b>	void i2c_smbus_arp_enable(uint32_t i2c_periph, uint32_t arpstate);
<b>Function descriptions</b>	enable or disable I2C ARP protocol in SMBus switch
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<i>arpstate</i>	ARP protocol in SMBus switch
<i>I2C_ARP_ENABLE</i>	enable ARP
<i>I2C_ARP_DISABLE</i>	disable ARP
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

```
/* enable I2C0 ARP protocol in SMBus switch */
```

```
i2c_smbus_arp_enable (I2C0, I2C_ARP_ENABLE);
```

## i2c\_flag\_get

The description of i2c\_flag\_get is shown as below:

**Table 3-295. Function i2c\_flag\_get**

<b>Function name</b>	i2c_flag_get
<b>Function prototype</b>	FlagStatus i2c_flag_get(uint32_t i2c_periph, i2c_flag_enum flag)
<b>Function descriptions</b>	check I2C flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>I2C_FLAG_SBSEND</i>	start condition send out
<i>I2C_FLAG_ADDSEND</i>	address is sent in master mode or received and matches in slave mode
<i>I2C_FLAG_BTC</i>	byte transmission finishes
<i>I2C_FLAG_ADD10SEN_D</i>	header of 10-bit address is sent in master mode
<i>I2C_FLAG_STPDET</i>	stop condition detected in slave mode
<i>I2C_FLAG_RBNE</i>	I2C_DATA is not empty during receiving
<i>I2C_FLAG_TBE</i>	I2C_DATA is empty during transmitting
<i>I2C_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus
<i>I2C_FLAG_LOSTARB</i>	arbitration lost in master mode
<i>I2C_FLAG_AERR</i>	acknowledge error
<i>I2C_FLAG_OUERR</i>	overrun or underrun situation occurs in slave mode
<i>I2C_FLAG_PECERR</i>	PEC error when receiving data
<i>I2C_FLAG_SMBTO</i>	timeout signal in SMBus mode
<i>I2C_FLAG_SMBALT</i>	SMBus alert status
<i>I2C_FLAG_MASTER</i>	a flag indicating whether I2C block is in master or slave mode
<i>I2C_FLAG_I2CBSY</i>	busy flag
<i>I2C_FLAG_TR</i>	whether the I2C is a transmitter or a receiver
<i>I2C_FLAG_RXGC</i>	general call address (00h) received
<i>I2C_FLAG_DEFSMB</i>	default address of SMBus device
<i>I2C_FLAG_HSTSMB</i>	SMBus host header detected in slave mode
<i>I2C_FLAG_DUMOD</i>	dual flag in slave mode indicating which address is matched in dual-address mode
<b>Output parameter{out}</b>	

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

Example:

```
/* check whether start condition send out */

FlagStatus flag_state = RESET;

flag_state = i2c_flag_get (I2C0, I2C_FLAG_SBSEND);
```

### i2c\_flag\_clear

The description of i2c\_flag\_clear is shown as below:

**Table 3-296. Function i2c\_flag\_clear**

<b>Function name</b>	i2c_flag_clear
<b>Function prototype</b>	void i2c_flag_clear(uint32_t i2c_periph, i2c_flag_enum flag)
<b>Function descriptions</b>	clear I2C flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
flag	flag type
I2C_FLAG_SMBALT	SMBus Alert status
I2C_FLAG_SMBTO	timeout signal in SMBus mode
I2C_FLAG_PECERR	PEC error when receiving data
I2C_FLAG_OUERR	over-run or under-run situation occurs in slave mode
I2C_FLAG_AERR	acknowledge error
I2C_FLAG_LOSTARB	arbitration lost in master mode
I2C_FLAG_BERR	a bus error
I2C_FLAG_ADDSEND	cleared by reading I2C_STAT0 and reading I2C_STAT1
<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 3-297. Function i2c\_interrupt\_enable**

<b>Function name</b>	i2c_interrupt_enable
<b>Function prototype</b>	void i2c_interrupt_enable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);
<b>Function descriptions</b>	enable I2C interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
inttype	interrupt type
I2C_INT_ERR	error interrupt enable
I2C_INT_EV	event interrupt enable
I2C_INT_BUF	buffer interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 event interrupt */
i2c_interrupt_enable (I2C0, I2C_INT_EV);
```

### i2c\_interrupt\_disable

The description of i2c\_interrupt\_disable is shown as below:

**Table 3-298. Function i2c\_interrupt\_disable**

<b>Function name</b>	i2c_interrupt_disable
<b>Function prototype</b>	void i2c_interrupt_disable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);
<b>Function descriptions</b>	disable I2C interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
inttype	interrupt type
I2C_INT_ERR	error interrupt disable

<i>I2C_INT_EV</i>	event interrupt disable
<i>I2C_INT_BUF</i>	buffer interrupt disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C0 event interrupt */

i2c_interrupt_disable (I2C0, I2C_INT_EV);
```

### i2c\_interrupt\_flag\_get

The description of i2c\_interrupt\_flag\_get is shown as below:

**Table 3-299. Function i2c\_interrupt\_flag\_get**

<b>Function name</b>	i2c_interrupt_flag_get
<b>Function prototype</b>	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag)
<b>Function descriptions</b>	check I2C interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<i>int_flag</i>	interrupt flag
<i>I2C_INT_FLAG_SBSENDD</i>	start condition sent out in master mode interrupt flag
<i>I2C_INT_FLAG_ADDSENDD</i>	address is sent in master mode or received and matches in slave mode interrupt flag
<i>I2C_INT_FLAG_BTCSEND</i>	byte transmission finishes
<i>I2C_INT_FLAG_ADD10SEND</i>	header of 10-bit address is sent in master mode interrupt flag
<i>I2C_INT_FLAG_STPDSEND</i>	stop condition detected in slave mode interrupt flag
<i>I2C_INT_FLAG_RBNE</i>	I2C_DATA is not Empty during receiving interrupt flag
<i>I2C_INT_FLAG_TBE</i>	I2C_DATA is empty during transmitting interrupt flag
<i>I2C_INT_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag
<i>I2C_INT_FLAG_LOSTARB</i>	arbitration lost in master mode interrupt flag
<i>I2C_INT_FLAG_AERR</i>	acknowledge error interrupt flag

<i>I2C_INT_FLAG_OUER</i> <i>R</i>	over-run or under-run situation occurs in slave mode interrupt flag
<i>I2C_INT_FLAG_PECERR</i> <i>RR</i>	PEC error when receiving data interrupt flag
<i>I2C_INT_FLAG_SMBTO</i> <i>O</i>	timeout signal in SMBus mode interrupt flag
<i>I2C_INT_FLAG_SMBA</i> <i>LT</i>	SMBus Alert status interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* check the byte transmission finishes interrupt flag is set or not*/
FlagStatus flag_state = RESET;
flag_state = i2c_interrupt_flag_get (I2C0, I2C_INT_FLAG_BTC);
```

### i2c\_interrupt\_flag\_clear

The description of i2c\_interrupt\_flag\_clear is shown as below:

**Table 3-300. 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
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt flag
<i>I2C_INT_FLAG_ADDSENDD</i>	address is sent in master mode or received and matches in slave mode interrupt flag
<i>I2C_INT_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag
<i>I2C_INT_FLAG_LOSTARB</i>	arbitration lost in master mode interrupt flag
<i>I2C_INT_FLAG_AERR</i>	acknowledge error interrupt flag
<i>I2C_INT_FLAG_OUERR</i> <i>R</i>	over-run or under-run situation occurs in slave mode interrupt flag

<i>I2C_INT_FLAG_PECE RR</i>	PEC error when receiving data interrupt flag
<i>I2C_INT_FLAG_SMBTO</i>	timeout signal in SMBus mode interrupt flag
<i>I2C_INT_FLAG_SMBA LT</i>	SMBus Alert status interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the acknowledge error interrupt flag */

i2c_interrupt_flag_clear (I2C0, I2C_INT_FLAG_AERR);
```

## 3.15. MISC

MISC is a software package that provide the interfaces for NVIC and SysTick. The NVIC and SysTick registers are listed in chapter [3.15.1](#), the MISC firmware functions are introduced in chapter [3.15.2](#).

### 3.15.1. Descriptions of Peripheral registers

**Table 3-301. NVIC Registers**

Registers	Descriptions
ISER <sup>(1)</sup>	Interrupt Set Enable Register
ICER <sup>(1)</sup>	Interrupt Clear Enable Register
ISPR <sup>(1)</sup>	Interrupt Set Pending Register
ICPR <sup>(1)</sup>	Interrupt Clear Pending Register
IABR <sup>(1)</sup>	Interrupt Active bit Register
IP <sup>(1)</sup>	Interrupt Priority Register
STIR <sup>(1)</sup>	Software Trigger Interrupt Register
CPUID <sup>(2)</sup>	CPUID Base Register
ICSR <sup>(2)</sup>	Interrupt Control and State Register
VTOR <sup>(2)</sup>	Vector Table Offset Register
AIRCR <sup>(2)</sup>	Application Interrupt and Reset Control Register
SCR <sup>(2)</sup>	System Control Register
CCR <sup>(2)</sup>	Configuration Control Register
SHP <sup>(2)</sup>	System Handlers Priority Registers
SHCSR <sup>(2)</sup>	System Handler Control and State Register

Registers	Descriptions
CFSR <sup>(2)</sup>	Configurable Fault Status Register
HFSR <sup>(2)</sup>	HardFault Status Register
DFSR <sup>(2)</sup>	Debug Fault Status Register
MMFAR <sup>(2)</sup>	MemManage Fault Address Register
BFAR <sup>(2)</sup>	BusFault Address Register
AFSR <sup>(2)</sup>	Auxiliary Fault Status Register
PFR <sup>(2)</sup>	Processor Feature Register
DFR <sup>(2)</sup>	Debug Feature Register
ADR <sup>(2)</sup>	Auxiliary Feature Register
MMFR <sup>(2)</sup>	Memory Model Feature Register
ISAR <sup>(2)</sup>	Instruction Set Attributes Register
CPACR <sup>(2)</sup>	Coprocessor Access Control Register

1. refer to the structure NVIC\_Type, is defined in the core\_cm4.h file

2. refer to the structure SCB\_Type, is defined in the core\_cm4.h file

**Table 3-302. SysTick Registers**

Registers	Descriptions
CTRL <sup>(1)</sup>	SysTick Control and Status Register
LOAD <sup>(1)</sup>	SysTick Reload Value Register
VAL <sup>(1)</sup>	SysTick Current Value Register
CALIB <sup>(1)</sup>	SysTick Calibration Register

1. refer to the structure SysTick\_Type, is defined in the core\_cm4.h file

### 3.15.2. Descriptions of Peripheral functions

#### Enum IRQn\_Type

**Table 3-303. Enum IRQn\_Type**

Member name	Function description
WWDGT_IRQHandler	window watchDog timer interrupt
LVD_IRQHandler	LVD through EXTI line detect interrupt
RTC_IRQHandler	RTC through EXTI line interrupt
FMC_IRQHandler	FMC interrupt
RCU_CTC_IRQHandler	RCU and CTC interrupt
EXTI0_1_IRQHandler	EXTI line 0 and 1 interrupts
EXTI2_3_IRQHandler	EXTI line 2 and 3 interrupts
EXTI4_15_IRQHandler	EXTI line 4 to 15 interrupts
TSI_IRQHandler	TSI Interrupt
DMA_Channel0_IRQHandler	DMA channel 0 interrupt
DMA_Channel1_2_IRQHandler	DMA channel 1 and channel 2 interrupts
DMA_Channel3_4_IRQHandler	DMA channel 3 and channel 4 interrupts
ADC_CMP_IRQHandler	ADC, CMP0 and CMP1 interrupts

Member name	Function description
TIMER0_BRK_UP_TRG_COM_IRQn	TIMER0 break, update, trigger and commutation interrupts
TIMER0_Channel_IRQn	TIMER0 channel capture compare interrupts
TIMER1_IRQn	TIMER1 interrupt
TIMER2_IRQn	TIMER2 interrupt
TIMER5_DAC_IRQn	TIMER5 and DAC interrupts
TIMER13_IRQn	TIMER13 interrupt
TIMER14_IRQn	TIMER14 interrupt
TIMER15_IRQn	TIMER15 interrupt
TIMER16_IRQn	TIMER16 interrupt
I2C0_EV_IRQn	I2C0 event interrupt
I2C1_EV_IRQn	I2C1 event interrupt
SPI0_IRQn	SPI0 interrupt
SPI1_IRQn	SPI1 interrupt
USART0_IRQn	USART0 interrupt
USART1_IRQn	USART1 interrupt
CEC_IRQn	CEC interrupt
I2C0_ER_IRQn	I2C0 error interrupt
I2C1_ER_IRQn	I2C1 error interrupt
DMA_Channel5_6_IRQn	DMA channel 5 and channel 6 interrupts
USBFS_WKUP_IRQn	USBFS wakeup interrupt
USBFS_IRQn	USBFS global interrupt

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

**Table 3-304. MISC firmware function**

Function name	Function description
nvic_priority_group_set	set the priority group
nvic_irq_enable	enable NVIC interrupt request
nvic_irq_disable	disable NVIC interrupt request
nvic_vector_table_set	set the NVIC vector table address
system_lowpower_set	set the state of the low power mode
system_lowpower_reset	reset the state of the low power mode
systick_clksource_set	set the systick clock source

### **nvic\_priority\_group\_set**

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

**Table 3-305. Function nvic\_priority\_group\_set**

Function name	nvic_priority_group_set
Function prototype	void nvic_priority_group_set(uint32_t nvic_prigrp);
Function descriptions	configure bits length of the priority group
Precondition	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>nvic_prigroup</b>	priority group
<b>NVIC_PRIGROUP_PR_E0_SUB4</b>	0 bits for pre-emption priority 4 bits for subpriority
<b>NVIC_PRIGROUP_PR_E1_SUB3</b>	1 bits for pre-emption priority 3 bits for subpriority
<b>NVIC_PRIGROUP_PR_E2_SUB2</b>	2 bits for pre-emption priority 2 bits for subpriority
<b>NVIC_PRIGROUP_PR_E3_SUB1</b>	3 bits for pre-emption priority 1 bits for subpriority
<b>NVIC_PRIGROUP_PR_E4_SUB0</b>	4 bits for pre-emption priority 0 bits for subpriority
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* priority group configuration , 0 bits for pre-emption priority 4 bits for subpriority */
nvic_priority_group_set(NVIC_PRIGROUP_PRE0_SUB4);
```

### **nvic\_irq\_enable**

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

**Table 3-306. Function nvic\_irq\_enable**

<b>Function name</b>	nvic_irq_enable
<b>Function prototype</b>	void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority);
<b>Function descriptions</b>	enable NVIC request, configure the priority of interrupt
<b>Precondition</b>	-
<b>The called functions</b>	NVIC_SetPriority、NVIC_EnableIRQ
<b>Input parameter{in}</b>	
<b>nvic_irq</b>	NVIC interrupt, refer to enum <a href="#">Enum IRQn_Type</a>
<b>Input parameter{in}</b>	
<b>nvic_irq_pre_priority</b>	the pre-emption priority needed to set (0~3)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-307. Function nvic\_irq\_disable**

<b>Function name</b>	nvic_irq_disable	
<b>Function prototype</b>	void nvic_irq_disable(uint8_t nvic_irq);	
<b>Function descriptions</b>	disable NVIC request	
<b>Precondition</b>	-	
<b>The called functions</b>	-	
<b>Input parameter{in}</b>		
<b>nvic_irq</b>	NVIC interrupt, refer to enum <a href="#">Enum IRQn_Type</a>	
<b>Output parameter{out}</b>		
-	-	
<b>Return value</b>		
-	-	

Example:

```
/* disable window watchDog timer interrupt */
nvic_irq_disable(WWDGT_IRQn);
```

### **nvic\_vector\_table\_set**

The description of nvic\_vector\_table\_set is shown as below:

**Table 3-308. Function nvic\_vector\_table\_set**

<b>Function name</b>	nvic_vector_table_set	
<b>Function prototype</b>	void nvic_vector_table_set(uint32_t nvic_vict_tab, uint32_t offset);	
<b>Function descriptions</b>	set the NVIC vector table address	
<b>Precondition</b>	-	
<b>The called functions</b>	-	
<b>Input parameter{in}</b>		
<b>nvic_vict_tab</b>	the RAM or FLASH base address	
<b>NVIC_VECTTAB_RAM</b>	RAM base address	
<b>NVIC_VECTTAB_FLASH</b>	Flash base address	
<b>Input parameter{in}</b>		
<b>Offset</b>	Vector Table offset (vector table start address= base address+offset)	
<b>Output parameter{out}</b>		
-	-	
<b>Return value</b>		
-	-	

Example:

```
/* set vector table address = NVIC_VECTTAB_FLASH +0x200 */
nvic_vector_table_set(NVIC_VECTTAB_FLASH,0x200);
```

### **system\_lowpower\_set**

The description of system\_lowpower\_set is shown as below:

**Table 3-309. Function system\_lowpower\_set**

<b>Function name</b>	system_lowpower_set
<b>Function prototype</b>	void system_lowpower_set(uint8_t lowpower_mode);
<b>Function descriptions</b>	the state of the low power mode management
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lowpower_mode</b>	the low power mode state
SCB_LPM_SLEEP_EXIT_ISR	if chose this para, the system always enter low power mode by exiting from ISR
SCB_LPM_DEEPSLEEP_P	if chose this para, the system will enter the DEEPSLEEP mode
SCB_LPM_WAKE_BY_ALL_INT	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 3-310. Function system\_lowpower\_reset**

<b>Function name</b>	system_lowpower_reset
<b>Function prototype</b>	void system_lowpower_reset(uint8_t lowpower_mode);
<b>Function descriptions</b>	the state of the low power mode management
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lowpower_mode</b>	the low power mode state
SCB_LPM_SLEEP_EXIT_ISR	if chose this para, the system will exit low power mode by exiting from ISR

<i>T_ISR</i>	
<i>SCB_LPM_DEEPSLEEP_P</i>	if chose this para, the system will enter the SLEEP mode
<i>SCB_LPM_WAKE_BY_ALL_INT</i>	if chose this para, the lowpower mode only can be woke up by the enable interrupts
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* the system will exit low power mode by exiting from ISR */
system_lowpower_reset (SCB_LPM_SLEEP_EXIT_ISR);
```

### systick\_clksource\_set

The description of systick\_clksource\_set is shown as below:

**Table 3-311. 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
<i>SYSTICK_CLKSOURCE_E_HCLK</i>	systick clock source is from HCLK
<i>SYSTICK_CLKSOURCE_E_HCLK_DIV8</i>	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);
```

## 3.16. PMU

According to the Power management unit (PMU), provides three types of power saving modes, including Sleep, Deep-sleep and Standby mode. The PMU registers are listed in chapter

[3.16.1](#), the PMU firmware functions are introduced in chapter [3.16.2](#).

### 3.16.1. Descriptions of Peripheral registers

PMU registers are listed in the table shown as below:

**Table 3-312. PMU Registers**

Registers	Descriptions
PMU_CTL	PMU control register
PMU_CS	PMU control and status register

### 3.16.2. Descriptions of Peripheral functions

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

**Table 3-313. PMU firmware function**

Function name	Function description
pmu_deinit	reset PMU registers
pmu_lvd_select	select low voltage detector threshold
pmu_ldo_output_select	select LDO output voltage
pmu_lvd_disable	disable PMU lvd
pmu_lowdriver_mode_enable	enable low-driver mode in deep-sleep mode
pmu_lowdriver_mode_disable	disable low-driver mode in deep-sleep mode
pmu_highdriver_mode_enable	enable high-driver mode
pmu_highdriver_mode_disable	disable high-driver mode
pmu_highdriver_switch_select	switch high-driver mode
pmu_lowpower_driver_config	in deep-sleep mode, low-driver mode when use low power LDO
pmu_normalpower_driver_config	in deep-sleep mode, low-driver mode when use normal power LDO
pmu_to_sleepmode	PMU work in sleep mode
pmu_to_deepsleepmode	PMU work at deepsleep mode
pmu_to_standbymode	pmu work at standby mode
pmu_wakeup_pin_enable	enable PMU wakeup pin
pmu_wakeup_pin_disable	disable PMU wakeup pin
pmu_backup_write_enable	enable backup domain write
pmu_backup_write_disable	disable backup domain write
pmu_flag_clear	clear flag bit
pmu_flag_get	get flag state

#### pmu\_deinit

The description of pmu\_deinit is shown as below:

**Table 3-314. Function pmu\_deinit**

<b>Function name</b>	pmu_deinit
<b>Function prototype</b>	void pmu_deinit(void);
<b>Function descriptions</b>	reset PMU register
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset PMU */
pmu_deinit();
```

### pmu\_lvd\_select

The description of pmu\_lvd\_select is shown as below:

**Table 3-315. Function pmu\_lvd\_select**

<b>Function name</b>	pmu_lvd_select
<b>Function prototype</b>	void pmu_lvd_select(uint32_t lvdt_n);
<b>Function descriptions</b>	select low voltage detector threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lvdt_n</b>	voltage threshold value
<i>PMU_LVDT_0</i>	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>	voltage threshold is 3.1V
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

```
/* select low voltage detector threshold as 3.1V */
```

```
pmu_lvd_select (PMU_LVDT_7);
```

### **pmu\_ldo\_output\_select**

The description of pmu\_ldo\_output\_select is shown as below:

**Table 3-316. 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
<i>PMU_LDOVS_MID</i>	LDO output voltage mid mode
<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\_lvd\_disable**

The description of pmu\_lvd\_disable is shown as below:

**Table 3-317. 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\_lowdriver\_mode\_enable**

The description of pmu\_lowdriver\_mode\_enable is shown as below:

**Table 3-318. Function pmu\_lowdriver\_mode\_enable**

<b>Function name</b>	pmu_lowdriver_mode_enable
<b>Function prototype</b>	void pmu_lowdriver_mode_enable(void);
<b>Function descriptions</b>	enable low-driver mode in deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable low-driver mode in deep-sleep mode */
```

```
pmu_lowdriver_mode_enable();
```

### **pmu\_lowdriver\_mode\_disable**

The description of pmu\_lowdriver\_mode\_disable is shown as below:

**Table 3-319. Function pmu\_lowdriver\_mode\_disable**

<b>Function name</b>	pmu_lowdriver_mode_disable
<b>Function prototype</b>	void pmu_lowdriver_mode_disable(void);
<b>Function descriptions</b>	disable low-driver mode in deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

```
/* disable low-driver mode in deep-sleep mode */
```

```
pmu_lowdriver_mode_disable();
```

### **pmu\_highdriver\_mode\_enable**

The description of pmu\_highdriver\_mode\_enable is shown as below:

**Table 3-320. Function pmu\_highdriver\_mode\_enable**

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

Example:

```
/* enable high-driver mode */
```

```
pmu_highdriver_mode_enable();
```

### **pmu\_highdriver\_mode\_disable**

The description of pmu\_highdriver\_mode\_disable is shown as below:

**Table 3-321. Function pmu\_highdriver\_mode\_disable**

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

Example:

```
/* disable high-driver mode */
```

pmu\_highdriver\_mode\_disable ( );

### **pmu\_highdriver\_switch\_select**

The description of pmu\_highdriver\_switch\_select is shown as below:

**Table 3-322. Function pmu\_highdriver\_switch\_select**

<b>Function name</b>	pmu_highdriver_switch_select
<b>Function prototype</b>	void pmu_highdriver_switch_select(uint32_t highdr_switch);
<b>Function descriptions</b>	switch high-driver mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>highdr_switch</b>	enable or disable high-driver mode switch
<i>PMU_HIGHDR_SWITC_H_NONE</i>	disable high-driver mode switch
<i>PMU_HIGHDR_SWITC_H_EN</i>	enable high-driver mode switch
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable high-driver mode switch */
pmu_highdriver_switch_select (PMU_HIGHDR_SWITCH_EN);
```

### **pmu\_lowpower\_driver\_config**

The description of pmu\_lowpower\_driver\_config is shown as below:

**Table 3-323. Function pmu\_lowpower\_driver\_config**

<b>Function name</b>	pmu_lowpower_driver_config
<b>Function prototype</b>	void pmu_lowpower_driver_config (uint32_t mode);
<b>Function descriptions</b>	low-driver mode when use low power LDO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mode</b>	Low power mode of LDO-
<i>PMU_NORMALDR_LO_WPWR</i>	normal driver when use low power LDO
<i>PMU_LOWDR_LOWP_WR</i>	low-driver mode enabled when LDEN is 11 and use low power LDO
<b>Output parameter{out}</b>	

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

Example:

```
/* normal driver when use low power LDO */
pmu_lowpower_driver_config (PMU_NORMALDR_LOWPWR);
```

### **pmu\_normalpower\_driver\_config**

The description of pmu\_normalpower\_driver\_config is shown as below:

**Table 3-324. Function pmu\_normalpower\_driver\_config**

<b>Function name</b>	pmu_normalpower_driver_config
<b>Function prototype</b>	void pmu_normalpower_driver_config (uint32_t mode);
<b>Function descriptions</b>	low-driver mode when use normal power LDO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mode</b>	Driver mode when use normal power LDO-
<b>PMU_NORMALDR_NORMALPWR</b>	normal driver when use normal power LDO
<b>PMU_LOWDR_NORMALPWR</b>	low-driver mode enabled when LDEN is 11 and use normal power LDO
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* normal driver when use normal power LDO */
pmu_normalpower_driver_config (PMU_NORMALDR_NORMALPWR);
```

### **pmu\_to\_sleepmode**

The description of pmu\_to\_sleepmode is shown as below:

**Table 3-325. Function pmu\_to\_sleepmode**

<b>Function name</b>	pmu_to_sleepmode
<b>Function prototype</b>	void pmu_to_sleepmode(uint8_t sleepmodecmd);
<b>Function descriptions</b>	PMU work at sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>sleepmodecmd</b>	command to enter sleep mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work at sleep mode */

pmu_to_sleepmode (WFI_CMD);
```

### **pmu\_to\_deepsleepmode**

The description of pmu\_to\_deepsleepmode is shown as below:

**Table 3-326. Function pmu\_to\_deepsleepmode**

<b>Function name</b>	pmu_to_deepsleepmode
<b>Function prototype</b>	void pmu_to_deepsleepmode(uint32_t ldo,uint8_t deepsleepmodecmd);
<b>Function descriptions</b>	PMU work at deepsleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ldo</b>	ldo work mode
<i>PMU_LDO_NORMAL</i>	LDO operates normally when pmu enter deepsleep mode
<i>PMU_LDO_LOWPOWER</i>	LDO work at low power mode when pmu enter deepsleep mode
<b>Input parameter{in}</b>	
<b>deepsleepmodecmd</b>	command to enter deepsleep mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work at deepsleep mode */

pmu_to_deepsleepmode (PMU_LDO_NORMAL, WFI_CMD);
```

### **pmu\_to\_standbymode**

The description of pmu\_to\_standbymode is shown as below:

**Table 3-327. Function pmu\_to\_standbymode**

<b>Function name</b>	pmu_to_standbymode
<b>Function prototype</b>	void pmu_to_standbymode(uint8_t standbymodecmd);
<b>Function descriptions</b>	pmu work at standby mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>standbymodecmd</b>	command to enter standby mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work at standby mode */

pmu_to_standby (WFI_CMD);
```

### **pmu\_wakeup\_pin\_enable**

The description of pmu\_wakeup\_pin\_enable is shown as below:

**Table 3-328. 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_PIN4</i>	WKUP Pin 4 (PC5)
<i>PMU_WAKEUP_PIN5</i>	WKUP Pin 5 (PB5)
<i>PMU_WAKEUP_PIN6</i>	WKUP Pin 6 (PB15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable wakeup pin6 */
```

---

```
pmu_wakeup_pin_enable (PMU_WAKEUP_PIN6);
```

### **pmu\_wakeup\_pin\_disable**

The description of pmu\_wakeup\_pin\_disable is shown as below:

**Table 3-329. 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>	
wakeup_pin	Wakeup pin
PMU_WAKEUP_PIN0	WKUP Pin 0 (PA0)
PMU_WAKEUP_PIN1	WKUP Pin 1 (PC13)
PMU_WAKEUP_PIN4	WKUP Pin 4 (PC5)
PMU_WAKEUP_PIN5	WKUP Pin 5 (PB5)
PMU_WAKEUP_PIN6	WKUP Pin 6 (PB15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable wakeup pin6 */
pmu_wakeup_pin_disable (PMU_WAKEUP_PIN6);
```

### **pmu\_backup\_write\_enable**

The description of pmu\_backup\_write\_enable is shown as below:

**Table 3-330. 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 3-331. Function pmu\_backup\_write\_disable**

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

Example:

```
/* disable backup domain write */
pmu_backup_write_disable();
```

### **pmu\_flag\_clear**

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

**Table 3-332. Function pmu\_flag\_clear**

<b>Function name</b>	pmu_flag_clear
<b>Function prototype</b>	void pmu_flag_clear(uint32_t flag_clear);
<b>Function descriptions</b>	clear flag bit
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag_clear</b>	flag
<b>PMU_FLAG_RESET_WAKEUP</b>	reset wakeup flag
<b>PMU_FLAG_RESET_STANDBY</b>	reset standby flag
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* clear flag bit */
pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

### pmu\_flag\_get

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

**Table 3-333. Function pmu\_flag\_get**

<b>Function name</b>	pmu_flag_get
<b>Function prototype</b>	FlagStatus pmu_flag_get(uint32_t flag);
<b>Function descriptions</b>	get flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flag
<i>PMU_FLAG_WAKEUP</i>	wakeup flag
<i>PMU_FLAG_STANDBY</i>	standby flag
<i>PMU_FLAG_LVD</i>	lvd flag
<i>PMU_FLAG_LDOVSR</i>	LDO voltage select ready flag
<i>PMU_FLAG_HDR</i>	high-driver ready flag
<i>PMU_FLAG_HDSR</i>	high-driver switch ready flag
<i>PMU_FLAG_LDR</i>	low-driver mode ready flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get flag state */

FlagStatus status;

status = pmu_flag_get (PMU_FLAG_WAKEUP);
```

## 3.17. RCU

RCU is the reset and clock unit. Reset Control includes the control of three kinds of reset: power reset, system reset and backup domain reset. The Clock Control unit provides a range of frequencies and clock functions. The RCU registers are listed in chapter [3.17.1](#), the RCU firmware functions are introduced in chapter [3.17.2](#).

### 3.17.1. Descriptions of Peripheral registers

**Table 3-334. RCU Registers**

Registers	Descriptions
RCU_CTL0	Control register 0
RCU_CFG0	Clock configuration register 0
RCU_INT	Clock interrupt register
RCU_APB2RST	APB2 reset register
RCU_APB1RST	APB1 reset register
RCU_AHBEN	AHB enable register
RCU_APB2EN	APB2 enable register
RCU_APB1EN	APB1 enable register
RCU_BDCTL	Backup domain control register
RCU_RSTSCK	Reset source/clock register
RCU_AHBRST	AHB reset register
RCU_CFG1	Clock configuration register 1
RCU_CFG2	Clock configuration register 2
RCU_CTL1	Control register 1
RCU_ADDCTL	Additional clock control register
RCU_ADDINT	Additional clock interrupt register
RCU_ADDAPB1EN	APB1 additional enable register
RCU_ADDAPB1RST	APB1 additional reset register
RCU_VKEY	Unlock voltage register
RCU_DSV	Deep-sleep mode voltage register

### 3.17.2. Descriptions of Peripheral functions

**Table 3-335. RCU firmware function**

Function name	Function description
rcu_deinit	deinitialize the RCU
rcu_periph_clock_enable	enable the peripherals clock
rcu_periph_clock_disable	disable the peripherals clock
rcu_periph_clock_sleep_enable	enable the peripherals clock when in sleep mode
rcu_periph_clock_sleep_disable	disable the peripherals clock when in sleep mode
rcu_periph_reset_enable	enable the peripherals reset
rcu_periph_reset_disable	disable the peripheral reset
rcu_bkp_reset_enable	enable the BKP domain reset
rcu_bkp_reset_disable	disable the BKP domain reset
rcu_system_clock_source_config	configure the system clock source
rcu_system_clock_source_get	get the system clock source
rcu_ahb_clock_config	configure the AHB clock prescaler selection
rcu_apb1_clock_config	configure the APB1 clock prescaler selection

Function name	Function description
rcu_apb2_clock_config	configure the APB2 clock prescaler selection
rcu_adc_clock_config	configure the ADC clock source and prescaler selection
rcu_usvfs_clock_config	configure the USBFS prescaler selection
rcu_ckout_config	configure the CK_OUT clock source and divider
rcu_pll_preselection_config	configure the PLL clock source preselection
rcu_pll_config	configure the main PLL clock
rcu_usart_clock_config	configure the usart clock
rcu_cec_clock_config	configure the CEC clock source selection
rcu_rtc_clock_config	configure the RTC clock source selection
rcu_ck48m_clock_config	configure the CK48M clock selection
rcu_hxtal_pdiv_config	configure the HXTAL divider used as input of PLL
rcu_lxtal_drive_capability_config	configure the LXTAL drive capability
rcu_flag_get	get the clock stabilization and periphral reset flags
rcu_all_reset_flag_clear	clear all the reset flag
rcu_interrupt_flag_get	get the clock stabilization interrupt and ckm flags
rcu_interrupt_flag_clear	clear the interrupt flags
rcu_interrupt_enable	enable the stabilization interrupt
rcu_interrupt_disable	disable the stabilization interrupt
rcu_osc_stab_wait	wait for oscillator stabilization flags is SET or oscillator startup is timeout
rcu_osc_on	turn on the oscillator
rcu_osc_off	turn off the oscillator
rcu_osc_bypass_mode_enable	enable the oscillator bypass mode
rcu_osc_bypass_mode_disable	disable the oscillator bypass mode
rcu_hxtal_clock_monitor_enable	enable the HXTAL clock monitor
rcu_hxtal_clock_monitor_disable	disable the HXTAL clock monitor
rcu_irc8m_adjust_value_set	set the IRC8M adjust value
rcu_irc28m_adjust_value_set	set the IRC28M adjust value
rcu_voltage_key_unlock	unlock Deep-sleep mode voltage register
rcu_deepsleep_voltage_set	set the deep-sleep mode voltage value
rcu_clock_freq_get	get the system clock, bus clock frequency

## reg\_idx

Table 3-336. Enum reg\_idx

enum name	Function description
IDX_AHBEN	index of AHB enable register
IDX_APB2EN	index of APB2 enable register
IDX_APB1EN	index of APB1 enable register
IDX_ADDAPB1EN	index of APB1 additional enable register
IDX_AHBRST	index of AHB reset register
IDX_APB2RST	index of APB2 reset register

enum name	Function description
IDX_APB1RST	index of APB1 reset register
IDX_ADDAPB1RST	index of APB1 additional reset register
IDX_CTL0	index of control register0
IDX_BDCTL	index of backup domain control register
IDX_CTL1	index of control register1
IDX_ADDCTL	index of additional clock control register
IDX_RSTSCK	index of reset source / clock register
IDX_INT	index of interrupt register
IDX_ADDINT	index of additional clock interrupt register
IDX_CFG0	index of configuration register0
IDX_CFG2	index of configuration register2

## rcu\_periph\_enum

Table 3-337. Enum rcu\_periph\_enum

enum name	Function description
RCU_DMA	DMA clock
RCU_CRC	CRC clock
RCU_GPIOA	GPIOA clock
RCU_GPIOB	GPIOB clock
RCU_GPIOC	GPIOC clock
RCU_GPIOD	GPIOD clock
RCU_GPIOF	GPIOF clock
RCU_TSI	TSI clock
RCU_CFGCMP	CFGCMP clock
RCU_ADC	ADC clock
RCU_TIMER0	TIMER0 clock
RCU_SPI0	SPI0 clock
RCU_USART0	USART0 clock
RCU_TIMER14	TIMER14 clock
RCU_TIMER15	TIMER15 clock
RCU_TIMER16	TIMER16 clock
RCU_TIMER1	TIMER1 clock
RCU_TIMER2	TIMER2 clock
RCU_TIMER13	TIMER13 clock
RCU_WWDGT	WWDT clock
RCU_SPI1	SPI1 clock
RCU_USART1	USART1 clock
RCU_I2C0	I2C0 clock
RCU_I2C1	I2C1 clock
RCU_PMU	PMU clock
RCU_DAC	DAC clock (only for GD32F350)

enum name	Function description
RCU_CEC	CEC clock (only for GD32F350)
RCU_TIMER5	TIMER5 clock (only for GD32F350)
RCU_USBFS	USBFS clock (only for GD32F350)
RCU_RTC	RTC clock
RCU_CTC	CTC clock

### rcu\_periph\_sleep\_enum

Table 3-338. Enum rcu\_periph\_sleep\_enum

enum name	Function description
RCU_SRAM_SLP	SRAM clock
RCU_FMC_SLP	FMC clock

### rcu\_periph\_reset\_enum

Table 3-339. Enum rcu\_periph\_reset\_enum

enum name	Function description
RCU_GPIOARST	GPIOA reset
RCU_GPIOBRST	GPIOB reset
RCU_GPIOCRST	GPIOC reset
RCU_GPIODRST	GPIOD reset
RCU_GPIOFRST	GPIOF reset
RCU_TSIRST	TSI reset
RCU_CFGCMRST	CFGCMP reset
RCU_ADCRST	ADC reset
RCU_TIMER0RST	TIMER0 reset
RCU_SPI0RST	SPI0 reset
RCU_USART0RST	USART0 reset
RCU_TIMER14RST	TIMER14 reset
RCU_TIMER15RST	TIMER15 reset
RCU_TIMER16RST	TIMER16 reset
RCU_TIMER1RST	TIMER1 reset
RCU_TIMER2RST	TIMER2 reset
RCU_TIMER13RST	TIMER13 reset
RCU_WWDGTRST	WWDGTRST reset
RCU_SPI1RST	SPI1 reset
RCU_USART1RST	USART1 reset
RCU_I2C0RST	I2C0 reset
RCU_I2C1RST	I2C1 reset
RCU_PMURST	PMU reset
RCU_DACRST	DAC reset (only for GD32F350)
RCU_CECRST	CEC reset (only for GD32F350)

enum name	Function description
RCU_TIMER5RST	TIMER5 reset (only for GD32F350)
RCU_USBFSRST	USBFS reset (only for GD32F350)
RCU_CTCRST	CTC reset

### rcu\_flag\_enum

Table 3-340. Enum rcu\_flag\_enum

enum name	Function description
RCU_FLAG_IRC40 KSTB	IRC40K stabilization flags
RCU_FLAG_LXTAL STB	LXTAL stabilization flags
RCU_FLAG_IRC8M STB	IRC8M stabilization flags
RCU_FLAG_HXTAL STB	HXTAL stabilization flags
RCU_FLAG_PLLST B	PLL stabilization flags
RCU_FLAG_IRC28 MSTB	IRC28M stabilization flags
RCU_FLAG_IRC48 MSTB	IRC48M stabilization flags
RCU_FLAG_V12RS T	V12 domain Power reset flags
RCU_FLAG_OBLR ST	Option byte loader reset flags
RCU_FLAG_EPRS T	External PIN reset flags
RCU_FLAG_PORR ST	Power reset flags
RCU_FLAG_SWRS T	Software reset flags
RCU_FLAG_FWDG TRST	Free Watchdog timer reset flags
RCU_FLAG_WWD GTRST	Window watchdog timer reset flags
RCU_FLAG_LPRST	Low-power reset flags

### rcu\_int\_flag\_enum

Table 3-341. Enum rcu\_int\_flag\_enum

enum name	Function description
RCU_INT_FLAG_IR	IRC40K stabilization interrupt flag

<b>enum name</b>	<b>Function description</b>
C40KSTB	
RCU_INT_FLAG_L XTALSTB	LXTAL stabilization interrupt flag
RCU_INT_FLAG_IR C8MSTB	IRC8M stabilization interrupt flag
RCU_INT_FLAG_H XTALSTB	HXTAL stabilization interrupt flag
RCU_INT_FLAG_P LLSTB	PLL stabilization interrupt flag
RCU_INT_FLAG_IR C28MSTB	IRC28M stabilization interrupt flag
RCU_INT_FLAG_C KM	CKM interrupt flag
RCU_INT_FLAG_IR C48MSTB	IRC48M stabilization interrupt flag

### **rcu\_int\_flag\_clear\_enum**

**Table 3-342. Enum rcu\_int\_flag\_clear\_enum**

<b>enum name</b>	<b>Function description</b>
RCU_INT_FLAG_IR C40KSTB_CLR	IRC40K stabilization interrupt flags clear
RCU_INT_FLAG_L XTALSTB_CLR	LXTAL stabilization interrupt flags clear
RCU_INT_FLAG_IR C8MSTB_CLR	IRC8M stabilization interrupt flags clear
RCU_INT_FLAG_H XTALSTB_CLR	HXTAL stabilization interrupt flags clear
RCU_INT_FLAG_P LLSTB_CLR	PLL stabilization interrupt flags clear
RCU_INT_FLAG_IR C28MSTB_CLR	IRC28M stabilization interrupt flags clear
RCU_INT_FLAG_C KM_CLR	CKM interrupt flags clear
RCU_INT_FLAG_IR C48MSTB_CLR	IRC48M stabilization interrupt flag clear

### **rcu\_int\_enum**

**Table 3-343. Enum rcu\_int\_enum**

<b>enum name</b>	<b>Function description</b>
RCU_INT_IRC40KS TB	IRC40K stabilization interrupt

enum name	Function description
RCU_INT_LXTALS_TB	LXTAL stabilization interrupt
RCU_INT_IRC8MS_TB	IRC8M stabilization interrupt
RCU_INT_HXTALS_TB	HXTAL stabilization interrupt
RCU_INT_PLLSTB	PLL stabilization interrupt
RCU_INT_IRC28M_STB	IRC28M stabilization interrupt
RCU_INT_IRC48M_STB	IRC48M stabilization interrupt

### rcu\_adc\_clock\_enum

Table 3-344. Enum rcu\_adc\_clock\_enum

enum name	Function description
RCU_ADCCK_IRC2_8M_DIV2	ADC clock source select IRC28M/2
RCU_ADCCK_IRC2_8M	ADC clock source select IRC28M
RCU_ADCCK_APB_2_DIV2	ADC clock source select APB2/2
RCU_ADCCK_AHB_DIV3	ADC clock source select AHB/3
RCU_ADCCK_APB_2_DIV4	ADC clock source select APB2/4
RCU_ADCCK_AHB_DIV5	ADC clock source select AHB/5
RCU_ADCCK_APB_2_DIV6	ADC clock source select APB2/6
RCU_ADCCK_AHB_DIV7	ADC clock source select AHB/7
RCU_ADCCK_APB_2_DIV8	ADC clock source select APB2/8
RCU_ADCCK_AHB_DIV9	ADC clock source select AHB/9

### rcu\_osc\_type\_enum

Table 3-345. Enum rcu\_osc\_type\_enum

enum name	Function description
RCU_HXTAL	HXTAL
RCU_LXTAL	LXTAL

enum name	Function description
RCU_IRC8M	IRC8M
RCU_IRC28M	IRC28M
RCU_IRC48M	IRC48M
RCU_IRC40K	IRC40K
RCU_PLL_CK	PLL

### rcu\_clock\_freq\_enum

Table 3-346. Enum rcu\_clock\_freq\_enum

enum name	Function description
CK_SYS	system clock
CK_AHB	AHB clock
CK_APB1	APB1 clock
CK_APB2	APB2 clock
CK_ADC	ADC clock
CK_CEC	CEC clock
CK_USART	USART clock

### rcu\_deinit

The description of rcu\_deinit is shown as below:

Table 3-347. 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 3-348. Function rcu\_periph\_clock\_enable**

<b>Function name</b>	rcu_periph_clock_enable
<b>Function prototype</b>	void rcu_periph_clock_enable(rcu_periph_enum periph);
<b>Function descriptions</b>	enable the peripherals clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-337. Enum rcu_periph_enum</a> .
<i>RCU_GPIOx</i>	GPIO ports clock (x=A,B,C,D,F)
<i>RCU_DMA</i>	DMA clock
<i>RCU_CRC</i>	CRC clock
<i>RCU_TSI</i>	TSI clock
<i>RCU_CFGCMP</i>	CFGCMP clock
<i>RCU_ADC</i>	ADC clock
<i>RCU_TIMERx</i>	TIMERx clock(x=0,1,2,5,13,14,15,16), RCU_TIMER5 only for GD32F350
<i>RCU_SPIx</i>	SPIx clock (x=0,1)
<i>RCU_USARTx</i>	USARTx clock (x=0,1)
<i>RCU_WWDGT</i>	WWDGT clock
<i>RCU_I2Cx</i>	I2Cx clock (x=0,1)
<i>RCU_USBFS</i>	USBFS clock (only for GD32F350)
<i>RCU_PMU</i>	PMU clock
<i>RCU_DAC</i>	DAC clock (only for GD32F350)
<i>RCU_CEC</i>	CEC clock (only for GD32F350)
<i>RCU_CTC</i>	CTC clock
<i>RCU_RTC</i>	RTC clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the USART0 clock */
rcu_periph_clock_enable(RCU_USART0);
```

### **rcu\_periph\_clock\_disable**

The description of rcu\_periph\_clock\_disable is shown as below:

**Table 3-349. Function rcu\_periph\_clock\_disable**

<b>Function name</b>	rcu_periph_clock_disable
<b>Function prototype</b>	void rcu_periph_clock_disable(rcu_periph_enum periph);
<b>Function descriptions</b>	disable the peripherals clock
<b>Precondition</b>	-

The called functions	-
Input parameter{in}	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-337. Enum rcu_periph_enum</a> .
<i>RCU_GPIOx</i>	GPIO ports clock (x=A,B,C,D,F)
<i>RCU_DMA</i>	DMA clock
<i>RCU_CRC</i>	CRC clock
<i>RCU_TSI</i>	TSI clock
<i>RCU_CFGCMP</i>	CFGCMP clock
<i>RCU_ADC</i>	ADC clock
<i>RCU_TIMERx</i>	TIMERx clock(x=0,1,2,5,13,14,15,16),(RCU_TIMER5 only for GD32F350)
<i>RCU_SPIx</i>	SPIx clock (x=0,1)
<i>RCU_USARTx</i>	USARTx clock (x=0,1)
<i>RCU_WWDGT</i>	WWDGT clock
<i>RCU_I2Cx</i>	I2Cx clock (x=0,1)
<i>RCU_USBFS</i>	USBFS clock (only for GD32F350)
<i>RCU_PMU</i>	PMU clock
<i>RCU_DAC</i>	DAC clock (only for GD32F350)
<i>RCU_CEC</i>	CEC clock (only for GD32F350)
<i>RCU_CTC</i>	CTC clock
<i>RCU_RTC</i>	RTC clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the USART0 clock */
rcu_periph_clock_disable(RCU_USART0);
```

### **rcu\_periph\_clock\_sleep\_enable**

The description of `rcu_periph_clock_sleep_enable` is shown as below:

**Table 3-350. Function `rcu_periph_clock_sleep_enable`**

<b>Function name</b>	<code>rcu_periph_clock_sleep_enable</code>
<b>Function prototype</b>	<code>void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);</code>
<b>Function descriptions</b>	enable the peripherals clock when in sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-338. Enum rcu_periph_sleep_enum</a> .
<i>RCU_FMC_SLP</i>	FMC clock
<i>RCU_SRAM_SLP</i>	SRAM clock

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the FMC clock when in sleep mode */
rcu_periph_clock_sleep_enable(RCU_FMC_SLP);
```

### **rcu\_periph\_clock\_sleep\_disable**

The description of `rcu_periph_clock_sleep_disable` is shown as below:

**Table 3-351. 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>	-
Input parameter{in}	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-338. Enum <code>rcu_periph_sleep_enum</code>.</a>
<i>RCU_FMC_SLP</i>	FMC clock
<i>RCU_SRAM_SLP</i>	SRAM clock
Output parameter{out}	
-	-
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 3-352. Function `rcu_periph_reset_enable`**

<b>Function name</b>	rcu_periph_reset_enable
<b>Function prototype</b>	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);
<b>Function descriptions</b>	enable the peripherals reset
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>periph_reset</b>	RCU peripherals reset, refer to <a href="#">Table 3-339. Enum</a>

	<a href="#"><i>rcu_periph_reset_enum</i></a>
<i>RCU_GPIOxRST</i>	reset GPIO ports clock (x=A,B,C,D,F)
<i>RCU_TSIRST</i>	reset TSI
<i>RCU_CFGCMRST</i>	reset CFGCMP clock
<i>RCU_ADCRST</i>	reset ADC clock
<i>RCU_TIMERxRST</i>	reset TIMERx clock (x=0,1,2,5,13,14,15,16), RCU_TIMER5 only for GD32F350
<i>RCU_SPIxRST</i>	reset SPIx clock (x=0,1)
<i>RCU_USARTxRST</i>	reset USARTx clock (x=0,1)
<i>RCU_WWDGTRST</i>	reset WWDGT clock
<i>RCU_I2CxRST</i>	reset I2Cx clock (x=0,1)
<i>RCU_USBFSRST</i>	reset USBFS (only for GD32F350)
<i>RCU_PMURST</i>	reset PMU clock
<i>RCU_DACRST</i>	reset DAC (only for GD32F350)
<i>RCU_CECRST</i>	reset CEC (only for GD32F350)
<i>RCU_CTCRST</i>	reset CTC
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 reset */
rcu_periph_reset_enable(RCU_SPI0RST);
```

### **rcu\_periph\_reset\_disable**

The description of `rcu_periph_reset_disable` is shown as below:

**Table 3-353. Function `rcu_periph_reset_disable`**

<b>Function name</b>	<code>rcu_periph_reset_disable</code>
<b>Function prototype</b>	<code>void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);</code>
<b>Function descriptions</b>	disable the peripheral reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph_reset</b>	RCU peripherals reset, refer to <a href="#"><u>Table 3-339. Enum <i>rcu_periph_reset_enum</i></u></a> .
<i>RCU_GPIOxRST</i>	reset GPIO ports clock (x=A,B,C,D,F)
<i>RCU_TSIRST</i>	reset TSI
<i>RCU_CFGCMRST</i>	reset CFGCMP clock
<i>RCU_ADCRST</i>	reset ADC clock
<i>RCU_TIMERxRST</i>	reset TIMERx clock (x=0,1,2,5,13,14,15,16), RCU_TIMER5 only for

	GD32F350
<i>RCU_SPIxRST</i>	reset SPIx clock (x=0,1)
<i>RCU_USARTxRST</i>	reset USARTx clock (x=0,1)
<i>RCU_WWDGTRST</i>	reset WWDGT clock
<i>RCU_I2CxRST</i>	reset I2Cx clock (x=0,1)
<i>RCU_USBFSRST</i>	reset USBFS (only for GD32F350)
<i>RCU_PMURST</i>	reset PMU clock
<i>RCU_DACRST</i>	reset DAC (only for GD32F350)
<i>RCU_CECRST</i>	reset CEC (only for GD32F350)
<i>RCU_CTCRST</i>	reset CTC
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 reset */
rcu_periph_reset_disable(RCU_SPI0RST);
```

### **rcu\_bkp\_reset\_enable**

The description of `rcu_bkp_reset_enable` is shown as below:

**Table 3-354. Function `rcu_bkp_reset_enable`**

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

Example:

```
/* reset the BKP domain */
rcu_bkp_reset_enable();
```

### **rcu\_bkp\_reset\_disable**

The description of `rcu_bkp_reset_disable` is shown as below:

**Table 3-355. Function rcu\_bkp\_reset\_disable**

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

Example:

```
/* disable the BKP domain reset */

rcu_bkp_reset_disable();
```

### **rcu\_system\_clock\_source\_config**

The description of rcu\_system\_clock\_source\_config is shown as below:

**Table 3-356. Function rcu\_system\_clock\_source\_config**

<b>Function name</b>	rcu_system_clock_source_config
<b>Function prototype</b>	void rcu_system_clock_source_config(uint32_t ck_sys);
<b>Function descriptions</b>	configure the system clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_sys</b>	system clock source select
<i>RCU_CKSYSRC_IRC_8M</i>	select CK_IRC8M as the CK_SYS source
<i>RCU_CKSYSRC_HXTAL</i>	select CK_HXTAL as the CK_SYS source
<i>RCU_CKSYSRC_PLL</i>	select CK_PLL as the CK_SYS source
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the CK_HXTAL as the CK_SYS source */

rcu_system_clock_source_config(RCU_CKSYSRC_HXTAL);
```

### **rcu\_system\_clock\_source\_get**

The description of rcu\_system\_clock\_source\_get is shown as below:

**Table 3-357. Function rcu\_system\_clock\_source\_get**

<b>Function name</b>	rcu_system_clock_source_get
<b>Function prototype</b>	uint32_t rcu_system_clock_source_get(void);
<b>Function descriptions</b>	get the system clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	RCU_SCSS_IRC8M/RCU_SCSS_HXTAL/RCU_SCSS_PLL

Example:

```
uint32_t temp_cksys_status;  
  
/* get the CK_SYS source */  
  
temp_cksys_status = rcu_system_clock_source_get();
```

### **rcu\_ahb\_clock\_config**

The description of rcu\_ahb\_clock\_config is shown as below:

**Table 3-358. Function rcu\_ahb\_clock\_config**

<b>Function name</b>	rcu_ahb_clock_config
<b>Function prototype</b>	void rcu_ahb_clock_config(uint32_t ck_ahb);
<b>Function descriptions</b>	configure the AHB clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_ahb</b>	AHB clock prescaler selection
<b>RCU_AHB_CKSYS_DI Vx</b>	select CK_SYS / x, (x=1, 2, 4, 8, 16, 64, 128, 256, 512)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure CK_SYS/128 */
```

---

```
rcu_ahb_clock_config(RCU_AHB_CKSYS_DIV128);
```

### **rcu\_apb1\_clock\_config**

The description of rcu\_apb1\_clock\_config is shown as below:

**Table 3-359. Function rcu\_apb1\_clock\_config**

<b>Function name</b>	rcu_apb1_clock_config
<b>Function prototype</b>	void rcu_apb1_clock_config(uint32_t ck_apb1);
<b>Function descriptions</b>	configure the APB1 clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_apb1</b>	APB1 clock prescaler selection
<i>RCU_APB1_CKAHB_D /Vx</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 3-360. 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 /Vx</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 3-361. 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, refer to <a href="#">Table 3-344. Enum rcu_adc_clock_enum</a> .
<i>RCU_ADCCK_IRC28M_DIV2</i>	select CK_IRC28M / 2 as CK_ADC
<i>RCU_ADCCK_IRC28M</i>	select CK_IRC28M as CK_ADC
<i>RCU_ADCCK_AHB_DIVx</i>	select (CK_AHB / x) as CK_ADC(x=3,5,7,9)
<i>RCU_ADCCK_APB2_DIVx</i>	select (CK_APB2 / x) as CK_ADC(x=2,4,6,8)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the ADC prescaler factor */
rcu_adc_clock_config(RCU_CKADC_CKAPB2_DIV8);
```

## rcu\_usvfs\_clock\_config

The description of rcu\_usvfs\_clock\_config is shown as below:

**Table 3-362. Function rcu\_usvfs\_clock\_config**

<b>Function name</b>	rcu_usvfs_clock_config
<b>Function prototype</b>	void rcu_usvfs_clock_config(uint32_t ck_usvfs);
<b>Function descriptions</b>	configure the USBFS clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_usvfs</b>	USBFS clock prescaler selection

<i>RCU_USBFS_CKPLL_DIV1</i>	select CK_PLL as CK_USBFS
<i>RCU_USBFS_CKPLL_DIV1_5</i>	select CK_PLL / 1.5 as CK_USBFS
<i>RCU_USBFS_CKPLL_DIV2</i>	select CK_PLL / 2 as CK_USBFS
<i>RCU_USBFS_CKPLL_DIV2_5</i>	select CK_PLL / 2.5 as CK_USBFS
<i>RCU_USBFS_CKPLL_DIV3</i>	select CK_PLL / 3 as CK_USBFS
<i>RCU_USBFS_CKPLL_DIV3_5</i>	select CK_PLL / 3.5 as CK_USBFS
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the CK_PLL as CK_USBFS */
rcu_usbfs_clock_config(RCU_USBFS_CKPLL_DIV1);
```

### **rcu\_ckout\_config**

The description of `rcu_ckout_config` is shown as below:

**Table 3-363. Function `rcu_ckout_config`**

<b>Function name</b>	<code>rcu_ckout_config</code>
<b>Function prototype</b>	<code>void rcu_ckout_config(uint32_t ckout_src, uint32_t ckout_div);</code>
<b>Function descriptions</b>	configure the CK_OUT clock source and division factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ckout_src</b>	CK_OUT clock source selection
<i>RCU_CKOUTSRC_NO_NE</i>	no clock selected
<i>RCU_CKOUTSRC_IRC_28M</i>	select high speed 28M internal oscillator clock
<i>RCU_CKOUTSRC_IRC_40K</i>	select high speed 40K internal oscillator clock
<i>RCU_CKOUTSRC_LXTAL</i>	select LXTAL clock
<i>RCU_CKOUTSRC_CKSYS</i>	select system clock CK_SYS

<i>RCU_CKOUTSRC_IRC 8M</i>	select high speed 8M internal oscillator clock
<i>RCU_CKOUTSRC_HXTAL</i>	select HXTAL clock
<i>RCU_CKOUTSRC_CK PLL_DIV1</i>	select CK_PLL clock
<i>RCU_CKOUTSRC_CK PLL_DIV2</i>	Select (CK_PLL / 2) clock
<b>Input parameter{in}</b>	
<b>ckout_div</b>	CK_OUT divider
<i>RCU_CKOUT_DIVx</i>	CK_OUT is divided by x(x=1,2,4,8,16,32,64,128)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the HXTAL as CK_OUT clock source */
rcu_ckout_config(RCU_CKOUTSRC_HXTAL);
```

### **rcu\_pll\_preselection\_config**

The description of `rcu_pll_preselection_config` is shown as below:

**Table 3-364. Function `rcu_pll_preselection_config`**

<b>Function name</b>	<code>rcu_pll_preselection_config</code>
<b>Function prototype</b>	<code>void rcu_pll_preselection_config(uint32_t pll_presel);</code>
<b>Function descriptions</b>	configure the PLL clock source preselection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pll_presel</b>	PLL clock source selection
<i>RCU_PLLPRESEL_IR C48M</i>	select IRC48M as PLL preselection clock
<i>RCU_PLLPRESEL_HXTAL</i>	select HXTAL as PLL preselection clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select HXTAL as PLL preselection clock */
```

---

```
rcu_pll_preselection_config(RCU_PLLPRESEL_HXTAL);
```

### **rcu\_pll\_config**

The description of rcu\_pll\_config is shown as below:

**Table 3-365. Function rcu\_pll\_config**

<b>Function name</b>	rcu_pll_config
<b>Function prototype</b>	void rcu_pll_config(uint32_t pll_src, uint32_t pll_mul);
<b>Function descriptions</b>	configure the PLL clock source selection and PLL multiply factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pll_src</b>	PLL clock source selection
<i>RCU_PLLSRC_IRC8M_DIV2</i>	select CK_IRC8M/2 as PLL source clock
<i>RCU_PLLSRC_HXTAL_IRC48M</i>	select HXTAL or 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 = 2..64)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL */
rcu_pll_config(RCU_PLLSRC_HXTAL_IRC48M, RCU_PLL_MUL10);
```

### **rcu\_usart\_clock\_config**

The description of rcu\_usart\_clock\_config is shown as below:

**Table 3-366. Function rcu\_usart\_clock\_config**

<b>Function name</b>	rcu_usart_clock_config
<b>Function prototype</b>	void rcu_usart_clock_config(uint32_t ck_usart);
<b>Function descriptions</b>	configure the USART clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_usart</b>	USART clock source selection
<i>RCU_USART0SRC_CKAPB2</i>	CK_USART0 select CK_APB2

<i>RCU_USART0SRC_C KSYS</i>	CK_USART0 select CK_SYS
<i>RCU_USART0SRC_LX TAL</i>	CK_USART0 select CK_LXTAL
<i>RCU_USART0SRC_IR C8M</i>	CK_USART0 select CK_IRC8M
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the LXTAL as USART0 clock */
rcu_usart_clock_config(RCU_USART0SRC_LXTAL);
```

### **rcu\_cec\_clock\_config**

The description of **rcu\_cec\_clock\_config** is shown as below:

**Table 3-367. Function **rcu\_cec\_clock\_config****

<b>Function name</b>	rcu_cec_clock_config
<b>Function prototype</b>	void rcu_cec_clock_config(uint32_t ck_cec);
<b>Function descriptions</b>	configure the CEC clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_cec</b>	CEC clock source selection
<i>RCU_CECSRC_IRC8M _DIV244</i>	CK_CEC select CK_IRC8M/244
<i>RCU_CECSRC_LXTAL</i>	select CK_LXTAL
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure theCEC clock source selection */
rcu_cec_clock_config(RCU_CECSRC_LXTAL);
```

### **rcu\_rtc\_clock\_config**

The description of **rcu\_rtc\_clock\_config** is shown as below:

**Table 3-368. Function rcu\_rtc\_clock\_config**

<b>Function name</b>	rcu_rtc_clock_config
<b>Function prototype</b>	void rcu_rtc_clock_config(uint32_t rtc_clock_source);
<b>Function descriptions</b>	configure the RTC clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_clock_source</b>	RTC clock source selection
<i>RCU_RTCSRC_NONE</i>	no clock selected
<i>RCU_RTCSRC_LXTAL</i>	select CK_LXTAL as RTC source clock
<i>RCU_RTCSRC_IRC40K</i>	select CK_IRC40K as RTC source clock
<i>RCU_RTCSRC_HXTAL_DIV_32</i>	select (CK_HXTAL / 32) as RTC source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the RTC clock source selection */
rcu_rtc_clock_config(RCU_RTCSRC_IRC40K);
```

### **rcu\_ck48m\_clock\_config**

The description of rcu\_ck48m\_clock\_config is shown as below:

**Table 3-369. Function rcu\_ck48m\_clock\_config**

<b>Function name</b>	rcu_ck48m_clock_config
<b>Function prototype</b>	void rcu_ck48m_clock_config(uint32_t ck48m_clock_source);
<b>Function descriptions</b>	configure the CK48M clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck48m_clock_source</b>	CK48M clock source selection
<i>RCU_CK48MSRC_PLL48M</i>	CK_PLL48M selected as CK48M source clock
<i>RCU_CK48MSRC_IRC48M</i>	CK_IRC48M selected as CK48M source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the CK_48M clock source selection */
rcu_ck48m_clock_config(RCU_CK48MSRC_IRC48M);
```

### **rcu\_hxtal\_prediv\_config**

The description of `rcu_hxtal_prediv_config` is shown as below:

**Table 3-370. Function `rcu_hxtal_prediv_config`**

<b>Function name</b>	rcu_hxtal_prediv_config
<b>Function prototype</b>	void rcu_hxtal_prediv_config(uint32_t hxtal_prediv)
<b>Function descriptions</b>	configure the HXTAL divider used as input of PLL
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>hxtal_prediv</b>	HXTAL divider used as input of PLL
<b>RCU_PLL_PREDVx</b>	HXTAL or IRC48M divided x used as input of PLL (x=1..16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL clock source selection*/
rcu_hxtal_prediv_config(RCU_PLL_PREDV2);
```

### **rcu\_lxtal\_drive\_capability\_config**

The description of `rcu_lxtal_drive_capability_config` is shown as below:

**Table 3-371. Function `rcu_lxtal_drive_capability_config`**

<b>Function name</b>	rcu_lxtal_drive_capability_config
<b>Function prototype</b>	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);
<b>Function descriptions</b>	configure the LXTAL drive capability
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lxtal_dricap</b>	drive capability of LXTAL
<b>RCU_LXTAL_LO_WDRI</b>	lower driving capability
<b>RCU_LXTAL_MED_LO_WDRI</b>	medium low driving capability
<b>RCU_LXTAL_MED_HI_GHDMI</b>	medium high driving capability

<i>RCU_LXTAL_HIGHDRI</i>	higher driving capability
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the LXTAL drive capability */
rcu_lxtal_drive_capability_config (RCU_LXTAL_LOWDRI);
```

### **rcu\_flag\_get**

The description of `rcu_flag_get` is shown as below:

**Table 3-372. Function `rcu_flag_get`**

<b>Function name</b>	<code>rcu_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus rcu_flag_get(rcu_flag_enum flag);</code>
<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 3-340. Enum <code>rcu_flag_enum</code>.</a>
<i>RCU_FLAG_IRC40KST</i> <i>B</i>	IRC40K stabilization flag
<i>RCU_FLAG_LXTALST</i> <i>B</i>	LXTAL stabilization flag
<i>RCU_FLAG_IRC8MST</i> <i>B</i>	IRC8M stabilization flag
<i>RCU_FLAG_HXTALST</i> <i>B</i>	HXTAL stabilization flag
<i>RCU_FLAG_PLLSTB</i>	PLL stabilization flag
<i>RCU_FLAG_IRC28MS</i> <i>TB</i>	IRC28M stabilization flag
<i>RCU_FLAG_IRC48MS</i> <i>TB</i>	IRC48M stabilization flag
<i>RCU_FLAG_V12RST</i>	V12 domain power reset flag
<i>RCU_FLAG_OBLRST</i>	option byte loader reset flag
<i>RCU_FLAG_EPRST</i>	external PIN reset flag
<i>RCU_FLAG_PORRST</i>	power reset flag
<i>RCU_FLAG_SWRST</i>	software reset flag
<i>RCU_FLAG_FWDGTR</i> <i>ST</i>	free watchdog timer reset flag

<i>RCU_FLAG_WWDGTR_ST</i>	window watchdog timer reset flag
<i>RCU_FLAG_LPRST</i>	low-power reset flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the clock stabilization flag */

if(RESET != rcu_flag_get(RCU_FLAG_LXTALSTB)){

}
```

### **rcu\_all\_reset\_flag\_clear**

The description of `rcu_all_reset_flag_clear` is shown as below:

**Table 3-373. 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 3-374. 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>	-

The called functions		-
Input parameter{in}		
<b>int_flag</b>	interrupt and ckm flags, refer to <a href="#">Table 3-341. Enum rcu_int_flag enum</a> .	
<i>RCU_INT_FLAG_IRC4OKSTB</i>	IRC40K stabilization interrupt flag	
<i>RCU_INT_FLAG_LXTALSTB</i>	LXTAL stabilization interrupt flag	
<i>RCU_INT_FLAG_IRC8MSTB</i>	IRC8M stabilization interrupt flag	
<i>RCU_INT_FLAG_HXTALSTB</i>	HXTAL stabilization interrupt flag	
<i>RCU_INT_FLAG_PLLSTB</i>	PLL stabilization interrupt flag	
<i>RCU_INT_FLAG_IRC28MSTB</i>	IRC28M stabilization interrupt flag	
<i>RCU_INT_FLAG_IRC48MSTB</i>	IRC48M stabilization interrupt flag	
<i>RCU_INT_FLAG_CKM</i>	HXTAL clock stuck interrupt flag	
Output parameter{out}		
-	-	-
Return value		
<b>FlagStatus</b>	SET or RESET	

Example:

```
/* get the clock stabilization interrupt flag */

if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){
}
```

### **rcu\_interrupt\_flag\_clear**

The description of `rcu_interrupt_flag_clear` is shown as below:

**Table 3-375. 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>	-
Input parameter{in}	
<b>int_flag_clear</b>	clock stabilization and stuck interrupt flags clear, refer to <a href="#">Table 3-342. Enum rcu_int_flag_clear enum</a> .
<i>RCU_INT_FLAG_IRC4OKSTB_CLR</i>	IRC40K stabilization interrupt flag clear

<i>RCU_INT_FLAG_LXTA LSTB_CLR</i>	LXTAL stabilization interrupt flag clear
<i>RCU_INT_FLAG_IRC8 MSTB_CLR</i>	IRC8M stabilization interrupt flag clear
<i>RCU_INT_FLAG_HXT ALSTB_CLR</i>	HXTAL stabilization interrupt flag clear
<i>RCU_INT_FLAG_PLLS TB_CLR</i>	PLL stabilization interrupt flag clear
<i>RCU_INT_FLAG_IRC2 8MSTB_CLR</i>	IRC28M stabilization interrupt flag clear
<i>RCU_INT_FLAG_IRC4 8MSTB_CLR</i>	IRC48M stabilization interrupt flag clear
<i>RCU_INT_FLAG_CKM _CLR</i>	clock stuck interrupt flag clear
<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 3-376. Function `rcu_interrupt_enable`**

<b>Function name</b>	<code>rcu_interrupt_enable</code>
<b>Function prototype</b>	<code>void rcu_interrupt_enable(rcu_int_enum stab_int);</code>
<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 3-343. Enum <code>rcu_int_enum</code>.</a>
<i>RCU_INT_IRC40KSTB</i>	IRC40K stabilization interrupt enable
<i>RCU_INT_LXTALSTB</i>	LXTAL stabilization interrupt enable
<i>RCU_INT_IRC8MSTB</i>	IRC8M stabilization interrupt enable
<i>RCU_INT_HXTALSTB</i>	HXTAL stabilization interrupt enable
<i>RCU_INT_PLLSTB</i>	PLL stabilization interrupt enable
<i>RCU_INT_IRC28MSTB</i>	IRC28M stabilization interrupt enable
<i>RCU_INT_IRC48MSTB</i>	IRC48M stabilization interrupt enable
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* enable the HXTAL stabilization interrupt */
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

### **rcu\_interrupt\_disable**

The description of `rcu_interrupt_disable` is shown as below:

**Table 3-377. Function `rcu_interrupt_disable`**

<b>Function name</b>	rcu_interrupt_disable
<b>Function prototype</b>	void rcu_interrupt_disable(rcu_int_enum stab_int);
<b>Function descriptions</b>	disable the stabilization interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>stab_int</b>	clock stabilization interrupt, refer to <a href="#">Table 3-343. Enum <code>rcu_int_enum</code></a> .
<i>RCU_INT_IRC40KSTB</i>	IRC40K stabilization interrupt disable
<i>RCU_INT_LXTALSTB</i>	LXTAL stabilization interrupt disable
<i>RCU_INT_IRC8MSTB</i>	IRC8M stabilization interrupt disable
<i>RCU_INT_HXTALSTB</i>	HXTAL stabilization interrupt disable
<i>RCU_INT_PLLSTB</i>	PLL stabilization interrupt disable
<i>RCU_INT_IRC28MSTB</i>	IRC28M stabilization interrupt disable
<i>RCU_INT_IRC48MSTB</i>	IRC48M stabilization interrupt disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the HXTAL stabilization interrupt */
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

### **rcu\_oscil\_stab\_wait**

The description of `rcu_oscil_stab_wait` is shown as below:

**Table 3-378. Function `rcu_oscil_stab_wait`**

<b>Function name</b>	rcu_oscil_stab_wait
<b>Function prototype</b>	ErrStatus rcu_oscil_stab_wait(rcu_oscil_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 3-345. Enum rcu_osc_type_enum</a> .
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<i>RCU_IRC8M</i>	internal 8M RC oscillators(IRC8M)
<i>RCU_IRC28M</i>	internal 28M RC oscillators(IRC28M)
<i>RCU_IRC48M</i>	internal 48M RC oscillators(IRC48M)
<i>RCU_IRC40K</i>	internal 40K RC oscillator(IRC40K)
<i>RCU_PLL_CK</i>	phase locked loop(PLL)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* wait for oscillator stabilization flag */

if(SUCCESS == rcu_osc_stab_wait(RCU_HXTAL)){

}
```

### rcu\_osc\_on

The description of rcu\_osc\_on is shown as below:

**Table 3-379. Function rcu\_osc\_on**

<b>Function name</b>	rcu_osc_on
<b>Function prototype</b>	void rcu_osc_on(rcu_osc_type_enum osci);
<b>Function descriptions</b>	turn on the oscillator
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-345. Enum rcu_osc_type_enum</a> .
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<i>RCU_IRC8M</i>	internal 8M RC oscillators(IRC8M)
<i>RCU_IRC28M</i>	internal 28M RC oscillators(IRC28M)
<i>RCU_IRC48M</i>	internal 48M RC oscillators(IRC48M)
<i>RCU_IRC40K</i>	internal 40K RC oscillator(IRC40K)
<i>RCU_PLL_CK</i>	phase locked loop(PLL)
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* turn on the high speed crystal oscillator */
rcu_osc_on(RCU_HXTAL);
```

### **rcu\_osc\_off**

The description of `rcu_osc_off` is shown as below:

**Table 3-380. Function `rcu_osc_off`**

<b>Function name</b>	rcu_osc_off
<b>Function prototype</b>	void rcu_osc_off(rcu_osc_type_enum osci);
<b>Function descriptions</b>	turn off the oscillator
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-345. Enum <code>rcu_osc_type_enum</code></a> .
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<i>RCU_IRC8M</i>	internal 8M RC oscillators(IRC8M)
<i>RCU_IRC28M</i>	internal 28M RC oscillators(IRC28M)
<i>RCU_IRC48M</i>	internal 48M RC oscillators(IRC48M)
<i>RCU_IRC40K</i>	internal 40K RC oscillator(IRC40K)
<i>RCU_PLL_CK</i>	phase locked loop(PLL)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn off the high speed crystal oscillator */
rcu_osc_off(RCU_HXTAL);
```

### **rcu\_osc\_bypass\_mode\_enable**

The description of `rcu_osc_bypass_mode_enable` is shown as below:

**Table 3-381. Function `rcu_osc_bypass_mode_enable`**

<b>Function name</b>	rcu_osc_bypass_mode_enable
<b>Function prototype</b>	void rcu_osc_bypass_mode_enable(rcu_osc_type_enum osci);
<b>Function descriptions</b>	enable the oscillator bypass mode
<b>Precondition</b>	HXTALEN or LXTALEN must be reset before it

The called functions	-
Input parameter{in}	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-345. Enum rcu_osc_type enum</a> .
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the high speed crystal oscillator bypass mode */
rcu_osc_bypass_mode_enable(RCU_HXTAL);
```

### **rcu\_osc\_bypass\_mode\_disable**

The description of `rcu_osc_bypass_mode_disable` is shown as below:

**Table 3-382. Function `rcu_osc_bypass_mode_disable`**

Function name	rcu_osc_bypass_mode_disable
Function prototype	void rcu_osc_bypass_mode_disable(rcu_osc_type_enum osci);
Function descriptions	disable the oscillator bypass mode
Precondition	HXTALEN or LXTALEN must be reset before it
The called functions	-
Input parameter{in}	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-345. Enum rcu_osc_type enum</a> .
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the high speed crystal oscillator bypass mode */
rcu_osc_bypass_mode_disable(RCU_HXTAL);
```

### **rcu\_hxtal\_clock\_monitor\_enable**

The description of `rcu_hxtal_clock_monitor_enable` is shown as below:

**Table 3-383. Function `rcu_hxtal_clock_monitor_enable`**

Function name	rcu_hxtal_clock_monitor_enable
Function prototype	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 3-384. Function `rcu_hxtal_clock_monitor_disable`**

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

Example:

```
/* disable the HXTAL clock monitor */
rcu_hxtal_clock_monitor_disable();
```

### **`rcu_irc8m_adjust_value_set`**

The description of `rcu_irc8m_adjust_value_set` is shown as below:

**Table 3-385. Function `rcu_irc8m_adjust_value_set`**

<b>Function name</b>	rcu_irc8m_adjust_value_set
<b>Function prototype</b>	void rcu_irc8m_adjust_value_set(uint32_t irc8m_adjval);
<b>Function descriptions</b>	set the IRC8M adjust value
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>irc8m_adjval</b>	IRC8M adjust value, must be between 0 and 0x1F
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the IRC8M adjust value */
rcu_irc8m_adjust_value_set(0x10);
```

### **rcu\_irc28m\_adjust\_value\_set**

The description of `rcu_irc28m_adjust_value_set` is shown as below:

**Table 3-386. Function `rcu_irc28m_adjust_value_set`**

<b>Function name</b>	rcu_irc28m_adjust_value_set
<b>Function prototype</b>	void rcu_irc28m_adjust_value_set(uint32_t irc28m_adjval);
<b>Function descriptions</b>	set the IRC28M adjust value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>irc28m_adjval</b>	IRC28M adjust value, must be between 0 and 0x1F
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the IRC28M adjust value */
rcu_irc28m_adjust_value_set(0x10);
```

### **rcu\_voltage\_key\_unlock**

The description of `rcu_voltage_key_unlock` is shown as below:

**Table 3-387. 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\_deepsleep\_voltage\_set**

The description of `rcu_deepsleep_voltage_set` is shown as below:

**Table 3-388. Function `rcu_deepsleep_voltage_set`**

<b>Function name</b>	<code>rcu_deepsleep_voltage_set</code>
<b>Function prototype</b>	<code>void rcu_deepsleep_voltage_set(uint32_t dsvol);</code>
<b>Function descriptions</b>	set voltage in deep sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dsvol</b>	deep sleep mode voltage
<i>RCU_DEEPSLEEP_V_1_0</i>	the core voltage is 1.0V in deep-sleep mode
<i>RCU_DEEPSLEEP_V_0_9</i>	the core voltage is 0.9V in deep-sleep mode
<i>RCU_DEEPSLEEP_V_0_8</i>	the core voltage is 0.8V in deep-sleep mode
<i>RCU_DEEPSLEEP_V_0_7</i>	the core voltage is 0.7V in deep-sleep mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the deep-sleep mode voltage */

rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1_0);
```

### **rcu\_clock\_freq\_get**

The description of `rcu_clock_freq_get` is shown as below:

**Table 3-389. Function rcu\_clock\_freq\_get**

<b>Function name</b>	rcu_clock_freq_get
<b>Function prototype</b>	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);
<b>Function descriptions</b>	get the system clock, bus clock and peripheral clock frequency
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>clock</b>	the clock frequency which to get
<i>CK_SYS</i>	system clock frequency
<i>CK_AHB</i>	AHB clock frequency
<i>CK_APB1</i>	APB1 clock frequency
<i>CK_APB2</i>	APB2 clock frequency
<i>CK_ADC</i>	ADC clock frequency
<i>CK_CEC</i>	CEC clock frequency
<i>CK_USART</i>	USART clock frequency
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	clock frequency of system, AHB, APB1, APB2, ADC or USART

Example:

```
uint32_t temp_freq;

/* get the system clock frequency */

temp_freq = rcu_clock_freq_get(CK_SYS);
```

## 3.18. RTC

The Real-time Clock (RTC) is usually used as a clock-calendar. The ones in the Backup Domain consist of a 32-bit up-counter, an alarm, a prescaler, a divider and the RTC clock configuration register. The RTC registers are listed in chapter [3.18.1](#), the FWDGT firmware functions are introduced in chapter [3.18.2](#).

### 3.18.1. Descriptions of Peripheral registers

RTC registers are listed in the table shown as below:

**Table 3-390. 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

Registers	Descriptions
RTC_PSC	RTC time prescaler register
RTC_ALRM0TD	RTC alarm 0 time and date register
RTC_WPK	RTC write protection key register
RTC_SS	RTC sub second register
RTC_SHIFTCTL	RTC shift function control register
RTC_TTS	RTC time of timestamp register
RTC_DTS	RTC date of timestamp register
RTC_SSTS	RTC sub second of timestamp register
RTC_HRFC	RTC high resolution frequency compensation register
RTC_TAMP	RTC tamper register
RTC_ALRM0SS	RTC alarm 0 sub second register
RTC_BKP0	RTC backup 0 register
RTC_BKP1	RTC backup 1 register
RTC_BKP2	RTC backup 2 register
RTC_BKP3	RTC backup 3 register
RTC_BKP4	RTC backup 4 register

### 3.18.2. Descriptions of Peripheral functions

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

**Table 3-391. RTC firmware function**

Function name	Function description
rtc_deinit	reset most of the RTC registers
rtc_init	initialize RTC registers
rtc_init_mode_enter	enter RTC init mode
rtc_init_mode_exit	exit RTC init mode
rtc_register_sync_wait	wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated
rtc_current_time_get	get current time and date
rtc_subsecond_get	get current subsecond value
rtc_alarm_config	configure RTC alarm
rtc_alarm_subsecond_config	configure subsecond of RTC alarm
rtc_alarm_get	get RTC alarm
rtc_alarm_subsecond_get	get RTC alarm subsecond
rtc_alarm_enable	enable RTC alarm
rtc_alarm_disable	disable RTC alarm
rtc_timestamp_enable	enable RTC time-stamp
rtc_timestamp_disable	disable RTC time-stamp
rtc_timestamp_get	get RTC timestamp time and date
rtc_timestamp_subsecond_get	get RTC time-stamp subsecond

Function name	Function description
rtc_tamper_enable	enable RTC tamper
rtc_tamper_disable	disable RTC tamper
rtc_interrupt_enable	enable specified RTC interrupt
rtc_interrupt_disable	disable specified RTC interrupt
rtc_flag_get	check specified flag
rtc_flag_clear	clear specified flag
rtc_alter_output_config	configure RTC alternate output source
rtc_calibration_config	configure RTC calibration register
rtc_hour_adjust	adjust the daylight saving time by adding or subtracting one hour from the current time
rtc_second_adjust	adjust RTC second or subsecond value of current time
rtc_bypass_shadow_enable	enable RTC bypass shadow registers function
rtc_bypass_shadow_disable	disable RTC bypass shadow registers function
rtc_refclock_detection_enable	enable RTC reference clock detection function
rtc_refclock_detection_disable	disable RTC reference clock detection function

## Structure rtc\_parameter\_struct

**Table 3-392. Structure rtc\_parameter\_struct**

Member name	Function description
rtc_year	RTC year value: 0x0 - 0x99(BCD format)
rtc_month	RTC month value
rtc_date	RTC date value: 0x1 - 0x31(BCD format)
rtc_day_of_week	RTC weekday value
rtc_hour	RTC hour value
rtc_minute	RTC minute value: 0x0 - 0x59(BCD format)
rtc_second	RTC second value: 0x0 - 0x59(BCD format)
rtc_factor_asyn	RTC asynchronous prescaler value: 0x0 - 0x7F
rtc_factor_syn	RTC synchronous prescaler value: 0x0 - 0x7FFF
rtc_am_pm	RTC AM/PM value
rtc_display_format	RTC time notation

## Structure rtc\_alarm\_struct

**Table 3-393. Structure rtc\_alarm\_struct**

Member name	Function description
rtc_alarm_mask	RTC alarm mask
rtc_weekday_or_date	specify RTC alarm is on date or weekday
rtc_alarm_day	RTC alarm date or weekday value
rtc_alarm_hour	RTC alarm hour value
rtc_alarm_minute	RTC alarm minute value: 0x0 - 0x59(BCD format)

<code>rtc_alarm_second</code>	RTC alarm second value: 0x0 - 0x59(BCD format)
<code>rtc_am_pm</code>	RTC alarm AM/PM value

## Structure `rtc_timestamp_struct`

**Table 3-394. Structure `rtc_timestamp_struct`**

Member name	Function description
<code>rtc_timestamp_month</code>	RTC time-stamp month value
<code>rtc_timestamp_date</code>	RTC time-stamp date value: 0x1 - 0x31(BCD format)
<code>rtc_timestamp_day</code>	RTC time-stamp weekday value
<code>rtc_timestamp_hour</code>	RTC time-stamp hour value
<code>rtc_timestamp_minute</code>	RTC time-stamp minute value: 0x0 - 0x59(BCD format)
<code>rtc_timestamp_second</code>	RTC time-stamp second value: 0x0 - 0x59(BCD format)
<code>rtc_am_pm</code>	RTC time-stamp AM/PM value

## Structure `rtc_tamper_struct`

**Table 3-395. Structure `rtc_tamper_struct`**

Member name	Function description
<code>rtc_tamper_source</code>	RTC tamper source
<code>rtc_tamper_trigger</code>	RTC tamper trigger
<code>rtc_tamper_filter</code>	RTC tamper consecutive samples needed during a voltage level detection
<code>rtc_tamper_sample_frequency</code>	RTC tamper sampling frequency during a voltage level detection
<code>rtc_tamper_precharge_enable</code>	RTC tamper precharge feature during a voltage level detection
<code>rtc_tamper_precharge_time</code>	RTC tamper precharge duration if precharge feature is enabled
<code>rtc_tamper_with_timestamp</code>	RTC tamper time-stamp feature

## `rtc_deinit`

The description of `rtc_deinit` is shown as below:

**Table 3-396. Function `rtc_deinit`**

<b>Function name</b>	<code>rtc_deinit</code>
<b>Function prototype</b>	<code>ErrStatus rtc_deinit(void);</code>
<b>Function descriptions</b>	reset most of the RTC registers
<b>Precondition</b>	-
<b>The called functions</b>	<code>rcu_periph_reset_enable/ rcu_periph_reset_disable</code> -

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	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 3-397. Function rtc\_init**

Function name	rtc_init
Function prototype	ErrStatus rtc_init(rtc_parameter_struct* rtc_initpara_struct);
Function descriptions	initialize RTC registers
Precondition	-
Input parameter{in}	
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 3-392. Structure rtc_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* reset most of the RTC registers*/
ErrStatus error_status = rtc_init();
```

### rtc\_init\_mode\_enter

The description of rtc\_init\_mode\_enter is shown as below:

**Table 3-398. Function rtc\_init\_mode\_enter**

Function name	rtc_init_mode_enter
Function prototype	ErrStatus rtc_init_mode_enter(void);
Function descriptions	enter RTC init mode
Precondition	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
ErrStatus	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 3-399. Function rtc\_init\_mode\_exit**

Function name	rtc_init_mode_exit
Function prototype	void rtc_init_mode_exit(void);
Function descriptions	exit RTC init mode
Precondition	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

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 3-400. Function rtc\_register\_sync\_wait**

Function name	rtc_register_sync_wait
Function prototype	ErrStatus rtc_register_sync_wait(void);
Function descriptions	wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated
Precondition	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

<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 3-401. Function rtc\_current\_time\_get**

<b>Function name</b>	rtc_current_time_get
<b>Function prototype</b>	void rtc_current_time_get(rtc_parameter_struct* rtc_initpara_struct);
<b>Function descriptions</b>	get current time and date
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>rtc_initpara_struct</b>	pointer to a rtc_parameter_struct structure which contains parameters for initialization of the rtc peripheral, the structure members can refer to members of the structure <a href="#">Table 3-392. Structure rtc_parameter_struct</a>
<b>Return value</b>	
-	-

Example:

```
/*get current time and date*/
rtc_parameter_struct rtc_initpara_struct;
rtc_current_time_get (&rtc_initpara_struct);
```

### rtc\_subsecond\_get

The description of rtc\_subsecond\_get is shown as below:

**Table 3-402. Function rtc\_subsecond\_get**

<b>Function name</b>	rtc_subsecond_get
<b>Function prototype</b>	uint32_t rtc_subsecond_get(void);
<b>Function descriptions</b>	get current subsecond value
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	

-	-
<b>Return value</b>	
<b>uint32_t</b>	current subsecond value(0x00-0xFFFF)

Example:

```
/*get current subsecond value*/
uint32_t sub_second = rtc_subsecond_get();
```

### rtc\_alarm\_config

The description of `rtc_alarm_config` is shown as below:

**Table 3-403. Function `rtc_alarm_config`**

<b>Function name</b>	rtc_alarm_config
<b>Function prototype</b>	void rtc_alarm_config(rtc_alarm_struct* rtc_alarm_time)
<b>Function descriptions</b>	configure RTC alarm
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>rtc_alarm_time</b>	pointer to a <code>rtc_alarm_struct</code> structure which contains parameters for RTC alarm configuration, the structure members can refer to members of the structure <a href="#">Table 3-393. Structure <code>rtc_alarm_struct</code></a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*rtc_alarm_config*/
rtc_alarm_struct rtc_alarm_time;
rtc_alarm_config(&rtc_alarm_time);
```

### rtc\_alarm\_subsecond\_config

The description of `rtc_alarm_subsecond_config` is shown as below:

**Table 3-404. Function `rtc_alarm_subsecond_config`**

<b>Function name</b>	rtc_alarm_subsecond_config
<b>Function prototype</b>	void rtc_alarm_subsecond_config(uint32_t mask_subsecond, uint32_t subsecond);
<b>Function descriptions</b>	configure subsecond of RTC alarm
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>mask_subsecond</b>	alarm subsecond mask

<i>RTC_MASKSSC_0_14</i>	mask alarm subsecond configuration
<i>RTC_MASKSSC_1_14</i>	mask RTC_ALRM0SS_SSC[14:1], and RTC_ALRM0SS_SSC[0] is to be compared
<i>RTC_MASKSSC_2_14</i>	mask RTC_ALRM0SS_SSC[14:2], and RTC_ALRM0SS_SSC[1:0] is to be compared
<i>RTC_MASKSSC_3_14</i>	mask RTC_ALRM0SS_SSC[14:3], and RTC_ALRM0SS_SSC[2:0] is to be compared
<i>RTC_MASKSSC_4_14</i>	mask RTC_ALRM0SS_SSC[14:4], and RTC_ALRM0SS_SSC[3:0] is to be compared
<i>RTC_MASKSSC_5_14</i>	mask RTC_ALRM0SS_SSC[14:5], and RTC_ALRM0SS_SSC[4:0] is to be compared
<i>RTC_MASKSSC_6_14</i>	mask RTC_ALRM0SS_SSC[14:6], and RTC_ALRM0SS_SSC[5:0] is to be compared
<i>RTC_MASKSSC_7_14</i>	mask RTC_ALRM0SS_SSC[14:7], and RTC_ALRM0SS_SSC[6:0] is to be compared
<i>RTC_MASKSSC_8_14</i>	mask RTC_ALRM0SS_SSC[14:8], and RTC_ALRM0SS_SSC[7:0] is to be compared
<i>RTC_MASKSSC_9_14</i>	mask RTC_ALRM0SS_SSC[14:9], and RTC_ALRM0SS_SSC[8:0] is to be compared
<i>RTC_MASKSSC_10_14</i>	mask RTC_ALRM0SS_SSC[14:10], and RTC_ALRM0SS_SSC[9:0] is to be compared
<i>RTC_MASKSSC_11_14</i>	mask RTC_ALRM0SS_SSC[14:11], and RTC_ALRM0SS_SSC[10:0] is to be compared
<i>RTC_MASKSSC_12_14</i>	mask RTC_ALRM0SS_SSC[14:12], and RTC_ALRM0SS_SSC[11:0] is to be compared
<i>RTC_MASKSSC_13_14</i>	mask RTC_ALRM0SS_SSC[14:13], and RTC_ALRM0SS_SSC[12:0] is to be compared
<i>RTC_MASKSSC_NONE</i>	mask none, and RTC_ALRM0SS_SSC[14:0] is to be compared
<b>Input parameter{in}</b>	
<b>subsecond</b>	alarm subsecond value(0x000 - 0x7FFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*configure subsecond of RTC alarm*/
rtc_subsecond_config (RTC_MASKSSC_9_14, 0x7FFF);
```

### rtc\_alarm\_get

The description of rtc\_alarm\_get is shown as below:

**Table 3-405. Function rtc\_alarm\_get**

<b>Function name</b>	rtc_alarm_get
<b>Function prototype</b>	void rtc_alarm_get(rtc_alarm_struct* rtc_alarm_time);
<b>Function descriptions</b>	get RTC alarm
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>rtc_alarm_time</b>	pointer to a rtc_alarm_struct structure which contains parameters for RTC alarm configuration, the structure members can refer to members of the structure <a href="#">Table 3-393. Structure rtc_alarm_struct</a>
<b>Return value</b>	
-	-

Example:

```
rtc_alarm_struct rtc_alarm_time;
rtc_alarm_get (&rtc_alarm_time);
```

### rtc\_alarm\_subsecond\_get

The description of rtc\_alarm\_subsecond\_get is shown as below:

**Table 3-406. Function rtc\_alarm\_subsecond\_get**

<b>Function name</b>	rtc_alarm_subsecond_get
<b>Function prototype</b>	uint32_t rtc_alarm_subsecond_get(void);
<b>Function descriptions</b>	get RTC alarm subsecond
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	RTC alarm subsecond value(0x0-0x7FFF)

Example:

```
/*get RTC alarm subsecond*/
uint32_t subsecond = rtc_alarm_subsecond_get();
```

### **rtc\_alarm\_enable**

The description of rtc\_alarm\_enable is shown as below:

**Table 3-407. Function rtc\_alarm\_enable**

<b>Function name</b>	rtc_alarm_enable
<b>Function prototype</b>	void rtc_alarm_enable(void);
<b>Function descriptions</b>	enable RTC alarm
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*enable RTC alarm*/
rtc_alarm_enable();
```

### **rtc\_alarm\_disable**

The description of rtc\_alarm\_disable is shown as below:

**Table 3-408. Function rtc\_alarm\_disable**

<b>Function name</b>	rtc_alarm_disable
<b>Function prototype</b>	ErrStatus rtc_alarm_disable(void);
<b>Function descriptions</b>	disable RTC alarm
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/*disable RTC alarm*/
ErrStatus error_status = rtc_alarm_disable();
```

### **rtc\_timestamp\_enable**

The description of rtc\_timestamp\_enable is shown as below:

**Table 3-409. Function rtc\_timestamp\_enable**

<b>Function name</b>	rtc_timestamp_enable
<b>Function prototype</b>	void rtc_timestamp_enable(uint32_t edge);
<b>Function descriptions</b>	enable RTC time-stamp
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>edge</b>	specify which edge to detect of time-stamp
<i>RTC_TIMESTAMP_RISING_EDGE</i>	rising edge is valid event edge for timestamp event
<i>RTC_TIMESTAMP_FALLING_EDGE</i>	falling edge is valid event edge for timestamp event
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*enable RTC time-stamp*/
rtc_timestamp_enable (RTC_TIMESTAMP_RISING_EDGE);
```

### **rtc\_timestamp\_disable**

The description of rtc\_timestamp\_disable is shown as below:

**Table 3-410. Function rtc\_timestamp\_disable**

<b>Function name</b>	rtc_timestamp_disable
<b>Function prototype</b>	void rtc_timestamp_disable(void);
<b>Function descriptions</b>	disable RTC time-stamp
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*disable RTC time-stamp*/
rtc_timestamp_disable ();
```

### **rtc\_timestamp\_get**

The description of rtc\_timestamp\_get is shown as below:

**Table 3-411. Function rtc\_timestamp\_get**

<b>Function name</b>	rtc_timestamp_get
<b>Function prototype</b>	void rtc_timestamp_get(rtc_timestamp_struct* rtc_timestamp);
<b>Function descriptions</b>	get RTC timestamp time and date
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
rtc_timestamp	Pointer to a rtc_timestamp_struct structure which contains parameters for RTC time-stamp configuration, the structure members can refer to members of the structure <a href="#">Table 3-395. Structure rtc_tamper_struct</a>
<b>Return value</b>	
-	-

Example:

```
/* get RTC timestamp time and date */

rtc_timestamp_struct rtc_timestamp;

rtc_timestamp_get(& rtc_timestamp);
```

### rtc\_timestamp\_subsecond\_get

The description of rtc\_timestamp\_subsecond\_get is shown as below:

**Table 3-412. Function rtc\_timestamp\_subsecond\_get**

<b>Function name</b>	rtc_timestamp_subsecond_get
<b>Function prototype</b>	uint32_t rtc_timestamp_subsecond_get(void);
<b>Function descriptions</b>	get RTC time-stamp subsecond
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	RTC time-stamp subsecond value

Example:

```
/* get RTC time-stamp subsecond */

uint32_t subsecond = rtc_timestamp_subsecond_get();
```

### rtc\_tamper\_enable

The description of rtc\_tamper\_enable is shown as below:

**Table 3-413. Function rtc\_tamper\_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>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 3-395. 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 3-414. 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>Input parameter{in}</b>	
source	specify which tamper source to be disabled
RTC_TAMPER0	RTC tamper0
RTC_TAMPER1	RTC tamper1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable RTC tamper */

rtc_tamper_disable(RTC_TAMPER0);
```

### rtc\_interrupt\_enable

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

**Table 3-415. Function rtc\_interrupt\_enable**

<b>Function name</b>	rtc_interrupt_enable
<b>Function prototype</b>	void rtc_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable specified RTC interrupt
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which interrupt source to be enabled
<i>RTC_INT_TIMESTAMP</i>	timestamp interrupt
<i>RTC_INT_ALARM</i>	alarm interrupt
<i>RTC_INT_TAMP</i>	tamp interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable specified RTC interrupt*/
rtc_interrupt_enable(RTC_INT_TAMP);
```

### rtc\_interrupt\_disable

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

**Table 3-416. Function rtc\_interrupt\_disable**

<b>Function name</b>	rtc_interrupt_disable
<b>Function prototype</b>	void rtc_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable specified RTC interrupt
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which RTC interrupt to disable
<i>RTC_INT_TIMESTAMP</i>	second interrupt
<i>RTC_INT_ALARM</i>	alarm interrupt
<i>RTC_INT_TAMP</i>	tamp interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable specified RTC interrupt */
```

```
rtc_interrupt_disable(RTC_INT_TAMP);
```

## rtc\_flag\_get

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

**Table 3-417. Function rtc\_flag\_get**

<b>Function name</b>	rtc_flag_get
<b>Function prototype</b>	FlagStatus rtc_flag_get(uint32_t flag);
<b>Function descriptions</b>	check specified flag
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag to check
<i>RTC_FLAG_RECALIBRATION</i>	recalibration pending flag
<i>RTC_FLAG_TAMP1</i>	tamper 1 event flag
<i>RTC_FLAG_TAMP0</i>	tamper 0 event flag
<i>RTC_FLAG_TIMESTAMPOVERFLOW</i>	time-stamp overflow event flag
<i>RTC_FLAG_TIMESTAMP</i>	time-stamp event flag
<i>RTC_FLAG_ALARM0</i>	alarm event flag
<i>RTC_FLAG_INIT</i>	init mode event flag
<i>RTC_FLAG_RSYN</i>	time and date registers synchronized event flag
<i>RTC_FLAG_YCM</i>	year parameter configured event flag
<i>RTC_FLAG_SHIFT</i>	shift operation pending flag
<i>RTC_FLAG_ALARM0WRITTEN</i>	alarm written available 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_TIMESTAMP);
```

## rtc\_flag\_clear

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

**Table 3-418. 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>Input parameter{in}</b>	
<b>flag</b>	specify which flag to clear
<i>RTC_FLAG_TAMP1</i>	tamper 1 event flag
<i>RTC_FLAG_TAMP0</i>	tamper 0 event flag
<i>RTC_FLAG_TIMESTAMP_MP_OVERFLOW</i>	time-stamp overflow event flag
<i>RTC_FLAG_TIMESTAMP_MP</i>	time-stamp event flag
<i>RTC_FLAG_ALARM0</i>	alarm 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_TIMESTAMP);
```

### rtc\_alter\_output\_config

The description of `rtc_alter_output_config` is shown as below:

**Table 3-419. Function `rtc_alter_output_config`**

<b>Function name</b>	rtc_alter_output_config
<b>Function prototype</b>	void rtc_alter_output_config(uint32_t source, uint32_t mode);
<b>Function descriptions</b>	configure rtc alternate output source
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	specify signal to output
<i>RTC_CALIBRATION_512HZ</i>	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 512Hz signal
<i>RTC_CALIBRATION_1HZ</i>	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 1Hz signal
<i>RTC_ALARM_HIGH</i>	when the alarm flag is set, the output pin is high
<i>RTC_ALARM_LOW</i>	when the Alarm flag is set, the output pin is low
<b>Input parameter{in}</b>	
<b>mode</b>	specify the output pin (PC13) mode when output alarm signal
<i>RTC_ALARM_OUTPUT_OD</i>	open drain mode
<i>RTC_ALARM_OUTPUT_PP</i>	push pull mode

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

Example:

```
/* configure rtc alternate output source */
rtc_alter_output_config(RTC_ALARM_LOW, RTC_ALARM_OUTPUT_PP);
```

### rtc\_calibration\_config

The description of `rtc_calibration_config` is shown as below:

**Table 3-420. Function `rtc_calibration_config`**

<b>Function name</b>	<code>rtc_calibration_config</code>
<b>Function prototype</b>	<code>ErrStatus rtc_calibration_config(uint32_t window, uint32_t plus, uint32_t minus);</code>
<b>Function descriptions</b>	configure RTC calibration register
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>window</b>	select calibration window
<code>RTC_CALIBRATION_WINDOW_32S</code>	2exp20 RTCCLK cycles, 32s if RTCCLK = 32768 Hz
<code>RTC_CALIBRATION_WINDOW_16S</code>	2exp19 RTCCLK cycles, 16s if RTCCLK = 32768 Hz
<code>RTC_CALIBRATION_WINDOW_8S</code>	2exp18 RTCCLK cycles, 8s if RTCCLK = 32768 Hz
<b>Input parameter{in}</b>	
<b>plus</b>	add RTC clock or not
<code>RTC_CALIBRATION_P_LUS_SET</code>	add one RTC clock every 2048 rtc clock
<code>RTC_CALIBRATION_P_LUS_RESET</code>	no effect
<b>Input parameter{in}</b>	
<b>minus</b>	the RTC clock to minus during the calibration window(0x0 - 0x1FF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* configure RTC calibration register*/
```

---

```
ErrStatus error_status = rtc_calibration_config( RTC_CALIBRATION_WINDOW_32S,
    RTC_CALIBRATION_PLUS_SET, 0xFF);
```

## **rtc\_hour\_adjust**

The description of `rtc_hour_adjust` is shown as below:

**Table 3-421. Function `rtc_hour_adjust`**

<b>Function name</b>	rtc_hour_adjust
<b>Function prototype</b>	void rtc_hour_adjust(uint32_t operation);
<b>Function descriptions</b>	adjust the daylight saving time by adding or subtracting one hour from the current time
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>operation</b>	hour adjustment operation
<i>RTC_CTL_A1H</i>	add one hour
<i>RTC_CTL_S1H</i>	subtract one hour
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* adjust the daylight saving time by adding one hour from the current time */
rtc_hour_adjust(RTC_CTL_A1H);
```

## **rtc\_second\_adjust**

The description of `rtc_second_adjust` is shown as below:

**Table 3-422. Function `rtc_second_adjust`**

<b>Function name</b>	rtc_second_adjust
<b>Function prototype</b>	ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);
<b>Function descriptions</b>	adjust RTC second or subsecond value of current time
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>add</b>	add 1s to current time or not
<i>RTC_SHIFT_ADD1S_R ESET</i>	no effect
<i>RTC_SHIFT_ADD1S_S ET</i>	add 1s to current time
<b>Input parameter{in}</b>	
<b>minus</b>	number of subsecond to minus from current time(0x0 - 0x7FFF)
<b>Output parameter{out}</b>	

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

Example:

```
/* adjust RTC second or subsecond value of current time */

ErrStatus error_status = rtc_second_adjust(RTC_SHIFT_ADD1S_SET, 0);
```

### **rtc\_bypass\_shadow\_enable**

The description of `rtc_bypass_shadow_enable` is shown as below:

**Table 3-423. Function `rtc_bypass_shadow_enable`**

<b>Function name</b>	rtc_bypass_shadow_enable
<b>Function prototype</b>	void rtc_bypass_shadow_enable(void);
<b>Function descriptions</b>	enable RTC bypass shadow registers function
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable RTC bypass shadow registers function*/

rtc_bypass_shadow_enable();
```

### **rtc\_bypass\_shadow\_disable**

The description of `rtc_bypass_shadow_disable` is shown as below:

**Table 3-424. Function `rtc_bypass_shadow_disable`**

<b>Function name</b>	rtc_bypass_shadow_disable
<b>Function prototype</b>	void rtc_bypass_shadow_disable (void);
<b>Function descriptions</b>	disable RTC bypass shadow registers function
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable RTC bypass shadow registers function*/
rtc_bypass_shadow_disable();
```

### **rtc\_refclock\_detection\_enable**

The description of `rtc_refclock_detection_enable` shown as below:

**Table 3-425. 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>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 3-426. 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>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* disableRTC reference clock detection function*/
ErrStatus error_status = rtc_refclock_detection_disable();;
```

## 3.19. SPI/I2S

The SPI/I2S module can communicate with external devices using the SPI protocol or the I2S audio protocol. The SPI/I2S registers are listed in chapter [3.19.1](#), the SPI/I2S firmware functions are introduced in chapter [3.19.2](#).

### 3.19.1. Descriptions of Peripheral registers

SPI/I2S registers are listed in the table shown as below:

**Table 3-427. SPI/I2S registers**

Registers	Descriptions
SPI_CTL0	SPI control register 0
SPI_CTL1	SPI control register 1
SPI_STAT	SPI status register
SPI_DATA	SPI data register
SPI_CRCPOLY	SPI CRC polynomial register
SPI_RCRC	SPI receive CRC register
SPI_TCRC	SPI transmit CRC register
SPI_I2SCTL	SPI/I2S control register
SPI_I2SPSC	SPI/I2S clock prescaler register
SPI_QCTL	SPI quad mode control register

### 3.19.2. Descriptions of Peripheral functions

SPI/I2S firmware functions are listed in the table shown as below:

**Table 3-428. SPI/I2S firmware function**

Function name	Function description
spi_i2s_deinit	reset SPI and I2S peripheral
spi_struct_para_init	initialize the parameters of SPI struct with the default values
spi_init	initialize SPI peripheral parameter
spi_enable	enable SPI
spi_disable	disable SPI
i2s_init	initialize I2S peripheral parameter
i2s_psc_config	configure I2S peripheral prescaler
i2s_enable	enable I2S
i2s_disable	disable I2S
spi_nss_output_enable	enable SPI NSS output function
spi_nss_output_disable	disable SPI NSS output function
spi_nss_internal_high	SPI NSS pin high level in software mode
spi_nss_internal_low	SPI NSS pin low level in software mode
spi_dma_enable	enable SPI DMA function

Function name	Function description
spi_dma_disable	disable SPI DMA function
spi_i2s_data_frame_format_config	configure SPI/I2S data frame format
spi_i2s_data_transmit	SPI transmit data
spi_i2s_data_receive	SPI receive data
spi_bidirectional_transfer_config	configure SPI bidirectional transfer direction
spi_crc_polynomial_set	set SPI CRC polynomial
spi_crc_polynomial_get	get SPI CRC polynomial
spi_crc_on	turn on SPI CRC function
spi_crc_off	turn off SPI CRC function
spi_crc_next	SPI next data is CRC value
spi_crc_get	get SPI CRC send value or receive value
spi_ti_mode_enable	enable SPI TI mode
spi_ti_mode_disable	disable SPI TI mode
spi_nssp_mode_enable	enable SPI NSS pulse mode
spi_nssp_mode_disable	disable SPI NSS pulse mode
qspi_enable	enable quad wire SPI
qspi_disable	disable quad wire SPI
qspi_write_enable	enable quad wire SPI write
qspi_read_enable	enable quad wire SPI read
qspi_io23_output_enable	enable quad wire SPI_IO2 and SPI_IO3 pin output
qspi_io23_output_disable	disable quad wire SPI_IO2 and SPI_IO3 pin output
spi_i2s_interrupt_enable	enable SPI and I2S interrupt
spi_i2s_interrupt_disable	disable SPI and I2S interrupt
spi_i2s_interrupt_flag_get	get SPI and I2S interrupt status
spi_i2s_flag_get	get SPI and I2S flag status
spi_crc_error_clear	clear SPI CRC error flag status

## Structure spi\_parameter\_struct

Table 3-429. Structure spi\_parameter\_struct

Member name	Function description
device_mode	SPI master or slave (SPI_MASTER, SPI_SLAVE)
trans_mode	SPI transtype (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	SPI frame size (SPI_FRAMESIZE_8BIT, SPI_FRAMESIZE_16BIT)
nss	SPI NSS control by hardware or software (SPI_NSS_SOFT, SPI_NSS_HARD)
endian	SPI big endian or little endian (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)

clock_polarity_phase_e	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 3-430. Function spi\_i2s\_deinit**

<b>Function name</b>	spi_i2s_deinit
<b>Function prototype</b>	void spi_i2s_deinit(uint32_t spi_periph);
<b>Function descriptions</b>	Reset SPIx and I2Sx peripheral
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
spi_periph	SPI/I2S peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset SPI0 */
spi_i2s_deinit(SPI0);
```

### spi\_struct\_para\_init

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

**Table 3-431. Function spi\_struct\_para\_init**

<b>Function name</b>	spi_struct_para_init
<b>Function prototype</b>	void spi_struct_para_init(spi_parameter_struct* spi_struct);
<b>Function descriptions</b>	Initialize the parameters of SPI struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_struct	SPI init parameter struct, the structure members can refer to <a href="#">Table 3-429.</a> <a href="#">Structure spi_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* initialize the parameters of SPI */
spi_parameter_struct spi_init_struct;
spi_struct_para_init(&spi_init_struct);
```

### spi\_init

The description of spi\_init is shown as below:

**Table 3-432. Function spi\_init**

Function name	spi_init
Function prototype	void spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);
Function descriptions	Initialize SPIx peripheral parameter
Precondition	-
The called functions	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Input parameter{in}</b>	
spi_struct	SPI parameter initialization stuct, the structure members can refer to members of the structure <a href="#">Table 3-429. Structure spi_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize SPI0 */
spi_parameter_struct spi_init_struct;
spi_init_struct.trans_mode = SPI_TRANSMODE_BDTRANSMIT;
spi_init_struct.device_mode = SPI_MASTER;
spi_init_struct.frame_size = SPI_FRAMESIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_CK_PL_HIGH_PH_2EDGE;
spi_init_struct.nss = SPI_NSS_SOFT;
spi_init_struct.prescale = SPI_PSC_8;
```

---

```

spi_init_struct.endian          = SPI_ENDIAN_MSB;

spi_init(SPI0, &spi_init_struct);

```

## **spi\_enable**

The description of spi\_enable is shown as below:

**Table 3-433. Function spi\_enable**

<b>Function name</b>	spi_enable
<b>Function prototype</b>	void spi_enable(uint32_t spi_periph);
<b>Function descriptions</b>	Enable SPIx
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* enable SPI0 */

spi_enable(SPI0);

```

## **spi\_disable**

The description of spi\_disable is shown as below:

**Table 3-434. Function spi\_disable**

<b>Function name</b>	spi_disable
<b>Function prototype</b>	void spi_disable(uint32_t spi_periph);
<b>Function descriptions</b>	Disable SPIx
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 */
```

```
spi_disable(SPI0);
```

### i2s\_init

The description of i2s\_init is shown as below:

**Table 3-435. Function i2s\_init**

<b>Function name</b>	i2s_init
<b>Function prototype</b>	void i2s_init(uint32_t spi_periph, uint32_t mode, uint32_t standard, uint32_t ckpl);
<b>Function descriptions</b>	Initialize I2S0 peripheral parameter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S0 peripheral
<b>SPIx</b>	x=0
<b>Input parameter{in}</b>	
<b>mode</b>	I2S operation mode
<b>I2S_MODE_SLAVETX</b>	I2S slave transmit mode
<b>I2S_MODE_SLAVERX</b>	I2S slave receive mode
<b>I2S_MODE_MASTERTX</b>	I2S master transmit mode
<b>I2S_MODE_MASTERRX</b>	I2S master receive mode
<b>Input parameter{in}</b>	
<b>standard</b>	I2S standard
<b>I2S_STD_PHILLIPS</b>	I2S phillips standard
<b>I2S_STD_MSB</b>	I2S MSB standard
<b>I2S_STD_LSB</b>	I2S LSB standard
<b>I2S_STD_PCMSHORT</b>	I2S PCM short standard
<b>I2S_STD_PCMLONG</b>	I2S PCM long standard
<b>Input parameter{in}</b>	
<b>ckpl</b>	I2S idle state clock polarity
<b>I2S_CKPL_LOW</b>	I2S clock polarity low level
<b>I2S_CKPL_HIGH</b>	I2S clock polarity high level
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize I2S0 */
```

---

```
i2s_init(SPI0, I2S_MODE_MASTERTX, I2S_STD_PHILLIPS, I2S_CKPL_LOW);
```

## i2s\_psc\_config

The description of i2s\_psc\_config is shown as below:

**Table 3-436. Function i2s\_psc\_config**

<b>Function name</b>	i2s_psc_config
<b>Function prototype</b>	void i2s_psc_config(uint32_t spi_periph, uint32_t audiosample, uint32_t frameformat, uint32_t mckout);
<b>Function descriptions</b>	Configure I2S0 prescaler
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S0 peripheral
<i>SPIx</i>	x=0
<b>Input parameter{in}</b>	
<b>audiosample</b>	I2S audio sample rate
<i>I2S_AUDIOSAMPLE_8K</i>	audio sample rate is 8KHz
<i>I2S_AUDIOSAMPLE_11K</i>	audio sample rate is 11KHz
<i>I2S_AUDIOSAMPLE_16K</i>	audio sample rate is 16KHz
<i>I2S_AUDIOSAMPLE_22K</i>	audio sample rate is 22KHz
<i>I2S_AUDIOSAMPLE_32K</i>	audio sample rate is 32KHz
<i>I2S_AUDIOSAMPLE_44K</i>	audio sample rate is 44KHz
<i>I2S_AUDIOSAMPLE_48K</i>	audio sample rate is 48KHz
<i>I2S_AUDIOSAMPLE_96K</i>	audio sample rate is 96KHz
<i>I2S_AUDIOSAMPLE_192K</i>	audio sample rate is 192KHz
<b>Input parameter{in}</b>	
<b>frameformat</b>	I2S data length and channel length
<i>I2S_FRAMEFORMAT_DT16B_CH16B</i>	I2S data length is 16 bit and channel length is 16 bit
<i>I2S_FRAMEFORMAT_DT16B_CH32B</i>	I2S data length is 16 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT24B_CH32B</i>	I2S data length is 24 bit and channel length is 32 bit

<i>I2S_FRAMEFORMAT_DT32B_CH32B</i>	I2S data length is 32 bit and channel length is 32 bit
<b>Input parameter{in}</b>	
<b>mckout</b>	I2S master clock output
<i>I2S_MCKOUT_ENABLE</i>	I2S master clock output enable
<i>I2S_MCKOUT_DISABLE</i>	I2S master clock output disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure I2S0 prescaler */

i2s_psc_config(SPI0, I2S_AUDIOSAMPLE_44K, I2S_FRAMEFORMAT_DT16B_CH16B,
I2S_MCKOUT_DISABLE);
```

### i2s\_enable

The description of i2s\_enable is shown as below:

**Table 3-437. Function i2s\_enable**

<b>Function name</b>	i2s_enable
<b>Function prototype</b>	void i2s_enable(uint32_t spi_periph);
<b>Function descriptions</b>	Enable I2S0
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S0 peripheral
<b>SPIx</b>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2S0*/

i2s_enable(SPI0);
```

### i2s\_disable

The description of i2s\_disable is shown as below:

**Table 3-438. Function i2s\_disable**

<b>Function name</b>	i2s_disable
<b>Function prototype</b>	void i2s_disable(uint32_t spi_periph);
<b>Function descriptions</b>	Disable I2S0
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	I2S0 peripheral
SPIx	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2S0*/
i2s_disable(SPI0);
```

### spi\_nss\_output\_enable

The description of spi\_nss\_output\_enable is shown as below:

**Table 3-439. Function spi\_nss\_output\_enable**

<b>Function name</b>	spi_nss_output_enable
<b>Function prototype</b>	void spi_nss_output_enable(uint32_t spi_periph);
<b>Function descriptions</b>	Enable SPIx NSS output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPIx peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 NSS output */
spi_nss_output_enable(SPI0);
```

### spi\_nss\_output\_disable

The description of spi\_nss\_output\_disable is shown as below:

**Table 3-440. Function spi\_nss\_output\_disable**

<b>Function name</b>	spi_nss_output_disable
<b>Function prototype</b>	void spi_nss_output_disable(uint32_t spi_periph);
<b>Function descriptions</b>	Disable SPIx NSS output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPIx peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 NSS output */

spi_nss_output_disable(SPI0);
```

### spi\_nss\_internal\_high

The description of spi\_nss\_internal\_high is shown as below:

**Table 3-441. Function spi\_nss\_internal\_high**

<b>Function name</b>	spi_nss_internal_high
<b>Function prototype</b>	void spi_nss_internal_high(uint32_t spi_periph);
<b>Function descriptions</b>	SPI NSS pin high level in software mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 NSS pin is pulled high level in software mode */

spi_nss_internal_high(SPI0);
```

### spi\_nss\_internal\_low

The description of spi\_nss\_internal\_low is shown as below:

**Table 3-442. Function spi\_nss\_internal\_low**

<b>Function name</b>	spi_nss_internal_low
<b>Function prototype</b>	void spi_nss_internal_low(uint32_t spi_periph);
<b>Function descriptions</b>	SPI NSS pin low level in software mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 NSS pin is pulled low level in software mode */
spi_nss_internal_low(SPI0);
```

### spi\_dma\_enable

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

**Table 3-443. Function spi\_dma\_enable**

<b>Function name</b>	spi_dma_enable
<b>Function prototype</b>	void spi_dma_enable(uint32_t spi_periph, uint8_t dma);
<b>Function descriptions</b>	Enable SPIx DMA function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Input parameter{in}</b>	
dma	SPI DMA mode
SPI_DMA_TRANSMIT	SPI transmit data use DMA
SPI_DMA_RECEIVE	SPI receive data use DMA
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 transmit data DMA function */
spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);
```

### spi\_dma\_disable

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

**Table 3-444. Function spi\_dma\_disable**

<b>Function name</b>	spi_dma_disable
<b>Function prototype</b>	void spi_dma_disable(uint32_t spi_periph, uint8_t dma);
<b>Function descriptions</b>	Disable SPIx DMA function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Input parameter{in}</b>	
dma	SPI DMA mode
SPI_DMA_TRANSMIT	SPI transmit data use DMA
SPI_DMA_RECEIVE	SPI receive data use DMA
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 transmit data DMA function */
spi_dma_disable(SPI0, SPI_DMA_TRANSMIT);
```

### spi\_i2s\_data\_frame\_format\_config

The description of spi\_i2s\_data\_frame\_format\_config is shown as below:

**Table 3-445. Function spi\_i2s\_data\_frame\_format\_config**

<b>Function name</b>	spi_i2s_data_frame_format_config
<b>Function prototype</b>	void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
<b>Function descriptions</b>	Configure SPIx/I2S0 data frame format
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Input parameter{in}</b>	
frame_format	SPI frame size
SPI_FRAMESIZE_8BIT	SPI frame size is 8 bits
SPI_FRAMESIZE_16BIT	SPI frame size is 16 bits

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

Example:

```
/* configure SPI0/I2S0 data frame format size is 16 bits */
spi_i2s_data_frame_format_config(SPI0, SPI_FRAMESIZE_16BIT);
```

### **spi\_i2s\_data\_transmit**

The description of spi\_i2s\_data\_transmit is shown as below:

**Table 3-446. Function spi\_i2s\_data\_transmit**

<b>Function name</b>	spi_i2s_data_transmit
<b>Function prototype</b>	void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);
<b>Function descriptions</b>	SPI transmit data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Input parameter{in}</b>	
data	16-bit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 transmit data */

spi_i2s_data_transmit(SPI0, spi0_send_array[send_n]);
```

### **spi\_i2s\_data\_receive**

The description of spi\_i2s\_data\_receive is shown as below:

**Table 3-447. Function spi\_i2s\_data\_receive**

<b>Function name</b>	spi_i2s_data_receive
<b>Function prototype</b>	uint16_t spi_i2s_data_receive(uint32_t spi_periph);
<b>Function descriptions</b>	SPI receive data
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1
Output parameter{out}	
-	-
Return value	
<b>uint16_t</b>	16-bit data

Example:

```
/* SPI0 receive data */

spi0_receive_array[receive_n] = spi_i2s_data_receive(SPI0);
```

### **spi\_bidirectional\_transfer\_config**

The description of **spi\_bidirectional\_transfer\_config** is shown as below:

**Table 3-448. Function spi\_bidirectional\_transfer\_config**

<b>Function name</b>	spi_bidirectional_transfer_config
<b>Function prototype</b>	void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);
<b>Function descriptions</b>	Configure SPIx bidirectional transfer direction
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1
Input parameter{in}	
<b>transfer_direction</b>	SPI transfer direction
<b>SPI_BIDIRECTIONAL_TRANSMIT</b>	SPI work in transmit-only mode
<b>SPI_BIDIRECTIONAL_RECEIVE</b>	SPI work in receive-only mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI0 works in transmit-only mode */

spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

### spi\_crc\_polynomial\_set

The description of spi\_crc\_polynomial\_set is shown as below:

**Table 3-449. 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>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Input parameter{in}</b>	
crc_poly	CRC polynomial value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set SPI0 CRC polynomial */
spi_crc_polynomial_set(SPI0, CRC_VALUE);
```

### spi\_crc\_polynomial\_get

The description of spi\_crc\_polynomial\_get is shown as below:

**Table 3-450. Function spi\_crc\_polynomial\_get**

<b>Function name</b>	spi_crc_polynomial_get
<b>Function prototype</b>	uint16_t spi_crc_polynomial_get(uint32_t spi_periph);
<b>Function descriptions</b>	Get SPI CRC polynomial
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	16 bit CRC polynomial value

Example:

```
/* get SPI0 CRC polynomial */
```

```

uint16_t crc_val;

crc_val = spi_crc_polynomial_get(SPI0);

```

### **spi\_crc\_on**

The description of spi\_crc\_on is shown as below:

**Table 3-451. Function spi\_crc\_on**

<b>Function name</b>	spi_crc_on
<b>Function prototype</b>	void spi_crc_on(uint32_t spi_periph);
<b>Function descriptions</b>	Turn on CRC function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<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 3-452. Function spi\_crc\_off**

<b>Function name</b>	spi_crc_off
<b>Function prototype</b>	void spi_crc_off(uint32_t spi_periph);
<b>Function descriptions</b>	Turn off CRC function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

```
/* turn off SPI0 CRC function */
```

```
spi_crc_off(SPI0);
```

### **spi\_crc\_next**

The description of spi\_crc\_next is shown as below:

**Table 3-453. Function spi\_crc\_next**

<b>Function name</b>	spi_crc_next
<b>Function prototype</b>	void spi_crc_next(uint32_t spi_periph);
<b>Function descriptions</b>	SPI next data is CRC value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<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 3-454. 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>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Input parameter{in}</b>	
crc	SPI crc value
SPI_CRC_TX	get transmit crc value
SPI_CRC_RX	get receive crc value
<b>Output parameter{out}</b>	
-	-

Return value	
uint16_t	16-bit CRC value

Example:

```
/* get SPI0 CRC send value */

uint16_t crc_val;

crc_val = spi_crc_get(SPI0,SPI_CRC_TX);
```

### spi\_ti\_mode\_enable

The description of spi\_ti\_mode\_enable is shown as below:

**Table 3-455. 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>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* enable SPI0 TI mode */

spi_ti_mode_enable(SPI0);
```

### spi\_ti\_mode\_disable

The description of spi\_ti\_mode\_disable is shown as below:

**Table 3-456. 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>	
spi_periph	SPI peripheral
SPIx	x=0,1

Output parameter{out}	
-	-
Return value	

Example:

```
/* disable SPI0 TI mode */
spi_ti_mode_disable(SPI0);
```

### spi\_nssp\_mode\_enable

The description of spi\_nssp\_mode\_enable is shown as below:

**Table 3-457. Function spi\_nssp\_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 NSS pulse mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1
Output parameter{out}	
-	-
Return value	

Example:

```
/* enable SPI0 NSS pulse mode */
spi_nssp_mode_enable(SPI0);
```

### spi\_nssp\_mode\_disable

The description of spi\_nssp\_mode\_disable is shown as below:

**Table 3-458. Function spi\_nssp\_mode\_disable**

<b>Function name</b>	spi_ti_mode_disable
<b>Function prototype</b>	void spi_ti_mode_disable(uint32_t spi_periph);
<b>Function descriptions</b>	Disable SPI NSS pulse mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1

Output parameter{out}	
-	-
Return value	

Example:

```
/* disable SPI0 NSS pulse mode */

spi_nssp_mode_disable(SPI0);
```

### qspi\_enable

The description of qspi\_enable is shown as below:

**Table 3-459. Function qspi\_enable**

Function name	qspi_enable
Function prototype	void qspi_enable(uint32_t spi_periph);
Function descriptions	Enable quad wire SPI
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=1
Output parameter{out}	
-	-
Return value	

Example:

```
/* enable SPI1 quad wire mode */

qspi_enable(SPI1);
```

### qspi\_disable

The description of qspi\_disable is shown as below:

**Table 3-460. Function qspi\_disable**

Function name	qspi_disable
Function prototype	void qspi_disable(uint32_t spi_periph);
Function descriptions	Disable quad wire SPI
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=1

Output parameter{out}	
-	-
Return value	

Example:

```
/* disable SPI1 quad wire mode */

qspi_disable(SPI1);
```

### **qspi\_write\_enable**

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

**Table 3-461. Function qspi\_write\_enable**

<b>Function name</b>	qspi_write_enable
<b>Function prototype</b>	void qspi_write_enable(uint32_t spi_periph);
<b>Function descriptions</b>	Enable quad wire SPI write
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=1
Output parameter{out}	
-	-
Return value	

Example:

```
/* enable SPI1 quad wire write */

qspi_write_enable(SPI1);
```

### **qspi\_read\_enable**

The description of qspi\_read\_enable is shown as below:

**Table 3-462. Function qspi\_read\_enable**

<b>Function name</b>	qspi_read_enable
<b>Function prototype</b>	void qspi_read_enable(uint32_t spi_periph);
<b>Function descriptions</b>	Enable quad wire SPI read
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=1
Output parameter{out}	
-	-
Return value	

Example:

```
/* enable SPI1 quad wire read */
qspi_read_enable(SPI1);
```

### **qspi\_io23\_output\_enable**

The description of qspi\_io23\_output\_enable is shown as below:

**Table 3-463. Function qspi\_io23\_output\_enable**

Function name	qspi_io23_output_enable	
Function prototype	void qspi_io23_output_enable(uint32_t spi_periph);	
Function descriptions	Enable SPI_IO2 and SPI_IO3 pin output	
Precondition	-	
The called functions	-	
Input parameter{in}		
spi_periph	SPI peripheral	
SPIx	x=1	
Output parameter{out}		
-	-	
Return value		

Example:

```
/* enable SPI1 SPI_IO2 and SPI_IO3 pin output */
qspi_io23_output_enable(SPI1);
```

### **qspi\_io23\_output\_disable**

The description of qspi\_io23\_output\_disable is shown as below:

**Table 3-464. Function qspi\_io23\_output\_disable**

Function name	qspi_io23_output_disable	
Function prototype	void qspi_io23_output_disable(uint32_t spi_periph);	
Function descriptions	Disable SPI_IO2 and SPI_IO3 pin output	

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* disable SPI1 SPI_IO2 and SPI_IO3 pin output */
qspi_io23_output_disable(SPI1);
```

### **spi\_i2s\_interrupt\_enable**

The description of spi\_i2s\_interrupt\_enable is shown as below:

**Table 3-465. Function spi\_i2s\_interrupt\_enable**

<b>Function name</b>	spi_i2s_interrupt_enable
<b>Function prototype</b>	void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	Enable SPIx and I2S0 interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>interrupt</b>	SPI/I2S interrupt
<b>SPI_I2SINT_TBE</b>	transmit buffer empty interrupt
<b>SPI_I2S_INT_RBNE</b>	receive buffer not empty interrupt
<b>SPI_I2S_INT_ERR</b>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* 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 3-466. Function spi\_i2s\_interrupt\_disable**

<b>Function name</b>	spi_i2s_interrupt_disable
<b>Function prototype</b>	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	Disable SPIx and I2S0 interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Input parameter{in}</b>	
interrupt	SPI/I2S interrupt
SPI_I2SINT_TBE	transmit buffer empty interrupt
SPI_I2S_INT_RBNE	receive buffer not empty interrupt
SPI_I2S_INT_ERR	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* 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 3-467. Function spi\_i2s\_interrupt\_flag\_get**

<b>Function name</b>	spi_i2s_interrupt_flag_get
<b>Function prototype</b>	FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	Get SPIx and I2S0 interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Input parameter{in}</b>	
interrupt	SPI/I2S interrupt
SPI_I2S_INT_FLAG_T	transmit buffer empty interrupt

<i>BE</i>	
<i>SPI_I2S_INT_FLAG_R</i> <i>BNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_FLAG_R</i> <i>XORERR</i>	overrun interrupt
<i>SPI_INT_FLAG_CONF</i> <i>ERR</i>	config error interrupt
<i>SPI_INT_FLAG_CRCE</i> <i>RR</i>	CRC error interrupt
<i>I2S_INT_FLAG_TXUR</i> <i>ERR</i>	underrun error interrupt
<i>SPI_I2S_INT_FLAG_F</i> <i>ERR</i>	format error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get SPI0 transmit buffer empty interrupt status */

If(RESET != spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE)){
    while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
    spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
}
```

### spi\_i2s\_flag\_get

The description of spi\_i2s\_flag\_get is shown as below:

**Table 3-468. Function spi\_i2s\_flag\_get**

<b>Function name</b>	spi_i2s_flag_get
<b>Function prototype</b>	FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	Get SPIx and I2S0 flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>flag</b>	SPI/I2S flag status
<b>SPI_FLAG_TBE</b>	transmit buffer empty flag
<b>SPI_FLAG_RBNE</b>	receive buffer not empty flag

<i>SPI_FLAG_TRANS</i>	transmit on-going flag
<i>SPI_FLAG_RXVERR</i>	receive overrun error flag
<i>SPI_FLAG_CONFERR</i>	mode config error flag
<i>SPI_FLAG_CRCERR</i>	CRC error flag
<i>SPI_FLAG_FERR</i>	SPI format error interrupt flag
<i>I2S_FLAG_TBE</i>	transmit buffer empty flag
<i>I2S_FLAG_RBNE</i>	receive buffer not empty flag
<i>I2S_FLAG_TRANS</i>	transmit on-going flag
<i>I2S_FLAG_RXVERR</i>	overrun error flag
<i>I2S_FLAG_TXURERR</i>	underrun error flag
<i>I2S_FLAG_CH</i>	channel side flag
<i>I2S_FLAG_FERR</i>	I2S format error interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get SPI0 transmit buffer empty flag status */

while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));

spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
```

### **spi\_crc\_error\_clear**

The description of `spi_crc_error_clear` is shown as below:

**Table 3-469. Function `spi_crc_error_clear`**

<b>Function name</b>	spi_crc_error_clear
<b>Function prototype</b>	void spi_crc_error_clear(uint32_t spi_periph);
<b>Function descriptions</b>	Clear SPI CRC error flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear SPI0 CRC error flag status */
```

---

```
spi_crc_error_clear(SPI0);
```

## 3.20. SYSCFG

The SYSCFG registers are listed in chapter [3.20.1](#), the SYSCFG firmware functions are introduced in chapter [3.20.2](#).

### 3.20.1. Descriptions of Peripheral registers

SYSCFG registers are listed in the table shown as below:

**Table 3-470. 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_CFG2	system configuration register 2
SYSCFG_CPSCTL	system I/O compensation control register

### 3.20.2. Descriptions of Peripheral functions

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

**Table 3-471. SYSCFG firmware function**

Function name	Function description
syscfg_deinit	deinit syscfg module
syscfg_dma_remap_enable	enable the DMA channels remapping
syscfg_dma_remap_disable	disable the DMA channels remapping
syscfg_high_current_enable	enable PB9 high current capability
syscfg_high_current_disable	disable PB9 high current capability
syscfg_exti_line_config	configure the GPIO pin as EXTI Line
syscfg_lock_config	connect TIMER0/14/15/16 break input to the selected parameter
syscfg_flag_get	check if the specified flag in SYSCFG_CFG2 is set or not
syscfg_flag_clear	clear the flag in SYSCFG_CFG2 by writing 1
syscfg_compensation_config	configure the I/O compensation cell
syscfg_cps_rdy_flag_get	check if the I/O compensation cell ready flag is set or not

#### syscfg\_deinit

The description of syscfg\_deinit is shown as below:

**Table 3-472. 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\_dma\_remap\_enable**

The description of syscfg\_dma\_remap\_enable is shown as below:

**Table 3-473. Function syscfg\_dma\_remap\_enable**

<b>Function name</b>	syscfg_dma_remap_enable
<b>Function prototype</b>	void syscfg_dma_remap_enable (void);
<b>Function descriptions</b>	enable the DMA channels remapping
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_dma_remap</b>	specify the DMA channels to remap
<b>SYSCFG_DMA_REMA P_TIMER16</b>	remap TIMER16 channel0 and UP DMA requests to channel1(default channel0)
<b>SYSCFG_DMA_REMA P_TIMER15</b>	remap TIMER15 channel2 and UP DMA requests to channel3(default channel2)
<b>SYSCFG_DMA_REMA P_USART0RX</b>	remap USART0 Rx DMA request to channel4(default channel2)
<b>SYSCFG_DMA_REMA P_USART0TX</b>	remap USART0 Tx DMA request to channel3(default channel1)
<b>SYSCFG_DMA_REMA P_ADC</b>	remap ADC DMA requests from channel0 to channel1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMA channel remap*/
syscfg_dma_remap_enable(SYSCFG_DMA_REMAP_TIMER16);
```

### **syscfg\_dma\_remap\_disable**

The description of syscfg\_dma\_remap\_disable is shown as below:

**Table 3-474. Function syscfg\_dma\_remap\_disable**

<b>Function name</b>	syscfg_dma_remap_disable
<b>Function prototype</b>	void syscfg_dma_remap_disable (void);
<b>Function descriptions</b>	disable the DMA channels remapping
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
syscfg_dma_remap	specify the DMA channels to remap
SYSCFG_DMA_REMA_P_TIMER16	remap TIMER16 channel0 and UP DMA requests to channel1(default channel0)
SYSCFG_DMA_REMA_P_TIMER15	remap TIMER15 channel2 and UP DMA requests to channel3(default channel2)
SYSCFG_DMA_REMA_P_USART0RX	remap USART0 Rx DMA request to channel4(default channel2)
SYSCFG_DMA_REMA_P_USART0TX	remap USART0 Tx DMA request to channel3(default channel1)
SYSCFG_DMA_REMA_P_ADC	remap ADC DMA requests from channel0 to channel1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA channel remap*/
syscfg_dma_remap_disable(SYSCFG_DMA_REMAP_TIMER16);
```

### **syscfg\_high\_current\_enable**

The description of syscfg\_high\_current\_enable is shown as below:

**Table 3-475. Function syscfg\_high\_current\_enable**

<b>Function name</b>	syscfg_high_current_enable
<b>Function prototype</b>	void syscfg_high_current_enable(void);
<b>Function descriptions</b>	enable PB9 high current capability

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

Example:

```
/* enable PB9 high current capability */

syscfg_high_current_enable();
```

### **syscfg\_high\_current\_disable**

The description of syscfg\_high\_current\_disable is shown as below:

**Table 3-476. Function syscfg\_high\_current\_disable**

<b>Function name</b>	syscfg_high_current_disable
<b>Function prototype</b>	void syscfg_high_current_disable(void);
<b>Function descriptions</b>	disable PB9 high current capability
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable PB9 high current capability */

syscfg_high_current_disable();
```

### **syscfg\_exti\_line\_config**

The description of syscfg\_exti\_line\_config is shown as below:

**Table 3-477. 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>	-

Input parameter{in}	
<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>	x=0..15(GPIOA, GPIOB, GPIOC, GPIOD, GPIOF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the GPIO pin as EXTI Line */
syscfg_exti_line_config(EXTI_SOURCE_GPIOA, EXTI_SOURCE_PIN0);
```

### **syscfg\_lock\_config**

The description of syscfg\_lock\_config is shown as below:

**Table 3-478. Function syscfg\_lock\_config**

<b>Function name</b>	syscfg_lock_config
<b>Function prototype</b>	void syscfg_lock_config (uint32_t syscfg_lock);
<b>Function descriptions</b>	connect TIMER0/14/15/16 break input to the selected parameter
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>syscfg_lock</b>	specify the parameter to be connected
<i>SYSCFG_LOCK_LOCK_UP</i>	Cortex-M4 lockup output connected to the break input
<i>SYSCFG_LOCK_SRA_M_PARITY_ERROR</i>	SRAM_PARITY check error connected to the break input
<i>SYSCFG_LOCK_LVD</i>	LVD interrupt connected to the break input
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure syscfg lock*/
syscfg_lock_config(SYSCFG_LOCK_LOCKUP);
```

### **syscfg\_flag\_get**

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

**Table 3-479. Function syscfg\_flag\_get**

<b>Function name</b>	syscfg_flag_get
<b>Function prototype</b>	FlagStatus syscfg_flag_get(uint32_t syscfg_flag);
<b>Function descriptions</b>	check if the specified flag in SYSCFG_CFG2 is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
syscfg_flag	specify the flag in SYSCFG_CFG2 to check
SYSCFG_SRAM_PCE F	SRAM parity check error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
FlagStatus	SET or RESET

Example:

```
/* get syscfg flag */

FlagStatus status;

status = syscfg_flag_get(SYSCFG_SRAM_PCEF);
```

### **syscfg\_flag\_clear**

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

**Table 3-480. Function syscfg\_flag\_clear**

<b>Function name</b>	syscfg_flag_clear
<b>Function prototype</b>	void syscfg_flag_clear (uint32_t syscfg_flag);
<b>Function descriptions</b>	clear the flag in SYSCFG_CFG2 by writing 1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
syscfg_flag	specify the flag in SYSCFG_CFG2 to check
SYSCFG_SRAM_PCE F	SRAM parity check error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear syscfg flag */

syscfg_flag_clear(SYSCFG_SRAM_PCEF);
```

### **syscfg\_compensation\_config**

The description of syscfg\_compensation\_config is shown as below:

**Table 3-481. Function syscfg\_compensation\_config**

<b>Function name</b>	syscfg_compensation_config
<b>Function prototype</b>	void syscfg_compensation_config(uint32_t syscfg_compensation);
<b>Function descriptions</b>	configure the I/O compensation cell
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_compensation</b>	specifies the I/O compensation cell mode
SYSCFG_COMPENSATION_ENABLE	I/O compensation cell is enabled
SYSCFG_COMPENSATION_DISABLE	I/O compensation cell is disabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the I/O compensation cell */
syscfg_compensation_config(SYSCFG_COMPENSATION_ENABLE);
```

### **syscfg\_cps\_rdy\_flag\_get**

The description of syscfg\_cps\_rdy\_flag\_get is shown as below:

**Table 3-482. Function syscfg\_cps\_rdy\_flag\_get**

<b>Function name</b>	syscfg_cps_rdy_flag_get
<b>Function prototype</b>	FlagStatus syscfg_cps_rdy_flag_get(void);
<b>Function descriptions</b>	check if the I/O compensation cell ready flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear syscfg flag */
```

```

FlagStatus ready_flag;
ready_flag = syscfg_cps_rdy_flag_get();

```

## 3.21. TIMER

The timers have a 16-bit counter that can be used as an unsigned counter and supports both input capture and output compare. Timers (TIMERx) are divided into five sorts: advanced timer (TIMER0), general level0 timer (TIMERx, x=1, 2), general level2 timer (TIMER13), general level2 timer (TIMERx, x=15, 16), Basic timer (TIMER5). The specific functions of different types of timer are different. The TIMER registers are listed in chapter [3.21.1](#), the TIMER firmware functions are introduced in chapter [3.21.2](#).

### 3.21.1. Descriptions of Peripheral registers

TIMERx registers are listed in the table shown as below:

**Table 3-483. TIMERx Registers**

Registers	Descriptions
TIMER_CTL0(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	Control register 0
TIMERx_CTL1(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	Control register 1
TIMERx_SMCFG(TIMERx, x=0, 1, 2, 14)	Slave mode configuration register
TIMERx_DMAINTEN(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	DMA and interrupt enable register
TIMERx_INTF(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	Interrupt flag register
TIMERx_SWEVG(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	Software event generation register
TIMERx_CHCTL0(TIMERx, x=0, 1, 2, 13, 14, 15, 16)	Channel control register 0
TIMERx_CHCTL1(TIMERx, x=0, 1, 2)	Channel control register 1
TIMERx_CHCTL2(TIMERx, x=0, 1, 2, 13, 14, 15, 16)	Channel control register 2
TIMERx_CNT(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	Counter register
TIMERx_PSC(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	Prescaler register
TIMERx_CAR(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	Counter auto reload register
TIMERx_CREP(TIMERx, x=0, 5, 14, 15, 16)	Counter repetition register

Registers	Descriptions
TIMERx_CH0CV(TIMERx, x=0, 1, 2, 13, 14, 15, 16)	Channel 0 capture/compare value register
TIMERx_CH1CV(TIMERx, x=0, 1, 2, 14)	Channel 1 capture/compare value register
TIMERx_CH2CV(TIMERx, x=0, 1, 2)	Channel 2 capture/compare value register
TIMERx_CH3CV(TIMERx, x=0, 1, 2)	Channel 3 capture/compare value register
TIMERx_IRMP(TIMERx, x=13)	Channel complementary protection register
TIMERx_CCHP(TIMERx, x=0, 1, 2, 14, 15, 16)	TIMER complementary channel protection register
TIMERx_DMACFG(TIMERx, x=0, 1, 2, 14, 15, 16)	DMA configuration register
TIMERx_DMATB(TIMERx, x=0, 1, 2, 14, 15, 16)	DMA transfer buffer register
TIMERx_CFG(TIMERx, x=0, 1, 2, 13, 14, 15, 16)	Configuration register

### 3.21.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

**Table 3-484. 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_ocpre_clear_source_config	configure TIMER OCPRE clear source selection

Function name	Function description
timer_interrupt_enable	enable the TIMER interrupt
timer_interrupt_disable	disable the TIMER interrupt
timer_interrupt_flag_get	get timer interrupt flag
timer_interrupt_flag_clear	clear TIMER interrupt flag
timer_flag_get	get TIMER flags
timer_flag_clear	clear TIMER flags
timer_dma_enable	enable the TIMER DMA
timer_dma_disable	disable the TIMER DMA
timer_channel_dma_request_source_select	channel DMA request source selection
timer_dma_transfer_config	configure the TIMER DMA transfer
timer_event_software_generate	software generate events
timer_break_struct_para_init	initialize the parameters of TIMER break parameter struct with the default values
timer_break_config	configure TIMER break function
timer_break_enable	enable TIMER break function
timer_break_disable	disable TIMER break function
timer_automatic_output_enable	enable TIMER output automatic function
timer_automatic_output_disable	disable TIMER output automatic function
timer_primary_output_config	configure TIMER primary output function
timer_channel_control_shadow_config	channel capture/compare control shadow register enable
timer_channel_control_shadow_update_config	configure TIMER channel control shadow register update control
timer_channel_output_struct_para_init	initialize the parameters of TIMER channel output parameter struct with the default values
timer_channel_output_config	configure TIMER channel output function
timer_channel_output_mode_config	configure TIMER channel output compare mode
timer_channel_output_pulse_value_config	configure TIMER channel output pulse value
timer_channel_output_shadow_config	configure TIMER channel output shadow function
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_	initialize the parameters of TIMER channel input parameter

Function name	Function description
para_init	struct with the default values
timer_input_capture_config	configure TIMER input capture parameter
timer_channel_input_capture_prescaler_config	configure TIMER channel input capture prescaler value
timer_channel_capture_value_register_read	read TIMER channel capture compare register value
timer_input_pwm_capture_config	configure TIMER input pwm capture function
timer_hall_mode_config	configure TIMER hall sensor mode
timer_input_trigger_source_select	select TIMER input trigger source
timer_master_output_trigger_source_select	select TIMER master mode output trigger source
timer_slave_mode_select	select TIMER slave mode
timer_master_slave_mode_config	configure TIMER master slave mode
timer_external_trigger_config	configure TIMER external trigger input
timer_quadrature_decoder_mode_config	configure TIMER quadrature decoder mode
timer_internal_clock_config	configure TIMER internal clock mode
timer_internal_trigger_as_external_clock_config	configure TIMER the internal trigger as external clock input
timer_external_trigger_as_external_clock_config	configure TIMER the external trigger as external clock input
timer_external_clock_mode0_config	configure TIMER the external clock mode 0
timer_external_clock_mode1_config	configure TIMER the external clock mode 1
timer_external_clock_mode1_disable	disable TIMER the external clock mode 1
timer_channel_remap_config	configure TIMER channel remap function
timer_write_chxval_register_config	configure TIMER write CHxVAL register selection
timer_output_value_selection_config	configure TIMER output value selection

## Structure timer\_parameter\_struct

Table 3-485. Structure timer\_parameter\_struct

Member name	Function description
prescaler	prescaler value (0~65535)
alignedmode	aligned mode (TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH)
counterdirection	counter direction (TIMER_COUNTER_UP, TIMER_COUNTER_DOWN)
period	period value (0~65535)
clockdivision	clock division value (TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4)

Member name	Function description
repetitioncounter	the counter repetition value (0~255)

## Structure timer\_break\_parameter\_struct

**Table 3-486. 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 3-487. Structure timer\_oc\_parameter\_struct**

Member name	Function description
outputstate	channel output state (TIMER_CCX_ENABLE, TIMER_CCX_DISABLE)
outputnstate	channel complementary output state (TIMER_CCXN_ENABLE, TIMER_CCXN_DISABLE)
ocpolarity	channel output polarity (TIMER_OC_POLARITY_HIGH, TIMER_OC_POLARITY_LOW)
ocnpolarity	channel complementary output polarity (TIMER_OCN_POLARITY_HIGH, TIMER_OCN_POLARITY_LOW)
ocidlestate	idle state of channel output (TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH)
ocnidlestate	idle state of channel complementary output (TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH)

## Structure timer\_ic\_parameter\_struct

**Table 3-488. 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)

Member name	Function description
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 3-489. Function timer\_deinit**

Function name	timer_deinit
Function prototype	void timer_deinit(uint32_t timer_periph);
Function descriptions	deinit a TIMER
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..2, 5, 13..16)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset TIMERO */
timer_deinit (TIMERO);
```

### timer\_struct\_para\_init

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

**Table 3-490. Function timer\_struct\_para\_init**

Function name	timer_struct_para_init
Function prototype	void timer_struct_para_init(timer_parameter_struct* initpara);
Function descriptions	initialize the parameters of TIMER init parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
initpara	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-485. Structure timer_parameter_struct</a> .
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* initialize TIMER init parameter struct with a default value */

timer_parameter_struct timer_initpara;

timer_struct_para_init(timer_initpara);
```

### timer\_init

The description of timer\_init is shown as below:

**Table 3-491. 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(x=0..2, 5, 13..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>initpara</b>	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-485. Structure timer_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize TIMER0 */

timer_parameter_struct timer_initpara;

timer_initpara.prescaler      = 107;
timer_initpara.alignedmode    = TIMER_COUNTER_EDGE;
timer_initpara.counterdirection = TIMER_COUNTER_UP;
timer_initpara.period         = 999;
timer_initpara.clockdivision  = TIMER_CKDIV_DIV1;
timer_initpara.repetitioncounter = 1;
```

```
timer_init(TIMER0,&timer_initpara);
```

### **timer\_enable**

The description of timer\_enable is shown as below:

**Table 3-492. Function timer\_enable**

<b>Function name</b>	timer_enable
<b>Function prototype</b>	void timer_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable a timer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 */
timer_enable (TIMER0);
```

### **timer\_disable**

The description of timer\_disable is shown as below:

**Table 3-493. Function timer\_disable**

<b>Function name</b>	timer_disable
<b>Function prototype</b>	void timer_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable a timer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 */
```

```
timer_disable (TIMER0);
```

### **timer\_auto\_reload\_shadow\_enable**

The description of timer\_auto\_reload\_shadow\_enable is shown as below:

**Table 3-494. 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>	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_enable (TIMER0);
```

### **timer\_auto\_reload\_shadow\_disable**

The description of timer\_auto\_reload\_shadow\_disable is shown as below:

**Table 3-495. 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>	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the TIMER0 auto reload shadow function */

timer_auto_reload_shadow_disable (TIMER0);
```

### **timer\_update\_event\_enable**

The description of timer\_update\_event\_enable is shown as below:

**Table 3-496. Function timer\_update\_event\_enable**

<b>Function name</b>	timer_update_event_enable
<b>Function prototype</b>	void timer_update_event_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable the update event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 the update event */

timer_update_event_enable (TIMER0);
```

### **timer\_update\_event\_disable**

The description of timer\_update\_event\_disable is shown as below:

**Table 3-497. Function timer\_update\_event\_disable**

<b>Function name</b>	timer_update_event_disable
<b>Function prototype</b>	void timer_update_event_disable (uint32_t timer_periph);
<b>Function descriptions</b>	disable the update event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* disable TIMER0 the update event */

timer_update_event_disable (TIMER0);
```

### timer\_counter\_alignment

The description of timer\_counter\_alignment is shown as below:

**Table 3-498. 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
<b>TIMERx(x=0..2)</b>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>aligned</b>	alignment mode
<b>TIMER_COUNTER_ED GE</b>	No center-aligned mode (edge-aligned mode). The direction of the counter is specified by the DIR bit.
<b>TIMER_COUNTER_CE NTER_DOWN</b>	Center-aligned and counting down assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0register). Only when the counter is counting down, compare interrupt flag of channels can be set.
<b>TIMER_COUNTER_CE NTER_UP</b>	Center-aligned and counting up assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0register). Only when the counter is counting up, compare interrupt flag of channels can be set.
<b>TIMER_COUNTER_CE NTER_BOTH</b>	Center-aligned and counting up/down assert mode. The counter counts under center-aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0 register). Both when the counter is counting up and counting down, compare interrupt flag of channels can be set.
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set TIMER0 counter center-aligned and counting up assert mode */
```

timer\_counter\_alignment (TIMER0, TIMER\_COUNTER\_CENTER\_UP);

### **timer\_counter\_up\_direction**

The description of timer\_counter\_up\_direction is shown as below:

**Table 3-499. Function timer\_counter\_up\_direction**

<b>Function name</b>	timer_counter_up_direction
<b>Function prototype</b>	void timer_counter_up_direction(uint32_t timer_periph);
<b>Function descriptions</b>	set TIMER counter up direction
<b>Precondition</b>	set TIMER counter no center-aligned mode (edge-aligned mode)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set TIMER0 counter up direction */

timer_counter_up_direction (TIMER0);
```

### **timer\_counter\_down\_direction**

The description of timer\_counter\_down\_direction is shown as below:

**Table 3-500. Function timer\_counter\_down\_direction**

<b>Function name</b>	timer_counter_down_direction
<b>Function prototype</b>	void timer_counter_down_direction(uint32_t timer_periph);
<b>Function descriptions</b>	set TIMER counter down direction
<b>Precondition</b>	set TIMER counter no center-aligned mode (edge-aligned mode)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set TIMER0 counter down direction */
```

timer\_counter\_down\_direction (TIMER0);

### **timer\_prescaler\_config**

The description of timer\_prescaler\_config is shown as below:

**Table 3-501. Function timer\_prescaler\_config**

<b>Function name</b>	timer_prescaler_config
<b>Function prototype</b>	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint8_t pscreload);
<b>Function descriptions</b>	configure TIMER prescaler
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>prescaler</b>	prescaler value (0~65535)
<b>Input parameter{in}</b>	
<b>pscreload</b>	prescaler reload mode
<i>TIMER_PSC_RELOAD _NOW</i>	the prescaler is loaded right now
<i>TIMER_PSC_RELOAD _UPDATE</i>	the prescaler is loaded at the next update event
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 prescaler */
timer_prescaler_config (TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

### **timer\_repetition\_value\_config**

The description of timer\_repetition\_value\_config is shown as below:

**Table 3-502. Function timer\_repetition\_value\_config**

<b>Function name</b>	timer_repetition_value_config
<b>Function prototype</b>	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
<b>Function descriptions</b>	configure TIMER repetition register value
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>repetition</b>	the counter repetition value (0~255)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 repetition register value */

timer_repetition_value_config (TIMER0, 98);
```

### **timer\_autoreload\_value\_config**

The description of timer\_autoreload\_value\_config is shown as below:

**Table 3-503. Function timer\_autoreload\_value\_config**

<b>Function name</b>	timer_autoreload_value_config
<b>Function prototype</b>	void timer_autoreload_value_config(uint32_t timer_periph, uint16_t autoreload);
<b>Function descriptions</b>	configure TIMER autoreload register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>autoreload</b>	the counter auto-reload value (0-65535)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER autoreload register value */

timer_autoreload_value_config (TIMER0, 3000);
```

### timer\_counter\_value\_config

The description of timer\_counter\_value\_config is shown as below:

**Table 3-504. Function timer\_counter\_value\_config**

<b>Function name</b>	timer_counter_value_config
<b>Function prototype</b>	void timer_counter_value_config(uint32_t timer_periph, uint16_t counter);
<b>Function descriptions</b>	configure TIMER counter register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>counter</b>	the counter value (0-65535)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 counter register value */

timer_counter_value_config (TIMER0, 3000);
```

### timer\_counter\_read

The description of timer\_counter\_read is shown as below:

**Table 3-505. Function timer\_counter\_read**

<b>Function name</b>	timer_counter_read
<b>Function prototype</b>	uint32_t timer_counter_read(uint32_t timer_periph);
<b>Function descriptions</b>	read TIMER counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	counter value (0-65535)

Example:

```
/* read TIMER0 counter value */

uint32_t i = 0;

i = timer_counter_read (TIMER0);
```

### **timer\_prescaler\_read**

The description of timer\_prescaler\_read is shown as below:

**Table 3-506. Function timer\_prescaler\_read**

<b>Function name</b>	timer_prescaler_read
<b>Function prototype</b>	uint16_t timer_prescaler_read(uint32_t timer_periph);
<b>Function descriptions</b>	read TIMER prescaler value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	prescaler register value (0~65535)

Example:

```
/* read TIMER0 prescaler value */

uint16_t i = 0;

i = timer_prescaler_read (TIMER0);
```

### **timer\_single\_pulse\_mode\_config**

The description of timer\_single\_pulse\_mode\_config is shown as below:

**Table 3-507. Function timer\_single\_pulse\_mode\_config**

<b>Function name</b>	timer_single_pulse_mode_config
<b>Function prototype</b>	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);
<b>Function descriptions</b>	configure TIMER single pulse mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral

<i>TIMERx(x=0..2, 5, 14..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>spmode</b>	pulse mode
<i>TIMER_SP_MODE_SIN_GLE</i>	single pulse mode
<i>TIMER_SP_MODE_REPETITIVE</i>	repetitive pulse mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 single pulse mode */
timer_single_pulse_mode_config (TIMER0, TIMER_SP_MODE_SINGLE);
```

### **timer\_update\_source\_config**

The description of timer\_update\_source\_config is shown as below:

**Table 3-508. Function timer\_update\_source\_config**

<b>Function name</b>	timer_update_source_config
<b>Function prototype</b>	void timer_update_source_config(uint32_t timer_periph, uint32_t update);
<b>Function descriptions</b>	configure TIMER update source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>update</b>	update source
<i>TIMER_UPDATE_SRC_GLOBAL</i>	Any of the following events generate an update interrupt or DMA request: - The UPG bit is set - The counter generates an overflow or underflow event - The slave mode controller generates an update event
<i>TIMER_UPDATE_SRC_REGULAR</i>	Only counter overflow/underflow generates an update interrupt or DMA request.
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER update only by counter overflow/underflow */

timer_update_source_config (TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

### **timer\_ocpre\_clear\_source\_config**

The description of timer\_ocpre\_clear\_source\_config is shown as below:

**Table 3-509. Function t timer\_ocpre\_clear\_source\_config**

<b>Function name</b>	timer_ocpre_clear_source_config
<b>Function prototype</b>	void timer_ocpre_clear_source_config (uint32_t timer_periph, uint8_t ocpreclear);
<b>Function descriptions</b>	configure TIMER OCPRE clear source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..2)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
ocpreclear	clear source
<i>TIMER_OCPRE_CLR_SOURCE_CLK_R</i>	OCPRE_CLR_INT is connected to the OCPRE_CLR input
<i>TIMER_OCPRE_CLK_SOURCE_ETIF_F</i>	OCPRE_CLR_INT is connected to ETIF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

例如：

```
/* configure TIMER0 OCPRE_CLR_INT is connected to the OCPRE_CLR input */

timer_ocpre_clear_source_config(TIMER0, TIMER_OCPRE_CLEAR_SOURCE_CLR);
```

### **timer\_interrupt\_enable**

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

**Table 3-510. 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
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	timer interrupt enable source
<b>TIMER_INT_UP</b>	update interrupt enable, TIMERx (x=0..2, 5, 13..16)
<b>TIMER_INT_CHO</b>	channel 0 interrupt enable, TIMERx(x=0..2, 13..16)
<b>TIMER_INT_CH1</b>	channel 1 interrupt enable, TIMERx(x=0..2, 14)
<b>TIMER_INT_CH2</b>	channel 2 interrupt enable, TIMERx(x=0..2)
<b>TIMER_INT_CH3</b>	channel 3 interrupt enable , TIMERx(x=0..2)
<b>TIMER_INT_CMT</b>	commutation interrupt enable, TIMERx (x=0, 14..16)
<b>TIMER_INT_TRG</b>	trigger interrupt enable, TIMERx(x=0..2, 14)
<b>TIMER_INT_BRK</b>	break interrupt enable, TIMERx (x=0, 14..16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the TIMER0 update interrupt */
timer_interrupt_enable (TIMER0, TIMER_INT_UP);
```

### timer\_interrupt\_disable

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

**Table 3-511. 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
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	timer interrupt disable source
<b>TIMER_INT_UP</b>	update interrupt disable, TIMERx (x=0..2, 5, 13..16)
<b>TIMER_INT_CHO</b>	channel 0 interrupt disable, TIMERx(x=0..2, 13..16)

<i>TIMER_INT_CH1</i>	channel 1 interrupt disable, TIMERx(x=0..2, 14)
<i>TIMER_INT_CH2</i>	channel 2 interrupt disable, TIMERx(x=0..2)
<i>TIMER_INT_CH3</i>	channel 3 interrupt disable, TIMERx(x=0..2)
<i>TIMER_INT_CMT</i>	commutation interrupt disable, TIMERx (x=0, 14..16)
<i>TIMER_INT_TRG</i>	trigger interrupt disable, TIMERx(x=0..2, 14)
<i>TIMER_INT_BRK</i>	break interrupt disable, TIMERx(x=0, 14..16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the TIMER0 update interrupt */

timer_interrupt_disable (TIMER0, TIMER_INT_UP);
```

### **timer\_interrupt\_flag\_get**

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

**Table 3-512. 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
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag,TIMERx(x=0..2, 5, 13..16)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag,TIMERx(x=0..2, 13..16)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag,TIMERx(x=0..2, 14)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag,TIMERx TIMERx(x=0..2)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag,TIMERx TIMERx(x=0..2)
<i>TIMER_INT_FLAG_CM<sub>T</sub></i>	channel commutation interrupt flag, TIMERx (x=0, 14..16)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, TIMERx(x=0..2, 14)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, TIMERx(x=0, 14..16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

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

Example:

```
/* get TIMER0 update interrupt flag */

FlagStatus Flag_interrupt = RESET;

Flag_interrupt = timer_interrupt_flag_get (TIMER0, TIMER_INT_FLAG_UP);
```

### **timer\_interrupt\_flag\_clear**

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

**Table 3-513. 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
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	the timer interrupt bits
<b>TIMER_INT_FLAG_UP</b>	update interrupt flag,TIMERx(x=0..2, 5, 13..16)
<b>TIMER_INT_FLAG_CH0</b>	channel 0 interrupt flag,TIMERx(x=0..2, 13..16)
<b>TIMER_INT_FLAG_CH1</b>	channel 1 interrupt flag,TIMERx(x=0..2, 14)
<b>TIMER_INT_FLAG_CH2</b>	channel 2 interrupt flag,TIMERx TIMERx(x=0..2)
<b>TIMER_INT_FLAG_CH3</b>	channel 3 interrupt flag,TIMERx TIMERx(x=0..2)
<b>TIMER_INT_FLAG_CM<sub>T</sub></b>	channel commutation interrupt flag, TIMERx (x=0, 14..16)
<b>TIMER_INT_FLAG_TRG</b>	trigger interrupt flag, TIMERx(x=0..2, 14)
<b>TIMER_INT_FLAG_BRK</b>	break interrupt flag, TIMERx(x=0, 14..16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TIMER0 update interrupt flag */

timer_interrupt_flag_clear (TIMER0, TIMER_INT_FLAG_UP);
```

### **timer\_flag\_get**

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

**Table 3-514. 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
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>flag</b>	the timer interrupt flags
<b>TIMER_FLAG_UP</b>	update flag, TIMERx(x=0..2, 5, 13..16)
<b>TIMER_FLAG_CH0</b>	channel 0 flag, TIMERx(x=0..2, 13..16)
<b>TIMER_FLAG_CH1</b>	channel 1 flag, TIMERx(x=0..2, 14)
<b>TIMER_FLAG_CH2</b>	channel 2 flag, TIMERx(x=0..2)
<b>TIMER_FLAG_CH3</b>	channel 3 flag, TIMERx(x=0..2)
<b>TIMER_FLAG_CMT</b>	channel commutation flag, TIMERx(x=0, 14..16)
<b>TIMER_FLAG_TRG</b>	trigger flag, TIMERx(x=0..2, 14)
<b>TIMER_FLAG_BRK</b>	break flag, TIMERx(x=0, 14..16)
<b>TIMER_FLAG_CH0O</b>	channel 0 overcapture flag, TIMERx(x=0..2, 3..16)
<b>TIMER_FLAG_CH1O</b>	channel 1 overcapture flag, TIMERx(x=0..2, 14)
<b>TIMER_FLAG_CH2O</b>	channel 2 overcapture flag, TIMERx(x=0..2)
<b>TIMER_FLAG_CH3O</b>	channel 3 overcapture flag, TIMERx(x=0..2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TIMER0 update flags */

FlagStatus Flag_status = RESET;

Flag_status = timer_flag_get (TIMER0, TIMER_FLAG_UP);
```

### **timer\_flag\_clear**

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

**Table 3-515. 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>	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>flag</b>	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag, TIMERx(x=0..2, 5, 13..16)
<i>TIMER_FLAG_CH0</i>	channel 0 flag, TIMERx(x=0..2, 13..16)
<i>TIMER_FLAG_CH1</i>	channel 1 flag, TIMERx(x=0..2, 14)
<i>TIMER_FLAG_CH2</i>	channel 2 flag, TIMERx(x=0..2)
<i>TIMER_FLAG_CH3</i>	channel 3 flag, TIMERx(x=0..2)
<i>TIMER_FLAG_CMT</i>	channel commutation flag, TIMERx(x=0, 14..16)
<i>TIMER_FLAG_TRG</i>	trigger flag, TIMERx(x=0..2, 14)
<i>TIMER_FLAG_BRK</i>	break flag, TIMERx(x=0, 14..16)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag, TIMERx(x=0..2, 13..16)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag, TIMERx(x=0..2, 14)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag, TIMERx(x=0..2)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag, TIMERx(x=0..2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear TIMER0 update flags */

timer_flag_clear (TIMER0, TIMER_FLAG_UP);
```

### timer\_dma\_enable

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

**Table 3-516. Function timer\_dma\_enable**

<b>Function name</b>	timer_dma_enable
<b>Function prototype</b>	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
<b>Function descriptions</b>	enable the TIMER DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>dma</b>	timer DMA source enable
<i>TIMER_DMA_UPD</i>	update DMA enable, TIMERx(x=0..2, 5, 14..16)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable, TIMERx(x=0..2, 14..16)

<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable, TIMERx(x=0..2, 4)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA enable, TIMERx(x=0..2)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA enable, TIMERx(x=0..2)
<i>TIMER_DMA_CMTD</i>	commutation DMA request enable, TIMERx(x=0, 14)
<i>TIMER_DMA_TRGD</i>	trigger DMA enable, TIMERx(x=0..2, 14)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the TIMER0 update DMA */

timer_dma_enable (TIMER0, TIMER_DMA_UPD);
```

### **timer\_dma\_disable**

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

**Table 3-517. Function timer\_dma\_disable**

<b>Function name</b>	timer_dma_disable
<b>Function prototype</b>	void timer_dma_disable (uint32_t timer_periph, uint16_t dma);
<b>Function descriptions</b>	disable the TIMER DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>dma</b>	timer DMA source disable
<i>TIMER_DMA_UPD</i>	update DMA enable, TIMERx(x=0..2, 5, 14..16)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable, TIMERx(x=0..2, 14..16)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable, TIMERx(x=0..2, 14)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA enable, TIMERx(x=0..2)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA enable, TIMERx(x=0..2)
<i>TIMER_DMA_CMTD</i>	commutation DMA request enable, TIMERx(x=0, 14)
<i>TIMER_DMA_TRGD</i>	trigger DMA enable, TIMERx(x=0..2, 14)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the TIMER0 update DMA */
```

```
timer_dma_disable(TIMER0, TIMER_DMA_UPD);
```

### **timer\_channel\_dma\_request\_source\_select**

The description of timer\_channel\_dma\_request\_source\_select is shown as below:

**Table 3-518. Function timer\_channel\_dma\_request\_source\_select**

<b>Function name</b>	timer_channel_dma_request_source_select
<b>Function prototype</b>	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
<b>Function descriptions</b>	channel DMA request source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 14..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>dma_request</b>	channel DMA request source selection
<i>TIMER_DMAREQUEST_CHANNELEVENT</i>	DMA request of channel n is sent when channel y event occurs
<i>TIMER_DMAREQUEST_UPDATEEVENT</i>	DMA request of channel n is sent when update event occurs
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* TIMER0 channel DMA request of channel n is sent when channel y event occurs */

timer_channel_dma_request_source_select(TIMER0,
TMR_DMAREQUEST_CHANNELEVENT);
```

### **timer\_dma\_transfer\_config**

The description of timer\_dma\_transfer\_config is shown as below:

**Table 3-519. Function timer\_dma\_transfer\_config**

<b>Function name</b>	timer_dma_transfer_config
<b>Function prototype</b>	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
<b>Function descriptions</b>	configure the TIMER DMA transfer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 14..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>dma_baseaddr</b>	DMA transfer access start address
<i>TIMER_DMACFG_DMA_TA_CTL0</i>	DMA transfer address is TIMER_CTL0, <i>TIMERx(x=0..2, 14..16)</i>
<i>TIMER_DMACFG_DMA_TA_CTL1</i>	DMA transfer address is TIMER_CTL1, <i>TIMERx(x=0..2, 14..16)</i>
<i>TIMER_DMACFG_DMA_TA_SMCFG</i>	DMA transfer address is TIMER_SMCFG, <i>TIMERx(x=0..2, 14)</i>
<i>TIMER_DMACFG_DMA_TA_DMAINTEN</i>	DMA transfer address is TIMER_DMAINTEN, <i>TIMERx(x=0..2, 14..16)</i>
<i>TIMER_DMACFG_DMA_TA_INTF</i>	DMA transfer address is TIMER_INTF, <i>TIMERx(x=0..2, 14..16)</i>
<i>TIMER_DMACFG_DMA_TA_SWEVG</i>	DMA transfer address is TIMER_SWEVG, <i>TIMERx(x=0..2, 14..16)</i>
<i>TIMER_DMACFG_DMA_TA_CHCTL0</i>	DMA transfer address is TIMER_CHCTL0, <i>TIMERx(x=0..2, 14..16)</i>
<i>TIMER_DMACFG_DMA_TA_CHCTL1</i>	DMA transfer address is TIMER_CHCTL1, <i>TIMERx(x=0..2)</i>
<i>TIMER_DMACFG_DMA_TA_CHCTL2</i>	DMA transfer address is TIMER_CHCTL2, <i>TIMERx(x=0..2, 14..16)</i>
<i>TIMER_DMACFG_DMA_TA_CNT</i>	DMA transfer address is TIMER_CNT, <i>TIMERx(x=0..2, 14..16)</i>
<i>TIMER_DMACFG_DMA_TA_PSC</i>	DMA transfer address is TIMER_PSC, <i>TIMERx(x=0..2, 14..16)</i>
<i>TIMER_DMACFG_DMA_TA_CAR</i>	MA transfer address is TIMER_CAR, <i>TIMERx(x=0..2, 14..16)</i>
<i>TIMER_DMACFG_DMA_TA_CREP</i>	DMA transfer address is TIMER_CREP, <i>TIMERx(x=0, 14..16)</i>
<i>TIMER_DMACFG_DMA_TA_CH0CV</i>	DMA transfer address is TIMER_CH0CV, <i>TIMERx(x=0..2, 14..16)</i>
<i>TIMER_DMACFG_DMA_TA_CH1CV</i>	DMA transfer address is TIMER_CH1CV, <i>TIMERx(x=0..2, 14)</i>
<i>TIMER_DMACFG_DMA_TA_CH2CV</i>	DMA transfer address is TIMER_CH2CV, <i>TIMERx(x=0..2)</i>
<i>TIMER_DMACFG_DMA_TA_CH3CV</i>	DMA transfer address is TIMER_CH3CV, <i>TIMERx(x=0..2)</i>
<i>TIMER_DMACFG_DMA_TA_CCHP</i>	DMA transfer address is TIMER_CCHP, <i>TIMERx(x=0, 14..16)</i>
<i>TIMER_DMACFG_DMA_TA_DMACFG</i>	DMA transfer address is TIMER_DMACFG, <i>TIMERx(x=0..2, 14..16)</i>
<b>Input parameter{in}</b>	

<b>dma_lenth</b>	DMA transfer count
<i>TIMER_DMACFG_DMA_TC_xTRANSFER</i>	x=1..18, DMA transfer x time
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the TIMER0 DMA transfer */

timer_dma_transfer_config(TIMER0, TIMER_DMACFG_DMATA_CTL0,
    TIMER_DMACFG_DMATC_5TRANSFER);
```

### timer\_event\_software\_generate

The description of timer\_event\_software\_generate is shown as below:

**Table 3-520. 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_U PG</i>	update event, TIMERx(x=0..2, 5, 13..16)
<i>TIMER_EVENT_SRC_C H0G</i>	channel 0 capture or compare event generation, TIMERx(x=0..2, 13..16)
<i>TIMER_EVENT_SRC_C H1G</i>	channel 1 capture or compare event generation, TIMERx(x=0..2, 14)
<i>TIMER_EVENT_SRC_C H2G</i>	channel 2 capture or compare event generation, TIMERx(x=0..2)
<i>TIMER_EVENT_SRC_C H3G</i>	channel 3 capture or compare event generation, TIMERx(x=0..2)
<i>TIMER_EVENT_SRC_C MTG</i>	channel commutation event generation, TIMERx(x=0, 14..16)
<i>TIMER_EVENT_SRC_T RGG</i>	trigger event generation, TIMERx(x=0..2, 14)

<i>TIMER_EVENT_SRC_B</i> RKG	break event generation, TIMERx(x=0, 14..16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* software generate update event*/
timer_event_software_generate (TIMER0, TIMER_EVENT_SRC_UPG);
```

### **timer\_break\_struct\_para\_init**

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

**Table 3-521. 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>	
breakpara	TIMER break parameter struct, the structure members can refer to <a href="#">Table 3-486. Structure timer break parameter struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize TIMER break parameter struct with a default value */
timer_break_parameter_struct timer_breakpara;
timer_break_struct_para_init(timer_breakpara);
```

### **timer\_break\_config**

The description of timer\_break\_config is shown as below:

**Table 3-522. 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..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>breakpara</b>	TIMER break parameter struct, the structure members can refer to <a href="#">Table 3-486, Structure timer_break_parameter_struct.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 break function */

timer_break_parameter_struct timer_breakpara;

timer_breakpara.runoffstate      = TIMER_ROS_STATE_DISABLE;
timer_breakpara.ideloffstate     = TIMER_IOS_STATE_DISABLE ;
timer_breakpara.deadtime         = 255;
timer_breakpara.breakpolarity    = TIMER_BREAK_POLARITY_LOW;
timer_breakpara.outputautostate = TIMER_OUTAUTO_ENABLE;
timer_breakpara.protectmode     = TIMER_CCHP_PROT_0;
timer_breakpara.breakstate       = TIMER_BREAK_ENABLE;

timer_break_config(TIMER0, &timer_breakpara);
```

### **timer\_break\_enable**

The description of timer\_break\_enable is shown as below:

**Table 3-523. Function timer\_break\_enable**

<b>Function name</b>	timer_break_enable
<b>Function prototype</b>	void timer_break_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable TIMER break function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-field in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 break function*/
timer_break_enable (TIMER0);
```

### **timer\_break\_disable**

The description of timer\_break\_disable is shown as below:

**Table 3-524. Function timer\_break\_disable**

<b>Function name</b>	timer_break_disable
<b>Function prototype</b>	void timer_break_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER break function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 break function*/
timer_break_disable (TIMER0);
```

### **timer\_automatic\_output\_enable**

The description of timer\_automatic\_output\_enable is shown as below:

**Table 3-525. 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..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 output automatic function */

timer_automatic_output_enable (TIMER0);
```

### **timer\_automatic\_output\_disable**

The description of timer\_automatic\_output\_disable is shown as below:

**Table 3-526. 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-filled in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 output automatic function */

timer_automatic_output_disable (TIMER0);
```

### **timer\_primary\_output\_config**

The description of timer\_primary\_output\_config is shown as below:

**Table 3-527. Function timer\_primary\_output\_config**

<b>Function name</b>	timer_primary_output_config
<b>Function prototype</b>	void timer_primary_output_config(uint32_t timer_periph, ControlStatus newvalue);

<b>Function descriptions</b>	configure TIMER primary output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 primary output function */
timer_primary_output_config (TIMER0, ENABLE);
```

### **timer\_channel\_control\_shadow\_config**

The description of `timer_channel_control_shadow_config` is shown as below:

**Table 3-528. Function `timer_channel_control_shadow_config`**

<b>Function name</b>	timer_channel_control_shadow_config
<b>Function prototype</b>	void timer_channel_control_shadow_config(uint32_t timer_periph, ControlStatus newvalue);
<b>Function descriptions</b>	channel commutation control shadow register enable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

/\* channel capture/compare control shadow register enable \*/

```
timer_channel_control_shadow_config (TIMER0, ENABLE);
```

### **timer\_channel\_control\_shadow\_update\_config**

The description of timer\_channel\_control\_shadow\_update\_config is shown as below:

**Table 3-529. Function timer\_channel\_control\_shadow\_update\_config**

<b>Function name</b>	timer_channel_control_shadow_update_config
<b>Function prototype</b>	void timer_channel_control_shadow_update_config(uint32_t timer_periph, uint8_t ccuctl);
<b>Function descriptions</b>	configure commutation control shadow register update control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>ccuctl</b>	channel control shadow register update control
<i>TIMER_UPDATECTL_CCU</i>	the shadow registers update by when CMTG bit is set
<i>TIMER_UPDATECTL_CCUTRI</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 3-530. Function timer\_channel\_output\_struct\_para\_init**

<b>Function name</b>	timer_channel_output_struct_para_init
<b>Function prototype</b>	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpara);
<b>Function descriptions</b>	initialize the parameters of TIMER channel output parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
ocpara	TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-487. Structure timer_oc_parameter_struct</a> .
Output parameter{out}	
-	-
Return value	
-	-

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 3-531. Function timer\_channel\_output\_config**

<b>Function name</b>	timer_channel_output_config
<b>Function prototype</b>	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara);
<b>Function descriptions</b>	configure TIMER channel output function
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0 (TIMERx(x=0..2, 13..16))
TIMER_CH_1	TIMER channel 1 (TIMERx(x=0..2, 14))
TIMER_CH_2	TIMER channel 2 (TIMERx(x=0..2))
TIMER_CH_3	TIMER channel 3 (TIMERx(x=0..2))
Input parameter{in}	
ocpara	TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-487. Structure timer_oc_parameter_struct</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output function */
```

```

timer_oc_parameter_struct timer_ocinitpara;

timer_ocinitpara.outputstate = TIMER_CCX_ENABLE;
timer_ocinitpara.outputnstate = TIMER_CCXN_ENABLE;
timer_ocinitpara.ocpolarity = TIMER_OC_POLARITY_HIGH;
timer_ocinitpara.ocnpolarity = TIMER_OCN_POLARITY_HIGH;
timer_ocinitpara.ocidlestate = TIMER_OC_IDLE_STATE_HIGH;
timer_ocinitpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;
timer_channel_output_config(TIMER0, TIMER_CH_0, &timer_ocinitpara);

```

### **timer\_channel\_output\_mode\_config**

The description of timer\_channel\_output\_mode\_config is shown as below:

**Table 3-532. 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
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<b>TIMER_CH_0</b>	TIMER channel 0 (TIMERx (x=0..2, 13..16))
<b>TIMER_CH_1</b>	TIMER channel 1 (TIMERx (x=0..2, 14))
<b>TIMER_CH_2</b>	TIMER channel 2 (TIMERx (x=0..2))
<b>TIMER_CH_3</b>	TIMER channel 3 (TIMERx (x=0..2))
<b>Input parameter{in}</b>	
<b>ocmode</b>	channel output compare mode
<b>TIMER_OC_MODE_TIMING</b>	timing mode
<b>TIMER_OC_MODE_ACTIVE</b>	set the channel output
<b>TIMER_OC_MODE_INACTIVE</b>	clear the channel output
<b>TIMER_OC_MODE_TOGGLE</b>	toggle on match
<b>TIMER_OC_MODE_FORCE</b>	force low mode

<i>W</i>	
<i>TIMER_OC_MODE_HI</i> <i>H</i>	force high mode
<i>TIMER_OC_MODE_PWM</i> <i>M0</i>	PWM mode 0
<i>TIMER_OC_MODE_PWM</i> <i>M1</i>	PWM mode 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel PWM 0 mode */
timer_channel_output_mode_config(TIMER0, TIMER_CH_0, TIMER_OC_MODE_PWM0);
```

### **timer\_channel\_output\_pulse\_value\_config**

The description of timer\_channel\_output\_pulse\_value\_config is shown as below:

**Table 3-533. Function timer\_channel\_output\_pulse\_value\_config**

<b>Function name</b>	timer_channel_output_pulse_value_config
<b>Function prototype</b>	void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse);
<b>Function descriptions</b>	configure TIMER channel output pulse value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx (x=0..2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx TIMERx(x=0..2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx (x=0..2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx (x=0..2))
<b>Input parameter{in}</b>	
<b>pulse</b>	channel output pulse value (0~65535)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 output pulse value */

timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

### **timer\_channel\_output\_shadow\_config**

The description of timer\_channel\_output\_shadow\_config is shown as below:

**Table 3-534. 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
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<b>TIMER_CH_0</b>	TIMER channel 0 (TIMERx (x=0..2, 13..16))
<b>TIMER_CH_1</b>	TIMER channel 1 (TIMERx (x=0..2, 14))
<b>TIMER_CH_2</b>	TIMER channel 2 (TIMERx (x=0..2))
<b>TIMER_CH_3</b>	TIMER channel 3 (TIMERx (x=0..2))
<b>Input parameter{in}</b>	
<b>ocshadow</b>	channel output shadow state
<b>TIMER_OC_SHADOW_ENABLE</b>	channel output shadow state enable
<b>TIMER_OC_SHADOW_DISABLE</b>	channel output shadow state disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*configure TIMER0 channel 0 output shadow function */

timer_channel_output_shadow_config (TIMER0, TIMER_CH_0,
TICKER_OC_SHADOW_ENABLE);
```

### **timer\_channel\_output\_fast\_config**

The description of timer\_channel\_output\_fast\_config is shown as below:

**Table 3-535. 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
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<b>TIMER_CH_0</b>	TIMER channel 0 (TIMERx (x=0..2, 13..16))
<b>TIMER_CH_1</b>	TIMER channel 1 (TIMERx (x=0..2, 14))
<b>TIMER_CH_2</b>	TIMER channel 2 (TIMERx (x=0..2))
<b>TIMER_CH_3</b>	TIMER channel 3 (TIMERx (x=0..2))
<b>Input parameter{in}</b>	
<b>ocfast</b>	channel output fast function
<b>TIMER_OC_FAST_ENA BLE</b>	channel output fast function enable
<b>TIMER_OC_FAST_DIS ABLE</b>	channel output fast function disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 output fast function */
timer_channel_output_fast_config (TIMER0, TIMER_CH_0, TIMER_OC_FAST_ENABLE);
```

### **timer\_channel\_output\_clear\_config**

The description of timer\_channel\_output\_clear\_config is shown as below:

**Table 3-536. Function timer\_channel\_output\_clear\_config**

<b>Function name</b>	timer_channel_output_clear_config
<b>Function prototype</b>	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);
<b>Function descriptions</b>	configure TIMER channel output clear function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>timer_periph</b>	TIMER periphera
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx (x=0..2))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx (x=0..2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx (x=0..2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx (x=0..2))
<b>Input parameter{in}</b>	
<b>occlear</b>	channel output clear function
<i>TIMER_OC_CLEAR_ENABLE</i>	channel output clear function enable
<i>TIMER_OC_CLEAR_DISABLE</i>	channel output clear function disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 output clear function */
timer_channel_output_clear_config (TIMER0, TIMER_CH_0,
TMR_OC_CLEAR_ENABLE);
```

### **timer\_channel\_output\_polarity\_config**

The description of `timer_channel_output_polarity_config` is shown as below:

**Table 3-537. Function `timer_channel_output_polarity_config`**

<b>Function name</b>	timer_channel_output_polarity_config
<b>Function prototype</b>	void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
<b>Function descriptions</b>	configure TIMER channel output polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx (x=0..2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx (x=0..2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..2))

<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx (x=0..2))
<b>Input parameter{in}</b>	
<b>ocpolarity</b>	channel output polarity
<i>TIMER_OC_POLARITY_HIGH</i>	channel output polarity is high
<i>TIMER_OC_POLARITY_LOW</i>	channel output polarity is low
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 output polarity */

timer_channel_output_polarity_config (TIMER0, TIMER_CH_0,
TIMER_OC_POLARITY_HIGH);
```

#### **timer\_channel\_complementary\_output\_polarity\_config**

The description of timer\_channel\_complementary\_output\_polarity\_config is shown as below:

**Table 3-538. Function timer\_channel\_complementary\_output\_polarity\_config**

<b>Function name</b>	timer_channel_complementary_output_polarity_config
<b>Function prototype</b>	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
<b>Function descriptions</b>	configure TIMER channel complementary output polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx (x=0..2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx (x=0..2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx (x=0..2))
<i>TIMER_CH_3</i>	TIMER channel 3(TIMERx (x=0..2))
<b>Input parameter{in}</b>	
<b>ocpolarity</b>	channel complementary output polarity
<i>TIMER_OCN_POLARITY_HIGH</i>	channel complementary output polarity is high
<i>TIMER_OCN_POLARITY_LOW</i>	channel complementary output polarity is low

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 3-539. Function timer\_channel\_output\_state\_config**

<b>Function name</b>	timer_channel_output_state_config
<b>Function prototype</b>	void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
<b>Function descriptions</b>	configure TIMER channel enable state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0( <i>TIMERx</i> (x=0..2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1( <i>TIMERx</i> (x=0..2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2( <i>TIMERx</i> (x=0..2))
<i>TIMER_CH_3</i>	TIMER channel 3( <i>TIMERx</i> (x=0..2))
<b>Input parameter{in}</b>	
<b>state</b>	TIMER channel enable state
<i>TIMER_CCX_ENABLE</i>	channel enable
<i>TIMER_CCX_DISABLE</i>	channel disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 enable state */
timer_channel_output_state_config (TIMER0, TIMER_CH_0, TIMER_CCX_ENABLE);
```

### timer\_channel\_complementary\_output\_state\_config

The description of timer\_channel\_complementary\_output\_state\_config is shown as below:

**Table 3-540. 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>	
timer_periph	TIMER peripheral
<i>TIMERx(x=0, 14..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0, <i>TIMERx</i> (x=0, 14..16)
<i>TIMER_CH_1</i>	TIMER channel 1, <i>TIMERx</i> (x=0)
<i>TIMER_CH_2</i>	TIMER channel 2, <i>TIMERx</i> (x=0)
<b>Input parameter{in}</b>	
state	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,
TICKER_CCXN_ENABLE);
```

### timer\_channel\_input\_struct\_para\_init

The description of timer\_channel\_input\_struct\_para\_init is shown as below:

**Table 3-541. 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 3-488. 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 3-542. 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
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<b>TIMER_CH_0</b>	TIMER channel 0(TIMERx (x=0..2, 13..16))
<b>TIMER_CH_1</b>	TIMER channel 1(TIMERx (x=0..2, 14))
<b>TIMER_CH_2</b>	TIMER channel 2(TIMERx (x=0..2))
<b>TIMER_CH_3</b>	TIMER channel 3(TIMERx (x=0..2))
<b>Input parameter{in}</b>	
<b>icpara</b>	TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-488. Structure timer_ic_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure TIMER0 input capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;
timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);

```

### **timer\_channel\_input\_capture\_prescaler\_config**

The description of `timer_channel_input_capture_prescaler_config` is shown as below:

**Table 3-543. 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
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<b>TIMER_CH_0</b>	TIMER channel 0(TIMERx (x=0..2, 13..16))
<b>TIMER_CH_1</b>	TIMER channel 1(TIMERx (x=0..2, 14))
<b>TIMER_CH_2</b>	TIMER channel 2(TIMERx (x=0..2))
<b>TIMER_CH_3</b>	TIMER channel 3(TIMERx (x=0..2))
<b>Input parameter{in}</b>	
<b>prescaler</b>	channel input capture prescaler value
<b>TIMER_IC_PSC_DIV1</b>	no prescaler
<b>TIMER_IC_PSC_DIV2</b>	divided by 2
<b>TIMER_IC_PSC_DIV4</b>	divided by 4
<b>TIMER_IC_PSC_DIV8</b>	divided by 8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

/\* configure TIMER0 channel 0 input capture prescaler value \*/

```
timer_channel_input_capture_prescaler_config (TIMER0, TIMER_CH_0,
    TIMER_IC_PSC_DIV2);
```

### **timer\_channel\_capture\_value\_register\_read**

The description of timer\_channel\_capture\_value\_register\_read is shown as below:

**Table 3-544. 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>	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
<b>Input parameter{in}</b>	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0(TIMERx (x=0..2, 13..16))
TIMER_CH_1	TIMER channel 1(TIMERx (x=0..2, 14))
TIMER_CH_2	TIMER channel 2(TIMERx (x=0..2))
TIMER_CH_3	TIMER channel 3(TIMERx (x=0..2))
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	channel capture compare register value (0~65535)

Example:

```
/* read TIMER0 channel 0 capture compare register value */

uint32_t ch0_value = 0;

ch0_value = timer_channel_capture_value_register_read (TIMER0, TIMER_CH_0);
```

### **timer\_input\_pwm\_capture\_config**

The description of timer\_input\_pwm\_capture\_config is shown as below:

**Table 3-545. Function timer\_input\_pwm\_capture\_config**

<b>Function name</b>	timer_input_pwm_capture_config
<b>Function prototype</b>	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
<b>Function descriptions</b>	configure TIMER input pwm capture function

<b>Precondition</b>	-
<b>The called functions</b>	timer_channel_input_capture_prescaler_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 14)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0
<i>TIMER_CH_1</i>	TIMER channel 1
<b>Input parameter{in}</b>	
<b>icpwm</b>	TIMER channel input pwm parameter struct, the structure members can refer to <a href="#">Table 3-488. Structure timer_ic_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 input pwm capture parameter */
timer_ic_parameter_struct timer_icinitpara;
timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;
timer_input_pwm_capture_config (TIMER0, TIMER_CH_0, &timer_icinitpara);
```

### timer\_hall\_mode\_config

The description of timer\_hall\_mode\_config is shown as below:

**Table 3-546. 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(x=0..2)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>hallmode</b>	TIMER hall sensor mode state

<i>TIMER_HALLINTERFA_CE_ENABLE</i>	TIMER hall sensor mode enable
<i>TIMER_HALLINTERFA_CE_DISABLE</i>	TIMER hall sensor mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 hall sensor mode */
timer_hall_mode_config (TIMER0, TIMER_HALLINTERFACE_ENABLE);
```

### **timer\_input\_trigger\_source\_select**

The description of timer\_input\_trigger\_source\_select is shown as below:

**Table 3-547. 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>	
<i>timer_periph</i>	TIMER peripheral
<i>TIMERx(x=0..2, 14)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<i>intrigger</i>	trigger selection
<i>TIMER_SMCFG_TRGS_EL_ITI0</i>	Internal trigger input 0(ITI0, TIMERx(x=0..2, 14))
<i>TIMER_SMCFG_TRGS_EL_ITI1</i>	Internal trigger input 0 (ITI1, TIMERx(x=0..2, 14))
<i>TIMER_SMCFG_TRGS_EL_ITI2</i>	Internal trigger input 0 (ITI2, TIMERx(ITI2, TIMERx(x=0..2)))
<i>TIMER_SMCFG_TRGS_EL_ITI3</i>	Internal trigger input 0(ITI3, TIMERx(x=0..2, 14))
<i>TIMER_SMCFG_TRGS_EL_CI0F_ED</i>	CI0 edge flag (CI0F_ED, TIMERx(x=0..2, 14))
<i>TIMER_SMCFG_TRGS_EL_CI0FE0</i>	channel 0 input Filtered output(CI0FE0, TIMERx(x=0..2, 14))
<i>TIMER_SMCFG_TRGS_EL_CI1FE1</i>	channel 1 input Filtered output(CI1FE1, TIMERx(x=0..2, 14))

<i>TIMER_SMCFG_TRGS_EL_ETIIP</i>	External trigger input filter output(ETIIP, TIMERx(x=0..2))
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select TIMER0 input trigger source */
timer_input_trigger_source_select (TIMER0, TIMER_SMCFG_TRGSEL_ITIO);
```

### **timer\_master\_output\_trigger\_source\_select**

The description of timer\_master\_output\_trigger\_source\_select is shown as below:

**Table 3-548. 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>	
<i>timer_periph</i>	TIMER peripheral
<i>TIMERx(x=0..2, 5, 14)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<i>outrigger</i>	master mode control
<i>TIMER_TRI_OUT_SRC_RESET</i>	Reset. When the UPG bit in the TIMERx_SWEVG register is set or a reset is generated by the slave mode controller, a TRGO pulse occurs. And in the latter case, the signal on TRGO is delayed compared to the actual reset
<i>TIMER_TRI_OUT_SRC_ENABLE</i>	Enable. This mode is useful to start several timers at the same time or to control a window in which a slave timer is enabled. In this mode the master mode controller selects the counter enable signal as TRGO. The counter enable signal is set when CEN control bit is set or the trigger input in pause mode is high. There is a delay between the trigger input in pause mode and the TRGO output, except if the master-slave mode is selected.
<i>TIMER_TRI_OUT_SRC_UPDATE</i>	Update. In this mode the master mode controller selects the update event as TRGO.
<i>TIMER_TRI_OUT_SRC_CH0</i>	Capture/compare pulse. In this mode the master mode controller generates a TRGO pulse when a capture or a compare match occurred in channel 0.
<i>TIMER_TRI_OUT_SRC_O0CPRE</i>	Compare. In this mode the master mode controller selects the O0CPRE signal is used as TRGO.

<i>TIMER_TRI_OUT_SRC_O1CPRE</i>	Compare. In this mode the master mode controller selects the O1CPRE signal is used as TRGO.
<i>TIMER_TRI_OUT_SRC_O2CPRE</i>	Compare. In this mode the master mode controller selects the O2CPRE signal is used as TRGO.
<i>TIMER_TRI_OUT_SRC_O3CPRE</i>	Compare. In this mode the master mode controller selects the O3CPRE signal is used as TRGO.
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select TIMER0 master mode output trigger source */
timer_master_output_trigger_source_select (TIMER0, TIMER_TRI_OUT_SRC_RESET);
```

### timer\_slave\_mode\_select

The description of timer\_slave\_mode\_select is shown as below:

**Table 3-549. Function timer\_slave\_mode\_select**

<b>Function name</b>	timer_slave_mode_select
<b>Function prototype</b>	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
<b>Function descriptions</b>	select TIMER slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 14)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>slavemode</b>	slave mode
<i>TIMER_SLAVE_MODE_DISABLE</i>	slave mode disable, TIMERx(x=0..2, 14)
<i>TIMER_ENCODER_MODE_DE0</i>	encoder mode 0, TIMERx(x=0..2)
<i>TIMER_ENCODER_MODE_DE1</i>	encoder mode 1, TIMERx(x=0..2)
<i>TIMER_ENCODER_MODE_DE2</i>	encoder mode 2, TIMERx(x=0..2)
<i>TIMER_SLAVE_MODE_RESTART</i>	restart mode, TIMERx(x=0..2, 14)
<i>TIMER_SLAVE_MODE_PAUSE</i>	pause mode, TIMERx(x=0..2, 14)
<i>TIMER_SLAVE_MODE_</i>	event mode, TIMERx(x=0..2, 14)

EVENT	
<i>TIMER_SLAVE_MODE_EXTERNAL0</i>	external clock mode 0, TIMERx(x=0..2, 14)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select TIMER0 slave mode */

timer_slave_mode_select (TIMER0, TIMER_ENCODER_MODE0);
```

### **timer\_master\_slave\_mode\_config**

The description of timer\_master\_slave\_mode\_config is shown as below:

**Table 3-550. Function timer\_master\_slave\_mode\_config**

Function name	timer_master_slave_mode_config
<b>Function prototype</b>	void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t masterslave);
<b>Function descriptions</b>	configure TIMER master slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
TIMERx(x=0..2, 14)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>masterslave</b>	master slave mode state
<i>TIMER_MASTER_SLAVE_MODE_ENABLE</i>	master slave mode enable
<i>TIMER_MASTER_SLAVE_MODE_DISABLE</i>	master slave mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 master slave mode */

timer_master_slave_mode_config (TIMER0, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

### timer\_external\_trigger\_config

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

**Table 3-551. Function timer\_external\_trigger\_config**

<b>Function name</b>	timer_external_trigger_config
<b>Function prototype</b>	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER external trigger input
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
<b>Input parameter{in}</b>	
<b>expolarity</b>	external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
<b>Input parameter{in}</b>	
<b>extfilter</b>	external trigger filter control (0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 external trigger input */

timer_external_trigger_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,
    TIMER_ETP_FALLING, 10);
```

### timer\_quadrature\_decoder\_mode\_config

The description of timer\_quadrature\_decoder\_mode\_config is shown as below:

**Table 3-552. Function timer\_quadrature\_decoder\_mode\_config**

<b>Function name</b>	timer_quadrature_decoder_mode_config
<b>Function prototype</b>	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity);
<b>Function descriptions</b>	configure TIMER quadrature decoder mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx(x=0..2)</b>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>decomode</b>	quadrature decoder mode
<b>TIMER_ENCODER_MODE0</b>	counter counts on CI0FE0 edge depending on CI1FE1 level
<b>TIMER_ENCODER_MODE1</b>	counter counts on CI1FE1 edge depending on CI0FE0 level
<b>TIMER_ENCODER_MODE2</b>	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
<b>TIMER_IC_POLARITY_RISING</b>	capture rising edge
<b>TIMER_IC_POLARITY_FALLING</b>	capture falling edge
<b>Input parameter{in}</b>	
<b>ic1polarity</b>	IC1 polarity
<b>TIMER_IC_POLARITY_RISING</b>	capture rising edge
<b>TIMER_IC_POLARITY_FALLING</b>	capture falling edge
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 quadrature decoder mode */
timer_quadrature_decoder_mode_config (TIMER0, TIMER_ENCODER_MODE0,
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

### **timer\_internal\_clock\_config**

The description of timer\_internal\_clock\_config is shown as below:

**Table 3-553. Function timer\_internal\_clock\_config**

<b>Function name</b>	timer_internal_clock_config
<b>Function prototype</b>	void timer_internal_clock_config(uint32_t timer_periph);
<b>Function descriptions</b>	configure TIMER internal clock mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
timer_periph	TIMER peripheral
TIMERx(x=0..2, 14)	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 internal clock mode */

timer_internal_clock_config (TIMER0);
```

### **timer\_internal\_trigger\_as\_external\_clock\_config**

The description of timer\_internal\_trigger\_as\_external\_clock\_config is shown as below:

**Table 3-554. Function timer\_internal\_trigger\_as\_external\_clock\_config**

<b>Function name</b>	timer_internal_trigger_as_external_clock_config
<b>Function prototype</b>	void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);
<b>Function descriptions</b>	configure TIMER the internal trigger as external clock input
<b>Precondition</b>	-
<b>The called functions</b>	timer_input_trigger_source_select
<b>Input parameter{in}</b>	
timer_periph	TIMER peripheral
TIMERx(x=0..2, 14)	TIMER peripheral selection
<b>Input parameter{in}</b>	
intrigger	trigger selection
TIMER_SMCFG_TRGS EL_ITI0	Internal trigger input 0 (ITI0), TIMERx(x=0..2, 14)
TIMER_SMCFG_TRGS EL_ITI1	Internal trigger input 0 (ITI1) , TIMERx(x=0..2, 14)
TIMER_SMCFG_TRGS EL_ITI2	Internal trigger input 0 (ITI2) , TIMERx(x=0..2, 14)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

--	--

Example:

```

/* configure TIMER0 the internal trigger ITI0 as external clock input */

timer_internal_trigger_as_external_clock_config (TIMER0, TIMER_SMCFG_TRGSEL_ITI0);

timer_external_trigger_as_external_clock_config
  
```

The description of `timer_external_trigger_as_external_clock_config` is shown as below:

**Table 3-555. Function `timer_external_trigger_as_external_clock_config`**

<b>Function name</b>	timer_external_trigger_as_external_clock_config
<b>Function prototype</b>	void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t exttrigger, uint16_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external trigger as external clock input
<b>Precondition</b>	-
<b>The called functions</b>	timer_input_trigger_source_select
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 14)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>exttrigger</b>	external trigger selection
<i>TIMER_SMCFG_TRGS_EL_CI0F_ED</i>	CI0 edge flag (CI0F_ED)
<i>TIMER_SMCFG_TRGS_EL_CI0FE0</i>	channel 0 input Filtered output (CI0FE0)
<i>TIMER_SMCFG_TRGS_EL_CI1FE1</i>	channel 1 input Filtered output (CI1FE1)
<b>Input parameter{in}</b>	
<b>expolarity</b>	external trigger polarity
<i>TIMER_IC_POLARITY_RISING</i>	active high or rising edge active
<i>TIMER_IC_POLARITY_FALLING</i>	active low or falling edge active
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	falling edge or rising edge active
<b>Input parameter{in}</b>	
<b>extfilter</b>	external trigger filter control (0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 the external trigger CI0FE0 as external clock input */
```

```
timer_external_trigger_as_external_clock_config (TIMER0,
    TIMER_SMCFG_TRGSEL_CI0FE0, TIMER_IC_POLARITY_RISING, 0);
```

### **timer\_external\_clock\_mode0\_config**

The description of timer\_external\_clock\_mode0\_config is shown as below:

**Table 3-556. 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
<b>TIMERx(x=0..2)</b>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	ETI external trigger prescaler
<b>TIMER_EXT_TRI_PSC_OFF</b>	no divided
<b>TIMER_EXT_TRI_PSC_DIV2</b>	divided by 2
<b>TIMER_EXT_TRI_PSC_DIV4</b>	divided by 4
<b>TIMER_EXT_TRI_PSC_DIV8</b>	divided by 8
<b>Input parameter{in}</b>	
<b>expolarity</b>	ETI external trigger polarity
<b>TIMER_ETP_FALLING</b>	active low or falling edge active
<b>TIMER_ETP_RISING</b>	active high or rising edge active
<b>Input parameter{in}</b>	
<b>extfilter</b>	ETI external trigger filter control (0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 the external clock mode0 */
```

```
timer_external_clock_mode0_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,
    TIMER_ETP_FALLING, 0);
```

### timer\_external\_clock\_mode1\_config

The description of timer\_external\_clock\_mode1\_config is shown as below:

**Table 3-557. Function timer\_external\_clock\_mode1\_config**

<b>Function name</b>	timer_external_clock_mode1_config
<b>Function prototype</b>	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external clock mode1
<b>Precondition</b>	-
<b>The called functions</b>	timer_external_trigger_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	ETI external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
<b>Input parameter{in}</b>	
<b>expolarity</b>	ETI external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
<b>Input parameter{in}</b>	
<b>extfilter</b>	ETI external trigger filter control (0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 the external clock mode1 */

timer_external_clock_mode1_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TIMER_ETP_FALLING, 0);
```

### timer\_external\_clock\_mode1\_disable

The description of timer\_external\_clock\_mode1\_disable is shown as below:

**Table 3-558. Function timer\_external\_clock\_mode1\_disable**

<b>Function name</b>	timer_external_clock_mode1_disable
<b>Function prototype</b>	void timer_external_clock_mode1_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER the external clock mode1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
timer_periph	TIMER peripheral
TIMERx(x=0..2)	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 the external clock mode1 */
timer_external_clock_mode1_disable (TIMER0);
```

### **timer\_channel\_remap\_config**

The description of timer\_channel\_remap\_config is shown as below:

**Table 3-559. 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>	
timer_periph	TIMER peripheral
TIMERx(x=13)	TIMER peripheral selection
<b>Input parameter{in}</b>	
remap	remap function selection
TIMER13_CI0_RMP_GP IO	timer13 channel 0 input is connected to GPIO(TIMER13_CH0)
TIMER13_CI0_RMP_RT CCLK	timer13 channel 0 input is connected to the RTCCLK
TIMER13_CI0_RMP_HX TAL_DIV32	timer13 channel 0 input is connected to HXTAL/32 clock
TIMER13_CI0_RMP_CK OUTSEL	timer13 channel 0 input is connected to CKOUTSEL
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* configure TIMER13 channel 0 input is connected to GPIO */
timer_channel_remap_config (TIMER13, TIMER13_CI0_RMP_GPIO);
```

### **timer\_write\_chxval\_register\_config**

The description of timer\_write\_chxval\_register\_config is shown as below:

**Table 3-560. Function timer\_write\_chxval\_register\_config**

<b>Function name</b>	timer_write_chxval_register_config
<b>Function prototype</b>	void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);
<b>Function descriptions</b>	configure TIMER write CHxVAL register selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..2, 13..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>ccsel</b>	write CHxVAL register selection
<i>TIMER_CHVSEL_DISA BLE</i>	no effect
<i>TIMER_CHVSEL_ENAB LE</i>	when write the CHxVAL register, if the write value is same as the CHxVAL value, the write access is ignored
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 write CHxVAL register selection */
timer_write_chxval_register_config(TIMER0, TIMER_CHVSEL_ENABLE);
```

### **timer\_output\_value\_selection\_config**

The description of timer\_output\_value\_selection\_config is shown as below:

**Table 3-561. Function timer\_output\_value\_selection\_config**

<b>Function name</b>	timer_output_value_selection_config
<b>Function prototype</b>	void timer_output_value_selection_config(uint32_t timer_periph, uint16_t outsel);

<b>Function descriptions</b>	configure TIMER output value selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx(x=0, 14..16)</b>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>outsel</b>	output value selection
<b>TIMER_OUTSEL_DISA BLE</b>	no effect
<b>TIMER_OUTSEL_ENAB LE</b>	if POEN and IOS is 0, the output disabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER output value selection */

timer_output_value_selection_config(TIMER0, TIMER_OUTSEL_ENABLE);
```

## 3.22. TSI

Touch Sensing Interface (TSI) provides a convenient solution for touch keys, sliders and capacitive proximity sensing applications. The TSI registers are listed in chapter [3.22.1](#), the TSI firmware functions are introduced in chapter [3.22.2](#).

### 3.22.1. Descriptions of Peripheral registers

TSI registers are listed in the table shown as below:

**Table 3-562. TSI Registers**

Registers	Descriptions
TSI_CTL0	Control register0
TSI_INTEN	Interrupt enable register
TSI_INTC	Interrupt flag clear register
TSI_INTF	Interrupt flag register
TSI_PHM	Pin hysteresis mode register
TSI_ASW	Analog switch register
TSI_SAMPCFG	Sample configuration register
TSI_CHCFG	Channel configuration register
TSI_GCTL	Group control register

Registers	Descriptions
TSI_GxCYCN (x=0..5)	Group x cycle number registers
TSI_CTL1	Control register1

### 3.22.2. Descriptions of Peripheral functions

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

**Table 3-563. TSI firmware function**

Function name	Function description
tsi_deinit	reset TSI peripheral
tsi_init	initialize TSI plus prescaler,charge plus,transfer plus,max cycle number
tsi_enable	enable TSI module
tsi_disable	disable TSI module
tsi_sample_pin_enable	enable sample pin
tsi_sample_pin_disable	disable sample pin
tsi_channel_pin_enable	enable channel pin
tsi_channel_pin_disable	disable channel pin
tsi_sofeware_mode_config	configure TSI triggering by software
tsi_software_start	start a charge-transfer sequence when TSI is in software trigger mode
tsi_software_stop	stop a charge-transfer sequence when TSI is in software trigger mode
tsi_hardware_mode_config	configure TSI triggering by hardware
tsi_pin_mode_config	configure TSI pin mode when charge-transfer sequence is IDLE
tsi_extend_charge_config	configure extend charge state
tsi_plus_config	configure charge plus and transfer plus
tsi_max_number_config	configure the max cycle number of a charge-transfer sequence
tsi_hysteresis_on	switch on hysteresis pin
tsi_hysteresis_off	switch off hysteresis pin
tsi_analog_on	switch on analog pin
tsi_analog_off	switch off analog pin
tsi_interrupt_enable	enable TSI interrupt
tsi_interrupt_disable	disable TSI interrupt
tsi_interrupt_flag_clear	Clear TSI interrupt flag
tsi_interrupt_flag_get	get TSI interrupt flag
tsi_flag_clear	clear flag
tsi_flag_get	get flag
tsi_group_enable	enable group

Function name	Function description
tsi_group_disable	disbale group
tsi_group_status_get	get group complete status
tsi_group0_cycle_get	get the cycle number for group0 as soon as a charge-transfer sequence completes
tsi_group1_cycle_get	get the cycle number for group1 as soon as a charge-transfer sequence completes
tsi_group2_cycle_get	get the cycle number for group2 as soon as a charge-transfer sequence completes
tsi_group3_cycle_get	get the cycle number for group3 as soon as a charge-transfer sequence completes
tsi_group4_cycle_get	get the cycle number for group4 as soon as a charge-transfer sequence completes
tsi_group5_cycle_get	get the cycle number for group5 as soon as a charge-transfer sequence completes

## tsi\_deinit

The description of tsi\_deinit is shown as below:

**Table 3-564. Function tsi\_deinit**

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

Example:

```
/* reset TSI*/
tsi_deinit();
```

## tsi\_init

The description of tsi\_init is shown as below:

**Table 3-565. Function tsi\_init**

Function name	tsi_init
Function prototype	void tsi_init(uint32_t prescaler,uint32_t charge_duration,uint32_t

	transfer_duration,uint32_t max_number);
<b>Function descriptions</b>	initialize TSI plus prescaler,charge plus,transfer plus,max cycle number
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>prescaler</b>	CTCLK clock division factor
<i>TSI_CTC DIV1</i>	$f_{CTCLK} = f_{HCLK}$
<i>TSI_CTC DIV2</i>	$f_{CTCLK} = f_{HCLK}/2$
<i>TSI_CTC DIV4</i>	$f_{CTCLK} = f_{HCLK}/4$
<i>TSI_CTC DIV8</i>	$f_{CTCLK} = f_{HCLK}/8$
<i>TSI_CTC DIV16</i>	$f_{CTCLK} = f_{HCLK}/16$
<i>TSI_CTC DIV32</i>	$f_{CTCLK} = f_{HCLK}/32$
<i>TSI_CTC DIV64</i>	$f_{CTCLK} = f_{HCLK}/64$
<i>TSI_CTC DIV128</i>	$f_{CTCLK} = f_{HCLK}/128$
<i>TSI_CTC DIV256</i>	$f_{CTCLK} = f_{HCLK}/256$
<i>TSI_CTC DIV512</i>	$f_{CTCLK} = f_{HCLK}/512$
<i>TSI_CTC DIV1024</i>	$f_{CTCLK} = f_{HCLK}/1024$
<i>TSI_CTC DIV2048</i>	$f_{CTCLK} = f_{HCLK}/2048$
<i>TSI_CTC DIV4096</i>	$f_{CTCLK} = f_{HCLK}/4096$
<i>TSI_CTC DIV8192</i>	$f_{CTCLK} = f_{HCLK}/8192$
<i>TSI_CTC DIV16384</i> 4	$f_{CTCLK} = f_{HCLK}/16384$
<i>TSI_CTC DIV32768</i> 8	$f_{CTCLK} = f_{HCLK}/32768$
<b>Input parameter{in}</b>	
<b>charge_duration</b>	charge state duration time
<i>TSI_CHARGE_1CTCL</i> $K(x=1..16)$	the duration time of charge state is x CTCLK
<b>Input parameter{in}</b>	
<b>transfer_duration</b>	charge transfer state duration time
<i>TSI_TRANSFER_xCTC</i> $LK(x=1..16)SS_CLEAR$	the duration time of transfer state is x CTCLK
<b>Input parameter{in}</b>	
<b>max_number</b>	max cycle number
<i>TSI_MAXNUM255</i>	the max cycle number of a sequence is 255
<i>TSI_MAXNUM511</i>	the max cycle number of a sequence is 511
<i>TSI_MAXNUM1023</i>	the max cycle number of a sequence is 1023
<i>TSI_MAXNUM2047</i>	the max cycle number of a sequence is 2047
<i>TSI_MAXNUM4095</i>	the max cycle number of a sequence is 4095
<i>TSI_MAXNUM8191</i>	the max cycle number of a sequence is 8191
<i>TSI_MAXNUM16383</i>	the max cycle number of a sequence is 16383
<b>Output parameter{out}</b>	

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

Example:

```
/* init TSI*/
tsi_init (TSI_CTCDIV_DIV4096, TSI_CHARGE_10CTCLK, TSI_TRANSFER_8CTCLK,
TSI_MAXNUM511);
```

### **tsi\_enable**

The description of tsi\_enable is shown as below:

**Table 3-566. Function tsi\_enable**

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

Example:

```
/* enable TSI module */
tsi_enable();
```

### **tsi\_disable**

The description of tsi\_disable is shown as below:

**Table 3-567. Function tsi\_disable**

<b>Function name</b>	tsi_disable
<b>Function prototype</b>	void tsi_disable (void);
<b>Function descriptions</b>	disable TSI module
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* disable TSI module */

tsi_disable();
```

### **tsi\_sample\_pin\_enable**

The description of tsi\_sample\_pin\_enable is shown as below:

**Table 3-568. Function tsi\_sample\_pin\_enable**

<b>Function name</b>	tsi_sample_pin_enable
<b>Function prototype</b>	void tsi_sample_pin_enable(uint32_t sample);
<b>Function descriptions</b>	enable sample pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sample</b>	sample pin
<i>TSI_SAMPCFG_GxPy( x=0..5,y=0..3)</i>	pin y of group x is sample pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable G5P3 sample pin */

tsi_sample_pin_enable (TSI_SAMPCFG_G5P3);
```

### **tsi\_sample\_pin\_disable**

The description of tsi\_sample\_pin\_disable is shown as below:

**Table 3-569. Function tsi\_sample\_pin\_disable**

<b>Function name</b>	tsi_sample_pin_disable
<b>Function prototype</b>	void tsi_sample_pin_disable(uint32_t sample);
<b>Function descriptions</b>	disable sample pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sample</b>	sample pin
<i>TSI_SAMPCFG_GxPy( x=0..5,y=0..3)</i>	pin y of group x is sample pin

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable G5P3 sample pin */

tsi_sample_pin_disable (TSI_SAMPCFG_G5P3);
```

### tsi\_channel\_pin\_enable

The description of tsi\_channel\_pin\_enable is shown as below:

**Table 3-570. Function tsi\_channel\_pin\_enable**

Function name	tsi_channel_pin_enable
Function prototype	void tsi_channel_pin_enable(uint32_t channel);
Function descriptions	enable channel pin
Precondition	-
The called functions	-
Input parameter{in}	
channel	channel pin
TSI_CHCFG_GxPy( x=0..5,y=0..3)	pin y of group x
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable G5P3 channel pin */

tsi_channel_pin_enable (TSI_CHCFG_G5P3);
```

### tsi\_channel\_pin\_disable

The description of tsi\_channel\_pin\_disable is shown as below:

**Table 3-571. Function tsi\_channel\_pin\_disable**

Function name	tsi_channel_pin_disable
Function prototype	void tsi_channel_pin_disable(uint32_t channel);
Function descriptions	disable channel pin
Precondition	-
The called functions	-
Input parameter{in}	
channel	channel pin

<i>TSI_CHCFG_GxPy( x=0..5,y=0..3)</i>	pin y of group x
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable G5P3 channel pin */

tsi_channel_pin_disable (TSI_CHCFG_G5P3);
```

### **tsi\_sofeware\_mode\_config**

The description of tsi\_sofeware\_mode\_config is shown as below:

**Table 3-572. Function tsi\_sofeware\_mode\_config**

<b>Function name</b>	tsi_sofeware_mode_config
<b>Function prototype</b>	void tsi_sofeware_mode_config (void);
<b>Function descriptions</b>	configure TSI triggering by software
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TSI triggering by software */

tsi_sofeware_mode_config ();
```

### **tsi\_software\_start**

The description of tsi\_software\_start is shown as below:

**Table 3-573. Function tsi\_software\_start**

<b>Function name</b>	tsi_software_start
<b>Function prototype</b>	void tsi_software_start (void);
<b>Function descriptions</b>	start a charge-transfer sequence when TSI is in software trigger mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* start a charge-transfer sequence when TSI is in software trigger mode */

tsi_software_start();
```

### tsi\_software\_stop

The description of tsi\_software\_stop is shown as below:

**Table 3-574. Function tsi\_software\_stop**

Function name	tsi_software_stop
Function prototype	void tsi_software_stop (void);
Function descriptions	stop a charge-transfer sequence when TSI is in software trigger mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* stop a charge-transfer sequence when TSI is in software trigger mode */

tsi_software_stop();
```

### tsi\_hardware\_mode\_config

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

**Table 3-575. Function tsi\_hardware\_mode\_config**

Function name	tsi_hardware_mode_config
Function prototype	void tsi_hardware_mode_config(uint8_t trigger_edge);
Function descriptions	configure TSI triggering by hardware
Precondition	-
The called functions	-
Input parameter{in}	
trigger_edge	the edge type in hardware trigger mode
TSI_FALLING_TRIGGER R	falling edge trigger TSI charge transfer sequence

<i>TSI_RISING_TRIGGER</i>	rising edge trigger TSI charge transfer sequence
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TSI triggering by hardware */

tsi_hardware_mode_config (TSI_FALLING_TRIGGER);
```

### **tsi\_pin\_mode\_config**

The description of *tsi\_pin\_mode\_config* is shown as below:

**Table 3-576. Function *tsi\_pin\_mode\_config***

<b>Function name</b>	tsi_pin_mode_config
<b>Function prototype</b>	void tsi_pin_mode_config(uint8_t pin_mode);
<b>Function descriptions</b>	configure TSI pin mode when charge-transfer sequence is IDLE
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>pin_mode</i>	pin mode when charge-transfer sequence is IDLE
<i>TSI_OUTPUT_LOW</i>	TSI pin will output low when IDLE
<i>TSI_INPUT_FLOATING</i>	TSI pin will keep input_floating when IDLE
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TSI pin mode when charge-transfer sequence is IDLE */

tsi_pin_mode_config (TSI_OUTPUT_LOW);
```

### **tsi\_extend\_charge\_config**

The description of *tsi\_extend\_charge\_config* is shown as below:

**Table 3-577. Function *tsi\_extend\_charge\_config***

<b>Function name</b>	tsi_extend_charge_config
<b>Function prototype</b>	void tsi_extend_charge_config(ControlStatus extend,uint8_t prescaler,uint32_t max_duration);
<b>Function descriptions</b>	configure extend charge state
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>extend</b>	enable or disable extend charge state
<b>ENABLE</b>	enable extend charge state
<b>DISABLE</b>	disable extend charge state
Input parameter{in}	
<b>prescaler</b>	ECCLK clock division factor
<i>TSI_EXTEND_DIV1</i>	$f_{ECCLK} = f_{HCLK}$
<i>TSI_EXTEND_DIV2</i>	$f_{ECCLK} = f_{HCLK} / 2$
<i>TSI_EXTEND_DIV3</i>	$f_{ECCLK} = f_{HCLK} / 3$
<i>TSI_EXTEND_DIV4</i>	$f_{ECCLK} = f_{HCLK} / 4$
<i>TSI_EXTEND_DIV5</i>	$f_{ECCLK} = f_{HCLK} / 5$
<i>TSI_EXTEND_DIV6</i>	$f_{ECCLK} = f_{HCLK} / 6$
<i>TSI_EXTEND_DIV7</i>	$f_{ECCLK} = f_{HCLK} / 7$
<i>TSI_EXTEND_DIV8</i>	$f_{ECCLK} = f_{HCLK} / 8$
Input parameter{in}	
<b>max_duration</b>	extend charge state maximum duration time
<i>value range 1...128</i>	extend charge state maximum duration time is $1*t_{ECCLK} \sim 128*t_{ECCLK}$
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure extend charge state */
tsi_extend_charge_config (ENABLE, TSI_EXTEND_DIV2, 10);
```

### tsi\_plus\_config

The description of tsi\_plus\_config is shown as below:

**Table 3-578. Function tsi\_plus\_config**

Function name	tsi_plus_config
<b>Function prototype</b>	void tsi_plus_config(uint32_t prescaler,uint32_t charge_duration,uint32_t transfer_duration);
<b>Function descriptions</b>	configure charge plus and transfer plus
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>prescaler</b>	CTCLK clock division factor
<i>TSI_CTCDIV_DIV1</i>	$f_{CTCLK} = f_{HCLK}$
<i>TSI_CTCDIV_DIV2</i>	$f_{CTCLK} = f_{HCLK} / 2$
<i>TSI_CTCDIV_DIV4</i>	$f_{CTCLK} = f_{HCLK} / 4$
<i>TSI_CTCDIV_DIV8</i>	$f_{CTCLK} = f_{HCLK} / 8$

<i>TSI_CTCDIV_DIV16</i>	$f_{CTCLK} = f_{HCLK} / 16$
<i>TSI_CTCDIV_DIV32</i>	$f_{CTCLK} = f_{HCLK} / 32$
<i>TSI_CTCDIV_DIV64</i>	$f_{CTCLK} = f_{HCLK} / 64$
<i>TSI_CTCDIV_DIV128</i>	$f_{CTCLK} = f_{HCLK} / 128$
<i>TSI_CTCDIV_DIV256</i>	$f_{CTCLK} = f_{HCLK} / 256$
<i>TSI_CTCDIV_DIV512</i>	$f_{CTCLK} = f_{HCLK} / 512$
<i>TSI_CTCDIV_DIV1024</i>	$f_{CTCLK} = f_{HCLK} / 1024$
<i>TSI_CTCDIV_DIV2048</i>	$f_{CTCLK} = f_{HCLK} / 2048$
<i>TSI_CTCDIV_DIV4096</i>	$f_{CTCLK} = f_{HCLK} / 4096$
<i>TSI_CTCDIV_DIV8192</i>	$f_{CTCLK} = f_{HCLK} / 8192$
<i>TSI_CTCDIV_DIV16384</i> 4	$f_{CTCLK} = f_{HCLK} / 16384$
<i>TSI_CTCDIV_DIV32768</i> 8	$f_{CTCLK} = f_{HCLK} / 32768$
<b>Input parameter{in}</b>	
<b>charge_duration</b>	charge state duration time
<i>TSI_CHARGE_1CTCL</i> $K(x=1..16)$	the duration time of charge state is x CTCLK
<b>Input parameter{in}</b>	
<b>transfer_duration</b>	charge transfer state duration time
<i>TSI_TRANSFER_xCTC</i> $LK(x=1..16)$	the duration time of transfer state is x CTCLK
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure charge plus and transfer plus */
tsi_plus_config (TSI_CTCDIV_DIV1024, TSI_CHARGE_10CTCLK,
TSI_TRANSFER_8CTCLK);
```

### **tsi\_max\_number\_config**

The description of tsi\_max\_number\_config is shown as below:

**Table 3-579. Function tsi\_max\_number\_config**

<b>Function name</b>	tsi_max_number_config
<b>Function prototype</b>	void tsi_max_number_config(uint32_t max_number);
<b>Function descriptions</b>	configure the max cycle number of a charge-transfer sequence
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>max_number</b>	max cycle number
<i>TSI_MAXNUM255</i>	the max cycle number of a sequence is 255
<i>TSI_MAXNUM511</i>	the max cycle number of a sequence is 511
<i>TSI_MAXNUM1023</i>	the max cycle number of a sequence is 1023
<i>TSI_MAXNUM2047</i>	the max cycle number of a sequence is 2047
<i>TSI_MAXNUM4095</i>	the max cycle number of a sequence is 4095
<i>TSI_MAXNUM8191</i>	the max cycle number of a sequence is 8191
<i>TSI_MAXNUM16383</i>	the max cycle number of a sequence is 16383
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the max cycle number of a charge-transfer sequence */
tsi_max_number_config (TSI_MAXNUM1023);
```

### **tsi\_hysteresis\_on**

The description of tsi\_hysteresis\_on is shown as below:

**Table 3-580. Function tsi\_hysteresis\_on**

<b>Function name</b>	tsi_hysteresis_on
<b>Function prototype</b>	void tsi_hysteresis_on(uint32_t group_pin);
<b>Function descriptions</b>	switch on hysteresis pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>group_pin</b>	select pin which will be switched on hysteresis
<i>TSI_PHM_GxPy(x=0..5, y=0..3)</i>	pin y of group x switch on hysteresis
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* switch on hysteresis pin */
tsi_hysteresis_on (TSI_PHM_G5P3);
```

### **tsi\_hysteresis\_off**

The description of tsi\_hysteresis\_off is shown as below:

**Table 3-581. Function tsi\_hysteresis\_off**

<b>Function name</b>	tsi_hysteresis_off
<b>Function prototype</b>	void tsi_hysteresis_off(uint32_t group_pin);
<b>Function descriptions</b>	switch off hysteresis pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
group_pin	select pin which will be switched off hysteresis
TSI_PHM_GxPy( $x=0..5$ , $y=0..3$ )	pin y of group x switch off hysteresis
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* switch off hysteresis pin */
tsi_hysteresis_off (TSI_PHM_G5P3);
```

### tsi\_analog\_on

The description of tsi\_analog\_on is shown as below:

**Table 3-582. Function tsi\_analog\_on**

<b>Function name</b>	tsi_analog_on
<b>Function prototype</b>	void tsi_analog_on(uint32_t group_pin);
<b>Function descriptions</b>	switch on analog pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
group_pin	select pin which will be switched on analog
TSI_ASW_GxPy ( $x=0..5,y=0..3$ )	pin y of group x switch on analog
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* switch on analog pin */
tsi_analog_on (TSI_ASW_G5P3);
```

### tsi\_analog\_off

The description of tsi\_analog\_off is shown as below:

**Table 3-583. Function tsi\_analog\_off**

<b>Function name</b>	tsi_analog_off
<b>Function prototype</b>	void tsi_analog_off(uint32_t group_pin);
<b>Function descriptions</b>	switch off analog pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>group_pin</b>	select pin which will be switched off analog
<i>TSI_ASW_GxPy (x=0..5,y=0..3)</i>	pin y of group x switch off analog
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* switch off analog pin */

tsi_analog_off (TSI_ASW_G5P3);
```

### tsi\_interrupt\_enable

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

**Table 3-584. Function tsi\_interrupt\_enable**

<b>Function name</b>	tsi_interrupt_enable
<b>Function prototype</b>	void tsi_interrupt_enable(uint32_t source);
<b>Function descriptions</b>	enable TSI interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	select interrupt which will be enabled
<i>TSI_INT_CCTCF</i>	charge-transfer complete flag interrupt enable
<i>TSI_INT_MNERR</i>	max cycle number error interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TSI_INT_CCTCF interrupt */
```

```
tsi_interrupt_enable(TSI_INT_CCTCF);
```

### **tsi\_interrupt\_disable**

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

**Table 3-585. Function tsi\_interrupt\_disable**

<b>Function name</b>	tsi_interrupt_disable
<b>Function prototype</b>	void tsi_interrupt_disable(uint32_t source);
<b>Function descriptions</b>	disable TSI interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	select interrupt which will be disabled
<i>TSI_INT_CCTCF</i>	charge-transfer complete flag interrupt disable
<i>TSI_INT_MNERR</i>	max cycle number error interrupt disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TSI_INT_CCTCF interrupt */

tsi_interrupt_disable(TSI_INT_CCTCF);
```

### **tsi\_interrupt\_flag\_clear**

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

**Table 3-586. Function tsi\_interrupt\_flag\_clear**

<b>Function name</b>	tsi_interrupt_flag_clear
<b>Function prototype</b>	void tsi_interrupt_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear TSI interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	select flag which will be cleared
<i>TSI_INT_FLAG_CTCF</i>	clear charge-transfer complete flag
<i>TSI_INT_FLAG_MNER</i> <i>R</i>	clear max cycle number error
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TSI_INT_FLAG_CTCF interrupt flag */

tsi_interrupt_flag_clear (TSI_INT_FLAG_CTCF);
```

### **tsi\_interrupt\_flag\_get**

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

**Table 3-587. Function tsi\_interrupt\_flag\_get**

<b>Function name</b>	tsi_interrupt_flag_get
<b>Function prototype</b>	FlagStatus tsi_interrupt_flag_get(uint32_t flag);
<b>Function descriptions</b>	get TSI interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag
<i>TSI_INT_FLAG_CTCF</i>	charge-transfer complete flag
<i>TSI_INT_FLAG_MNER R</i>	max Cycle Number Error
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TSI_INT_FLAG_CTCF interrupt flag */

FlagStatus flag = RESET;

flag = tsi_interrupt_flag_get (TSI_INT_FLAG_CTCF);
```

### **tsi\_flag\_clear**

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

**Table 3-588. Function tsi\_flag\_clear**

<b>Function name</b>	tsi_flag_clear
<b>Function prototype</b>	void tsi_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	select flag which will be cleared
<i>TSI_FLAG_CTCF</i>	clear charge-transfer complete flag
<i>TSI_FLAG_MNERR</i>	clear max cycle number error

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear TSI_FLAG_CTCF_CLR flag */

tsi_flag_clear (TSI_FLAG_CTCF_CLR);
```

### tsi\_flag\_get

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

**Table 3-589. Function tsi\_flag\_get**

<b>Function name</b>	tsi_flag_get
<b>Function prototype</b>	FlagStatus tsi_flag_get(uint32_t flag);
<b>Function descriptions</b>	get flag
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>flag</b>	specify which flag
<i>TSI_FLAG_CTCF</i>	charge-transfer complete flag
<i>TSI_FLAG_MNERR</i>	max Cycle Number Error
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TSI_FLAG_CTCF flag */

FlagStatus flag = RESET;

flag = tsi_flag_get (TSI_FLAG_CTCF);
```

### tsi\_group\_enable

The description of tsi\_group\_enable is shown as below:

**Table 3-590. Function tsi\_group\_enable**

<b>Function name</b>	tsi_group_enable
<b>Function prototype</b>	void tsi_group_enable(uint32_t group);
<b>Function descriptions</b>	enable group
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>group</b>	select group to be enabled
<i>TSI_GCTL_GEx(x=0..5)</i>	the x group will be enabled
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable group 5 */
tsi_group_enable (TSI_GCTL_GE5);
```

### **tsi\_group\_disable**

The description of tsi\_group\_disable is shown as below:

**Table 3-591. Function tsi\_group\_disable**

<b>Function name</b>	tsi_group_disable
<b>Function prototype</b>	void tsi_group_disable(uint32_t group);
<b>Function descriptions</b>	disbale group
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>group</b>	select group to be disabled
<i>TSI_GCTL_GEx(x=0..5)</i>	the x group will be disabled
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable group 5 */
tsi_group_disable (TSI_GCTL_GE5);
```

### **tsi\_group\_status\_get**

The description of tsi\_group\_status\_get is shown as below:

**Table 3-592. Function tsi\_group\_status\_get**

<b>Function name</b>	tsi_group_status_get
<b>Function prototype</b>	FlagStatus tsi_group_status_get(uint32_t group);
<b>Function descriptions</b>	get group complete status
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>group</b>	select group
<i>TSI_GCTL_GCx(x=0..5)</i>	get the complete status of group x
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get group complete status */

FlagStatus flag = RESET;

flag = tsi_flag_get (TSI_GCTL_GC5);
```

### **tsi\_group0\_cycle\_get**

The description of tsi\_group0\_cycle\_get is shown as below:

**Table 3-593. Function tsi\_group0\_cycle\_get**

<b>Function name</b>	tsi_group0_cycle_get
<b>Function prototype</b>	uint16_t tsi_group0_cycle_get(void);
<b>Function descriptions</b>	get the cycle number for group0 as soon as a charge-transfer sequence completes
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
<b>uint16_t</b>	0-8192

Example:

```
/* get the cycle number for group0 as soon as a charge-transfer sequence completes */

uint16_t flag = 0;

flag = tsi_group0_cycle_get ();
```

### **tsi\_group1\_cycle\_get**

The description of tsi\_group1\_cycle\_get is shown as below:

**Table 3-594. Function tsi\_group1\_cycle\_get**

<b>Function name</b>	tsi_group1_cycle_get
<b>Function prototype</b>	uint16_t tsi_group1_cycle_get(void);

<b>Function descriptions</b>	get the cycle number for group1 as soon as a charge-transfer sequence completes
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	0-8192

Example:

```
/* get the cycle number for group1 as soon as a charge-transfer sequence completes */

uint16_t flag = 0;

flag = tsi_group1_cycle_get();
```

### **tsi\_group2\_cycle\_get**

The description of tsi\_group2\_cycle\_get is shown as below:

**Table 3-595. Function tsi\_group2\_cycle\_get**

<b>Function name</b>	tsi_group2_cycle_get
<b>Function prototype</b>	uint16_t tsi_group2_cycle_get(void);
<b>Function descriptions</b>	get the cycle number for group2 as soon as a charge-transfer sequence completes
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	0-8192

Example:

```
/* get the cycle number for group2 as soon as a charge-transfer sequence completes */

uint16_t flag = 0;

flag = tsi_group2_cycle_get();
```

### **tsi\_group3\_cycle\_get**

The description of tsi\_group3\_cycle\_get is shown as below:

**Table 3-596. Function tsi\_group3\_cycle\_get**

<b>Function name</b>	tsi_group3_cycle_get	
<b>Function prototype</b>	uint16_t tsi_group3_cycle_get(void);	
<b>Function descriptions</b>	get the cycle number for group3 as soon as a charge-transfer sequence completes	
<b>Precondition</b>	-	
<b>The called functions</b>	-	
<b>Input parameter{in}</b>		
-	-	
<b>Output parameter{out}</b>		
-	-	
<b>Return value</b>		
<b>uint16_t</b>	0-8192	

Example:

```
/* get the cycle number for group3 as soon as a charge-transfer sequence completes */

uint16_t flag = 0;

flag = tsi_group3_cycle_get();
```

### **tsi\_group4\_cycle\_get**

The description of tsi\_group4\_cycle\_get is shown as below:

**Table 3-597. Function tsi\_group4\_cycle\_get**

<b>Function name</b>	tsi_group4_cycle_get	
<b>Function prototype</b>	uint16_t tsi_group4_cycle_get(void);	
<b>Function descriptions</b>	get the cycle number for group4 as soon as a charge-transfer sequence completes	
<b>Precondition</b>	-	
<b>The called functions</b>	-	
<b>Input parameter{in}</b>		
-	-	
<b>Output parameter{out}</b>		
-	-	
<b>Return value</b>		
<b>uint16_t</b>	0-8192	

Example:

```
/* get the cycle number for group4 as soon as a charge-transfer sequence completes */

uint16_t flag = 0;

flag = tsi_group4_cycle_get();
```

### **tsi\_group5\_cycle\_get**

The description of tsi\_group5\_cycle\_get is shown as below:

**Table 3-598. Function tsi\_group5\_cycle\_get**

<b>Function name</b>	tsi_group5_cycle_get
<b>Function prototype</b>	uint16_t tsi_group5_cycle_get(void);
<b>Function descriptions</b>	get the cycle number for group5 as soon as a charge-transfer sequence completes
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	0-8192

Example:

```
/* get the cycle number for group5 as soon as a charge-transfer sequence completes */

uint16_t flag = 0;

flag = tsi_group5_cycle_get();
```

## **3.23. USART**

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter [3.23.1](#), the USART firmware functions are introduced in chapter [3.23.2](#).

### **3.23.1. Descriptions of Peripheral registers**

USART registers are listed in the table shown as below:

**Table 3-599. USART Registers**

<b>Registers</b>	<b>Descriptions</b>
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

Registers	Descriptions
USART_STAT	Status register
USART_INTC	Status clear register
USART_RDATA	Receive data register
USART_TDATA	Transmit data register
USART_RFCS	Receive FIFO control and status register

### 3.23.2. Descriptions of Peripheral functions

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

**Table 3-600. USART firmware function**

Function name	Function description
uart_deinit	reset USART
uart_baudrate_set	configure USART baud rate value
uart_parity_config	configure USART parity function
uart_word_length_set	configure USART word length
uart_stop_bit_set	configure USART stop bit length
uart_enable	enable USART
uart_disable	disable USART
uart_transmit_config	configure USART transmitter
uart_receive_config	configure USART receiver
uart_data_first_config	data is transmitted/received with the LSB/MSB first
uart_invert_config	configure USART inverted
uart_overrun_enable	enable the USART overrun function
uart_overrun_disable	disable the USART overrun function
uart_oversample_config	configure the USART oversample mode
uart_sample_bit_config	configure sample bit method
uart_receiver_timeout_enable	enable receiver timeout
uart_receiver_timeout_disable	disable receiver timeout
uart_receiver_timeout_threshold_config	configure receiver timeout threshold
uart_data_transmit	USART transmit data function
uart_data_receive	USART receive data function
uart_autobaud_detection_enable	enable auto baud rate detection
uart_autobaud_detection_disable	disable auto baud rate detection
uart_autobaud_detection_mode_config	configure auto baud rate detection mode
uart_address_config	configure the address of the USART in wake up by address match mode
uart_address_detection_mode_config	configure address detection mode
uart_mute_mode_enable	enable mute mode

Function name	Function description
uart_mute_mode_disable	disable mute mode
uart_mute_mode_wakeup_config	configure wakeup method in mute mode
uart_lin_mode_enable	enable LIN mode
uart_lin_mode_disable	disable LIN mode
uart_lin_break_detection_length_config	configure LIN break frame length
uart_halfduplex_enable	enable half duplex mode
uart_halfduplex_disable	disable half duplex mode
uart_clock_enable	enable USART clock
uart_clock_disable	disable USART clock
uart_synchronous_clock_config	configure USART synchronous mode parameters
uart_guard_time_config	configure guard time value in smartcard mode
uart_smartcard_mode_enable	enable smartcard mode
uart_smartcard_mode_disable	disable smartcard mode
uart_smartcard_mode_nack_enable	enable NACK in smartcard mode
uart_smartcard_mode_nack_disable	disable NACK in smartcard mode
uart_smartcard_mode_early_nack_enable	enable early NACK in smartcard mode
uart_smartcard_mode_early_nack_disable	disable early NACK in smartcard mode
uart_smartcard_autoretry_config	configure smartcard auto-retry number
uart_block_length_config	configure block length
uart_irda_mode_enable	enable IrDA mode
uart_irda_mode_disable	disable IrDA mode
uart_prescaler_config	configure the peripheral clock prescaler in USART IrDA low-power mode or SmartCard mode
uart_irda_lowpower_config	configure IrDA low-power
uart_hardware_flow_rts_config	configure hardware flow control RTS
uart_hardware_flow_cts_config	configure hardware flow control CTS
uart_rs485_driver_enable	enable RS485 driver
uart_rs485_driver_disable	disable RS485 driver
uart_driver_asserttime_config	configure driver enable assertion time
uart_driver_deasserttime_config	configure driver enable de-assertion time
uart_depolarity_config	configure driver enable polarity mode
uart_dma_receive_config	configure USART DMA for reception
uart_dma_transmit_config	configure USART DMA for transmission
uart_reception_error_dma_disable	disable DMA on reception error
uart_reception_error_dma_enable	enable DMA on reception error
uart_wakeup_enable	enable USART to wakeup the mcu from deep-sleep mode
uart_wakeup_disable	disable USART to wakeup the mcu from deep-sleep mode
uart_wakeup_mode_config	configure the USART wakeup mode from deep-sleep mode

Function name	Function description
uart_command_enable	enable USART command
uart_receive_fifo_enable	enable receive FIFO
uart_receive_fifo_disable	disable receive FIFO
uart_receive_fifo_counter_number	read receive FIFO counter number
uart_flag_get	get flag in STAT/RFCs register
uart_flag_clear	clear flag in STAT register
uart_interrupt_enable	enable USART interrupt
uart_interrupt_disable	disable USART interrupt
uart_interrupt_flag_get	get USART interrupt and flag status
uart_interrupt_flag_clear	clear USART interrupt flag

## Enum `uart_flag_enum`

**Table 3-601. Enum `uart_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_ABD	auto baudrate detection flag
USART_FLAG_ABDE	auto baudrate detection error
USART_FLAG_EB	end of block flag
USART_FLAG_RT	receiver timeout flag
USART_FLAG_CTS	CTS level
USART_FLAG_CTSF	CTS change flag
USART_FLAG_LBD	LIN break detected flag
USART_FLAG_TBE	transmit data buffer empty
USART_FLAG_TC	transmission complete
USART_FLAG_RBNE	read data buffer not empty
USART_FLAG_IDLE	IDLE line detected flag
USART_FLAG_ORERR	overrun error
USART_FLAG_NERR	noise error flag
USART_FLAG_FERR	frame error flag
USART_FLAG_PERR	parity error flag
USART_FLAG_RFF	receive FIFO full flag
USART_FLAG_RFE	receive FIFO empty flag

**Enum usart\_interrupt\_flag\_enum**
**Table 3-602. Enum usart\_interrupt\_flag\_enum**

Member name	Function description
USART_INT_FLAG_EB	end of block interrupt flag
USART_INT_FLAG_RT	receiver timeout interrupt flag
USART_INT_FLAG_AM	address match interrupt flag
USART_INT_FLAG_PERR	parity error interrupt flag
USART_INT_FLAG_TBE	transmitter buffer empty interrupt flag
USART_INT_FLAG_TC	transmission complete interrupt flag
USART_INT_FLAG_RBNE	read data buffer not empty interrupt flag
USART_INT_FLAG_RBNE_ORE RR	overrun error interrupt flag
USART_INT_FLAG_IDLE	IDLE line detected interrupt flag
USART_INT_FLAG_LBD	LIN break detected interrupt flag
USART_INT_FLAG_WU	wakeup from deep-sleep mode interrupt flag
USART_INT_FLAG_CTS	CTS interrupt flag
USART_INT_FLAG_ERR_NERR	noise error interrupt flag
USART_INT_FLAG_ERR_ORER R	overrun error interrupt flag
USART_INT_FLAG_ERR_FERR	frame error interrupt flag
USART_INT_FLAG_RFFINT	receive FIFO full interrupt flag

**Enum usart\_interrupt\_enum**
**Table 3-603. Enum usart\_interrupt\_enum**

Member name	Function description
USART_INT_EB	end of block interrupt
USART_INT_RT	receiver timeout interrupt
USART_INT_AM	address match interrupt
USART_INT_PERR	parity error interrupt
USART_INT_TBE	transmitter buffer empty interrupt
USART_INT_TC	transmission complete interrupt
USART_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt
USART_INT_IDLE	IDLE line detected interrupt
USART_INT_LBD	LIN break detected interrupt
USART_INT_WU	wakeup from deep-sleep mode interrupt
USART_INT_CTS	CTS interrupt
USART_INT_ERR	error interrupt
USART_INT_RFF	receive FIFO full interrupt

### Enum `uart_invert_enum`

**Table 3-604. Enum `uart_invert_enum`**

Member name	Function description
<code>USART_DINV_ENABLE</code>	data bit level inversion
<code>USART_DINV_DISABLE</code>	data bit level not inversion
<code>USART_TXPIN_ENABLE</code>	TX pin level inversion
<code>USART_TXPIN_DISABLE</code>	TX pin level not inversion
<code>USART_RXPIN_ENABLE</code>	RX pin level inversion
<code>USART_RXPIN_DISABLE</code>	RX pin level not inversion
<code>USART_SWAP_ENABLE</code>	swap TX/RX pins
<code>USART_SWAP_DISABLE</code>	not swap TX/RX pins

### `uart_deinit`

The description of `uart_deinit` is shown as below:

**Table 3-605. Function `uart_deinit`**

<b>Function name</b>	uart_deinit
<b>Function prototype</b>	<code>void usart_deinit(uint32_t usart_periph);</code>
<b>Function descriptions</b>	reset USART
<b>Precondition</b>	-
<b>The called functions</b>	<code>rcu_periph_reset_enable / rcu_periph_reset_disable</code>
<b>Input parameter{in}</b>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset USART0 */
usart_deinit(USART0);
```

### `uart_baudrate_set`

The description of `uart_baudrate_set` is shown as below:

**Table 3-606. Function `uart_baudrate_set`**

<b>Function name</b>	uart_baudrate_set
<b>Function prototype</b>	<code>void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);</code>
<b>Function descriptions</b>	configure USART baud rate value
<b>Precondition</b>	-

The called functions		rcu_clock_freq_get
Input parameter{in}		
uart_periph		uart peripheral
USARTx		x=0,1
Input parameter{in}		
baudval		baud rate value
Output parameter{out}		
-		-
Return value		
-		-

Example:

```
/* configure USART0 baud rate value */
uart_baudrate_set(USART0, 115200);
```

### uart\_parity\_config

The description of `uart_parity_config` is shown as below:

**Table 3-607. Function `uart_parity_config`**

Function name	uart_parity_config
Function prototype	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
Function descriptions	configure USART parity
Precondition	-
The called functions	-
Input parameter{in}	
uart_periph	uart peripheral
USARTx	x=0,1
Input parameter{in}	
paritycfg	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 */
uart_parity_config(USART0, USART_PM EVEN);
```

### uart\_word\_length\_set

The description of usart\_word\_length\_set is shown as below:

**Table 3-608. Function usart\_word\_length\_set**

<b>Function name</b>	usart_word_length_set
<b>Function prototype</b>	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);
<b>Function descriptions</b>	configure USART word length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>wlen</b>	USART word length configure
<b>USART_WL_8BIT</b>	8 bits
<b>USART_WL_9BIT</b>	9 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 word length */
usart_word_length_set(USART0, USART_WL_9BIT);
```

### uart\_stop\_bit\_set

The description of usart\_stop\_bit\_set is shown as below:

**Table 3-609. Function usart\_stop\_bit\_set**

<b>Function name</b>	usart_stop_bit_set
<b>Function prototype</b>	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);
<b>Function descriptions</b>	configure USART stop bit length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>stblen</b>	USART stop bit configure
<b>USART_STB_1BIT</b>	1 bit
<b>USART_STB_0_5BIT</b>	0.5 bit
<b>USART_STB_2BIT</b>	2 bits

<b>USART_STB_1_5BIT</b>	1.5 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 stop bit length */

uart_stop_bit_set(USART0, USART_STB_1_5BIT);
```

### uart\_enable

The description of `uart_enable` is shown as below:

**Table 3-610. Function `uart_enable`**

<b>Function name</b>	uart_enable
<b>Function prototype</b>	void uart_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>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 */

uart_enable(USART0);
```

### uart\_disable

The description of `uart_disable` is shown as below:

**Table 3-611. Function `uart_disable`**

<b>Function name</b>	uart_disable
<b>Function prototype</b>	void uart_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable USART
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral

<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 */

uart_disable(USART0);
```

### uart\_transmit\_config

The description of `uart_transmit_config` is shown as below:

**Table 3-612. Function `uart_transmit_config`**

<b>Function name</b>	uart_transmit_config
<b>Function prototype</b>	void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);
<b>Function descriptions</b>	configure USART transmitter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>txconfig</b>	enable or disable USART transmitter
<b>USART_TRANSMIT_ENABLE</b>	enable USART transmission
<b>USART_TRANSMIT_DISABLE</b>	disable USART transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 transmitter */

uart_transmit_config(USART0, USART_TRANSMIT_ENABLE);
```

### uart\_receive\_config

The description of `uart_receive_config` is shown as below:

**Table 3-613. Function `uart_receive_config`**

<b>Function name</b>	uart_receive_config
----------------------	---------------------

<b>Function prototype</b>	void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);
<b>Function descriptions</b>	configure USART receiver
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>rxconfig</b>	enable or disable USART receiver
<b>USART_RECEIVE_EN_ABLE</b>	enable USART reception
<b>USART_RECEIVE_DIS_ABLE</b>	disable USART reception
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-614. Function usart\_data\_first\_config**

<b>Function name</b>	usart_data_first_config
<b>Function prototype</b>	void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);
<b>Function descriptions</b>	data is transmitted/received with the LSB/MSB first
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>msbf</b>	LSB/MSB
<b>USART_MSBF_LSB</b>	LSB first
<b>USART_MSBF_MSB</b>	MSB first
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LSB of data first */

uart_data_first_config(USART0, USART_MSBF_LSB);
```

### **uart\_invert\_config**

The description of `uart_invert_config` is shown as below:

**Table 3-615. Function `uart_invert_config`**

<b>Function name</b>	uart_invert_config
<b>Function prototype</b>	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);
<b>Function descriptions</b>	USART inverted configure
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>invertpara</b>	refer to usart_invert_enum
<b>USART_DINV_ENABL_E</b>	data bit level inversion
<b>USART_DINV_DISABL_E</b>	data bit level not inversion
<b>USART_TXPIN_ENABL_E</b>	TX pin level inversion
<b>USART_TXPIN_DISABL_E</b>	TX pin level not inversion
<b>USART_RXPIN_ENABL_E</b>	RX pin level inversion
<b>USART_RXPIN_DISABL_E</b>	RX pin level not inversion
<b>USART_SWAP_ENABL_E</b>	swap TX/RX pins
<b>USART_SWAP_DISABL_E</b>	not swap TX/RX pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 inversion */
```

```
uart_invert_config(USART0, USART_DINV_ENABLE);
```

### **uart\_overrun\_enable**

The description of `uart_overrun_enable` is shown as below:

**Table 3-616. Function `uart_overrun_enable`**

<b>Function name</b>	uart_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>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 overrun */

uart_overrun_enable(USART0);
```

### **uart\_overrun\_disable**

The description of `uart_overrun_disable` is shown as below:

**Table 3-617. Function `uart_overrun_disable`**

<b>Function name</b>	uart_overrun_disable
<b>Function prototype</b>	void usart_overrun_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable the USART overrun function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 overrun */
```

```
uart_overrun_disableUSART0);
```

### **uart\_oversample\_config**

The description of `uart_oversample_config` is shown as below:

**Table 3-618. Function `uart_oversample_config`**

<b>Function name</b>	uart_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>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>oversamp</b>	oversample value
<b>USART_OVSMOD_8</b>	oversampling by 8
<b>USART_OVSMOD_16</b>	oversampling by 16
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 oversampling by 8 */
uart_oversample_configUSART0,USART_OVSMOD_8);
```

### **uart\_sample\_bit\_config**

The description of `uart_sample_bit_config` is shown as below:

**Table 3-619. Function `uart_sample_bit_config`**

<b>Function name</b>	uart_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>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>osb</b>	sample bit
<b>USART_OSB_1BIT</b>	1 bit

<b>USART_OSB_3BIT</b>	3 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config USART0 1 bit sample mode */

uart_sample_bit_config(USART0, USART_OSB_1BIT);
```

### **uart\_receiver\_timeout\_enable**

The description of `uart_receiver_timeout_enable` is shown as below:

**Table 3-620. Function `uart_receiver_timeout_enable`**

<b>Function name</b>	uart_receiver_timeout_enable
<b>Function prototype</b>	void uart_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>	uart peripheral
<b>USARTx</b>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 receiver timeout */

uart_receiver_timeout_enable(USART0);
```

### **uart\_receiver\_timeout\_disable**

The description of `uart_receiver_timeout_disable` is shown as below:

**Table 3-621. Function `uart_receiver_timeout_disable`**

<b>Function name</b>	uart_receiver_timeout_disable
<b>Function prototype</b>	void uart_receiver_timeout_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable receiver timeout
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral

USARTx	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 receiver timeout */

uart_receiver_timeout_disable(USART0);
```

### **uart\_receiver\_timeout\_threshold\_config**

The description of `uart_receiver_timeout_threshold_config` is shown as below:

**Table 3-622. Function `uart_receiver_timeout_threshold_config`**

<b>Function name</b>	uart_receiver_timeout_threshold_config
<b>Function prototype</b>	void uart_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>	uart peripheral
USARTx	x=0
<b>Input parameter{in}</b>	
<b>rtimeout</b>	receiver timeout
0x00000000- 0x00FFFFFF	receiver timeout value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the receiver timeout threshold of USART0 */

uart_receiver_timeout_threshold_config(USART0, 115200*3);
```

### **uart\_data\_transmit**

The description of `uart_data_transmit` is shown as below:

**Table 3-623. Function `uart_data_transmit`**

<b>Function name</b>	uart_data_transmit
<b>Function prototype</b>	void uart_data_transmit(uint32_t usart_periph, uint32_t data);

<b>Function descriptions</b>	USART transmit data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>data</b>	data of transmission
<b>0-0x1FF</b>	data of transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 transmit data */

uart_data_transmit(USART0, 0xAA);
```

### **uart\_data\_receive**

The description of **uart\_data\_receive** is shown as below:

**Table 3-624. Function `uart_data_receive`**

<b>Function name</b>	uart_data_receive
<b>Function prototype</b>	void <code>uart_data_receive(uint32_t usart_periph);</code>
<b>Function descriptions</b>	USART receive data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	data of received (0-0x1FF)

Example:

```
/* USART0 receive data */

uint16_t temp;

temp = uart_data_receive(USART0);
```

## **uart\_autobaud\_detection\_enable**

The description of `uart_autobaud_detection_enable` is shown as below:

**Table 3-625. Function `uart_autobaud_detection_enable`**

<b>Function name</b>	uart_autobaud_detection_enable
<b>Function prototype</b>	void uart_autobaud_detection_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable auto baud rate detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
usart_periph	uart peripheral
USARTx	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 auto baud rate detection */
uart_autobaud_detection_enable(USART0);
```

## **uart\_autobaud\_detection\_disable**

The description of `uart_autobaud_detection_disable` is shown as below:

**Table 3-626. Function `uart_autobaud_detection_disable`**

<b>Function name</b>	uart_autobaud_detection_disable
<b>Function prototype</b>	void uart_autobaud_detection_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable auto baud rate detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
usart_periph	uart peripheral
USARTx	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 auto baud rate detection */
uart_autobaud_detection_disable(USART0);
```

### uart\_autobaud\_detection\_mode\_config

The description of usart\_autobaud\_detection\_mode\_config is shown as below:

**Table 3-627. Function usart\_autobaud\_detection\_mode\_config**

<b>Function name</b>	usart_autobaud_detection_mode_config
<b>Function prototype</b>	void usart_autobaud_detection_mode_config(uint32_t usart_periph, uint32_t abdmod);
<b>Function descriptions</b>	configure auto baud rate detection mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
<b>Input parameter{in}</b>	
<b>abdmod</b>	auto baud rate detection mode
<b>USART_ABDM_FTOR</b>	falling edge to rising edge measurement
<b>USART_ABDM_FTOF</b>	falling edge to falling edge measurement
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 auto baud rate detection mode */
usart_autobaud_detection_mode_config(USART0, USART_ABDM_FTOR);
```

### uart\_address\_config

The description of usart\_address\_config is shown as below:

**Table 3-628. Function usart\_address\_config**

<b>Function name</b>	usart_address_config
<b>Function prototype</b>	void usart_address_config(uint32_t usart_periph, uint8_t addr);
<b>Function descriptions</b>	configure the address of the USART terminal
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>addr</b>	address of USART
<b>0-0xFF</b>	address of USART
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* configure address of the USART0 */
uart_address_config(USART0, 0x00);
```

### **uart\_address\_detection\_mode\_config**

The description of `uart_address_detection_mode_config` is shown as below:

**Table 3-629. Function `uart_address_detection_mode_config`**

<b>Function name</b>	uart_address_detection_mode_config
<b>Function prototype</b>	void usart_address_detection_mode_config(uint32_t usart_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>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>addmod</b>	address detection mode
<b>USART_ADDM_4BIT</b>	4 bits
<b>USART_ADDM_FULLB IT</b>	full bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*configure address detection mode */
uart_address_config(USART0, USART_ADDM_4BIT);
```

### **uart\_mute\_mode\_enable**

The description of `uart_mute_mode_enable` is shown as below:

**Table 3-630. Function `uart_mute_mode_enable`**

<b>Function name</b>	uart_mute_mode_enable
<b>Function prototype</b>	void usart_mute_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable mute mode

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 receiver in mute mode */

uart_mute_mode_enable(USART0);
```

### **uart\_mute\_mode\_disable**

The description of **uart\_mute\_mode\_disable** is shown as below:

**Table 3-631. Function **uart\_mute\_mode\_disable****

<b>Function name</b>	uart_mute_mode_disable
<b>Function prototype</b>	void uart_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>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 receiver in mute mode */

uart_mute_mode_disable(USART0);
```

### **uart\_mute\_mode\_wakeup\_config**

The description of **uart\_mute\_mode\_wakeup\_config** is shown as below:

**Table 3-632. Function **uart\_mute\_mode\_wakeup\_config****

<b>Function name</b>	uart_mute_mode_wakeup_config
<b>Function prototype</b>	void uart_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>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>wmethod</b>	two methods be used to enter or exit the mute mode
<b>USART_WM_IDLE</b>	idle line
<b>USART_WM_ADDR</b>	address mask
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 wakeup method in mute mode */
uart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

### uart\_lin\_mode\_enable

The description of `uart_lin_mode_enable` is shown as below:

**Table 3-633. Function `uart_lin_mode_enable`**

<b>Function name</b>	uart_lin_mode_enable
<b>Function prototype</b>	void usart_lin_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable LIN mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 LIN mode enable */
uart_lin_mode_enable(USART0);
```

### **uart\_lin\_mode\_disable**

The description of usart\_lin\_mode\_disable is shown as below:

**Table 3-634. Function usart\_lin\_mode\_disable**

<b>Function name</b>	usart_lin_mode_disable
<b>Function prototype</b>	void usart_lin_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable LIN mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 LIN mode disable */

usart_lin_mode_disable(USART0);
```

### **uart\_lin\_break\_decton\_length\_config**

The description of usart\_lin\_break\_decton\_length\_config is shown as below:

**Table 3-635. Function usart\_lin\_break\_decton\_length\_config**

<b>Function name</b>	usart_lin_break_decton_length_config
<b>Function prototype</b>	void usart_lin_break_decton_length_config(uint32_t usart_periph, uint32_t lblen);
<b>Function descriptions</b>	configure lin break frame length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
<b>Input parameter{in}</b>	
<b>lblen</b>	LIN break detection length
<b>USART_LBLEN_10B</b>	10 bits
<b>USART_LBLEN_11B</b>	11 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LIN break frame length */

uart_lin_break_detection_length_config(USART0, USART_LBLEN_10B);
```

### **uart\_halfduplex\_enable**

The description of `uart_halfduplex_enable` is shown as below:

**Table 3-636. Function `uart_halfduplex_enable`**

<b>Function name</b>	uart_halfduplex_enable
<b>Function prototype</b>	void usart_halfduplex_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable half-duplex mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 half duplex mode*/

uart_halfduplex_enable(USART0);
```

### **uart\_halfduplex\_disable**

The description of `uart_halfduplex_disable` is shown as below:

**Table 3-637. Function `uart_halfduplex_disable`**

<b>Function name</b>	uart_halfduplex_disable
<b>Function prototype</b>	void usart_halfduplex_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable half-duplex mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 half duplex mode*/
usart_halfduplex_disable(USART0);
```

### **usart\_clock\_enable**

The description of usart\_clock\_enable is shown as below:

**Table 3-638. 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 USART clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
usart_periph	uart peripheral
USARTx	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 CK pin */
usart_synchronous_clock_enable(USART0);
```

### **usart\_clock\_disable**

The description of usart\_clock\_disable is shown as below:

**Table 3-639. Function usart\_clock\_disable**

<b>Function name</b>	usart_clock_disable
<b>Function prototype</b>	void usart_clock_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable USART clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
usart_periph	uart peripheral
USARTx	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 CK pin */

uart_synchronous_clock_disable(USART0);
```

### **uart\_synchronous\_clock\_config**

The description of `uart_synchronous_clock_config` is shown as below:

**Table 3-640. Function `uart_synchronous_clock_config`**

<b>Function name</b>	<code>uart_synchronous_clock_config</code>
<b>Function prototype</b>	<code>void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);</code>
<b>Function descriptions</b>	configure USART synchronous mode parameters
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0,1
<b>Input parameter{in}</b>	
<code>clen</code>	last bit clock pulse
<code>USART_CLEN_NONE</code>	clock pulse of the last data bit (MSB) is not output to the CK pin
<code>USART_CLEN_EN</code>	clock pulse of the last data bit (MSB) is output to the CK pin
<b>Input parameter{in}</b>	
<code>cph</code>	clock phase
<code>USART_CPH_1CK</code>	first clock transition is the first data capture edge
<code>USART_CPH_2CK</code>	second clock transition is the first data capture edge
<b>Input parameter{in}</b>	
<code>cpl</code>	clock polarity
<code>USART_CPL_LOW</code>	steady low value on CK pin
<code>USART_CPL_HIGH</code>	steady high value on CK pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 synchronous mode parameters */

uart_synchronous_clock_config(USART0,USART_CLEN_EN,USART_CPH_2CK,
USART_CPL_HIGH);
```

### **uart\_guard\_time\_config**

The description of `uart_guard_time_config` is shown as below:

**Table 3-641. 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>	
usart_periph	uart peripheral
USARTx	x=0
<b>Input parameter{in}</b>	
guat	guard time value
0-0x000000FF	guard time value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 guard time value in smartcard mode */
usart_guard_time_config(USART0, 0x0000 0055);
```

### usart\_smartcard\_mode\_enable

The description of usart\_smartcard\_mode\_enable is shown as below:

**Table 3-642. 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>	
usart_periph	uart peripheral
USARTx	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 smartcard mode enable */
usart_smartcard_mode_enable(USART0);
```

### **uart\_smartcard\_mode\_disable**

The description of `uart_smartcard_mode_disable` is shown as below:

**Table 3-643. Function `uart_smartcard_mode_disable`**

<b>Function name</b>	uart_smartcard_mode_disable
<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>	uart peripheral
<b>USARTx</b>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 smartcard mode disable */

uart_smartcard_mode_disable(USART0);
```

### **uart\_smartcard\_mode\_nack\_enable**

The description of `uart_smartcard_mode_nack_enable` is shown as below:

**Table 3-644. Function `uart_smartcard_mode_nack_enable`**

<b>Function name</b>	uart_smartcard_mode_nack_enable
<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>	uart peripheral
<b>USARTx</b>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 NACK in smartcard mode */

uart_smartcard_mode_nack_enable(USART0);
```

### **uart\_smartcard\_mode\_nack\_disable**

The description of `uart_smartcard_mode_nack_disable` is shown as below:

**Table 3-645. Function `uart_smartcard_mode_nack_disable`**

<b>Function name</b>	uart_smartcard_mode_nack_disable
<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>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 NACK in smartcard mode */

uart_smartcard_mode_nack_disable(USART0);
```

### **uart\_smartcard\_mode\_early\_nack\_enable**

The description of `uart_smartcard_mode_early_nack_enable` is shown as below:

**Table 3-646. Function `uart_smartcard_mode_early_nack_enable`**

<b>Function name</b>	uart_smartcard_mode_early_nack_enable
<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>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 early NACK in smartcard mode */

uart_smartcard_mode_early_nack_enable(USART0);
```

### **uart\_smartcard\_mode\_early\_nack\_disable**

The description of `uart_smartcard_mode_early_nack_disable` is shown as below:

**Table 3-647. Function `uart_smartcard_mode_early_nack_disable`**

<b>Function name</b>	uart_smartcard_mode_early_nack_disable
<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>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 early NACK in smartcard mode */

uart_smartcard_mode_early_nack_disable(USART0);
```

### **uart\_smartcard\_autoretry\_config**

The description of `uart_smartcard_autoretry_config` is shown as below:

**Table 3-648. Function `uart_smartcard_autoretry_config`**

<b>Function name</b>	uart_smartcard_autoretry_config
<b>Function prototype</b>	<code>void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrnum);</code>
<b>Function descriptions</b>	configure smartcard auto-retry number
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0
<b>Input parameter{in}</b>	
<code>scrnum</code>	smartcard auto-retry number
<code>0x00000000-</code> <code>0x00000007</code>	smartcard auto-retry number
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure smartcard auto-retry number */

uart_smartcard_autoretry_config(USART0, 0x00000007);
```

### **uart\_block\_length\_config**

The description of `uart_block_length_config` is shown as below:

**Table 3-649. Function `uart_block_length_config`**

<b>Function name</b>	uart_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>	
usart_periph	uart peripheral
USARTx	x=0
<b>Input parameter{in}</b>	
bl	block length
0-0x000000FF	block length
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure block length in Smartcard T=1 reception */

uart_block_length_config(USART0, 0x000000FF);
```

### **uart\_irda\_mode\_enable**

The description of `uart_irda_mode_enable` is shown as below:

**Table 3-650. Function `uart_irda_mode_enable`**

<b>Function name</b>	uart_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>	
usart_periph	uart peripheral
USARTx	x=0
<b>Output parameter{out}</b>	

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

Example:

```
/* enable USART0 IrDA mode */

uart_irda_mode_enable(USART0);
```

### **uart\_irda\_mode\_disable**

The description of `uart_irda_mode_disable` is shown as below:

**Table 3-651. Function `uart_irda_mode_disable`**

<b>Function name</b>	uart_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>	uart peripheral
<b>USARTx</b>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 IrDA mode */

uart_irda_mode_disable(USART0);
```

### **uart\_prescaler\_config**

The description of `uart_prescaler_config` is shown as below:

**Table 3-652. Function `uart_prescaler_config`**

<b>Function name</b>	uart_prescaler_config
<b>Function prototype</b>	void usart_prescaler_config(uint32_t usart_periph, uint8_t psc);
<b>Function descriptions</b>	configure the peripheral clock prescaler in USART IrDA low-power mode or SmartCard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0

Input parameter{in}	
<b>psc</b>	clock prescaler
0x00-0xFF	clock prescaler
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the USART0 peripheral clock prescaler in USART IrDA low-power mode */
uart_prescaler_config(USART0, 0x00);
```

### **uart\_irda\_lowpower\_config**

The description of `uart_irda_lowpower_config` is shown as below:

**Table 3-653. Function `uart_irda_lowpower_config`**

<b>Function name</b>	uart_irda_lowpower_config
<b>Function prototype</b>	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
<b>Function descriptions</b>	configure IrDA low-power
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
Input parameter{in}	
<b>irlp</b>	IrDA low-power or normal
<b>USART_IRLP_LOW</b>	low-power
<b>USART_IRLP_NORMAL</b>	normal
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 IrDA low-power */
uart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

### **uart\_hardware\_flow\_rts\_config**

The description of `uart_hardware_flow_rts_config` is shown as below:

**Table 3-654. 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>	
usart_periph	uart peripheral
USARTx	x=0,1
<b>Input parameter{in}</b>	
rtsconfig	enable or disable RTS
USART_RTS_ENABLE	enable RTS
USART_RTS_DISABLE	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 3-655. 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>	
usart_periph	uart peripheral
USARTx	x=0,1
<b>Input parameter{in}</b>	
ctsconfig	enable or disable CTS
USART_CTS_ENABLE	enable CTS
USART_CTS_DISABLE	disable CTS
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* configure USART0 hardware flow control CTS */
uart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

### **uart\_rs485\_driver\_enable**

The description of `uart_rs485_driver_enable` is shown as below:

**Table 3-656. Function `uart_rs485_driver_enable`**

<b>Function name</b>	uart_rs485_driver_enable
<b>Function prototype</b>	void uart_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>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 RS485 driver */
uart_rs485_driver_enable(USART0);
```

### **uart\_rs485\_driver\_disable**

The description of `uart_rs485_driver_disable` is shown as below:

**Table 3-657. Function `uart_rs485_driver_disable`**

<b>Function name</b>	uart_rs485_driver_disable
<b>Function prototype</b>	void uart_rs485_driver_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable USARTRS485 driver
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* disable USART0 RS485 driver */

uart_rs485_driver_disable (USART0);
```

### **uart\_driver\_assertime\_config**

The description of `uart_driver_assertime_config` is shown as below:

**Table 3-658. Function `uart_driver_assertime_config`**

<b>Function name</b>	uart_driver_assertime_config
<b>Function prototype</b>	void usart_driver_assertime_config(uint32_t usart_periph, uint32_t deatime);
<b>Function descriptions</b>	configure driver enable assertion time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>deatime</b>	driver enable assertion time
<b>0-0x0000001F</b>	driver enable assertion time
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set USART0 driver assertime */

uart_driver_assertime_config(USART0,0x0000001F);
```

### **uart\_driver\_deassertime\_config**

The description of `uart_driver_deassertime_config` is shown as below:

**Table 3-659. Function `uart_driver_deassertime_config`**

<b>Function name</b>	uart_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>	-

Input parameter{in}	
uart_periph	uart peripheral
USARTx	x=0,1
Input parameter{in}	
deatetime	driver enable de-assertion time
0x00000000-0x0000001F	driver enable de-assertion time
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set USART0 driver deassertime */
uart_driver_deassertime_config(USART0, 0x0000001F);
```

### uart\_depolarity\_config

The description of `uart_depolarity_config` is shown as below:

**Table 3-660. Function `uart_depolarity_config`**

<b>Function name</b>	uart_depolarity_config
<b>Function prototype</b>	void uart_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}	
uart_periph	uart peripheral
USARTx	x=0,1
Input parameter{in}	
dep	DE signal
USART_DEP_HIGH	DE signal is active high
USART_DEP_LOW	DE signal is active low
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure driver enable polarity mode */
uart_driver_depolarity_config(USART0, USART_DEP_HIGH);
```

### uart\_dma\_receive\_config

The description of usart\_dma\_receive\_config is shown as below:

**Table 3-661. 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 reception
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>dmacmd</b>	enable or disable DMA for reception
<b>USART_DENR_ENABL E</b>	DMA enable for reception
<b>USART_DENR_DISAB LE</b>	DMA disable for reception
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 DMA enable for reception */
usart_dma_receive_config(USART0, USART_DENR_ENABLE);
```

### uart\_dma\_transmit\_config

The description of usart\_dma\_transmit\_config is shown as below:

**Table 3-662. 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 transmission
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>dmacmd</b>	enable or disable DMA for transmission
<b>USART_DENT_ENABL</b>	DMA enable for transmission

<i>E</i>	
<i>USART_DENT_DISAB</i>	DMA disable for transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 DMA enable for transmission */

uart_dma_transmit_config(USART0, USART_DENT_ENABLE);
```

### **uart\_reception\_error\_dma\_disable**

The description of `uart_reception_error_dma_disable` is shown as below:

**Table 3-663. Function `uart_reception_error_dma_disable`**

<b>Function name</b>	uart_reception_error_dma_disable
<b>Function prototype</b>	void uart_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>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA on reception error */

uart_reception_error_dma_disable(USART0);
```

### **uart\_reception\_error\_dma\_enable**

The description of `uart_reception_error_dma_enable` is shown as below:

**Table 3-664. Function `uart_reception_error_dma_enable`**

<b>Function name</b>	uart_reception_error_dma_enable
<b>Function prototype</b>	void uart_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}	
usart_periph	uart peripheral
USARTx	x=0,1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMA on reception error */

uart_reception_error_dma_enable(USART0);
```

### uart\_wakeup\_enable

The description of `uart_wakeup_enable` is shown as below:

**Table 3-665. Function `uart_wakeup_enable`**

<b>Function name</b>	uart_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}	
usart_periph	uart peripheral
USARTx	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 wake up enable */

uart_wakeup_enable(USART0);
```

### uart\_wakeup\_disable

The description of `uart_wakeup_disable` is shown as below:

**Table 3-666. Function `uart_wakeup_disable`**

<b>Function name</b>	uart_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}	
uart_periph	uart peripheral
USARTx	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 wake up disable */

uart_wakeup_disable(USART0);
```

### uart\_wakeup\_mode\_config

The description of `uart_wakeup_mode_config` is shown as below:

**Table 3-667. Function `uart_wakeup_mode_config`**

Function name	uart_wakeup_mode_config
Function prototype	void uart_wakeup_mode_config(uint32_t usart_periph, uint32_t wum);
Function descriptions	wakeup mode from deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
uart_periph	uart peripheral
USARTx	x=0
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 */

uart_wakeup_mode_config(USART0, USART_WUM_ADDR);
```

### uart\_command\_enable

The description of `uart_command_enable` is shown as below:

**Table 3-668. 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>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>cmdtype</b>	USART interrupt flag
<b>USART_CMD_ABDCM</b> <i>D</i>	auto baudrate detection command
<b>USART_CMD_SBKCM</b> <i>D</i>	send break command
<b>USART_CMD_MMCMD</b>	mute mode command
<b>USART_CMD_RXFCM</b> <i>D</i>	receive data flush command
<b>USART_CMD_TXFCM</b> <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_ABDCMD);
```

### **usart\_receive\_fifo\_enable**

The description of usart\_receive\_fifo\_enable is shown as below:

**Table 3-669. Function usart\_receive\_fifo\_enable**

<b>Function name</b>	usart_receive_fifo_enable
<b>Function prototype</b>	void usart_receive_fifo_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable receive FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	

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

Example:

```
/* enable receive FIFO */

uart_receive_fifo_enable(USART0);
```

### **uart\_receive\_fifo\_disable**

The description of `uart_receive_fifo_disable` is shown as below:

**Table 3-670. Function `uart_receive_fifo_disable`**

<b>Function name</b>	uart_receive_fifo_disable
<b>Function prototype</b>	void uart_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>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable receive FIFO */

uart_receive_fifo_disable(USART0);
```

### **uart\_receive\_fifo\_counter\_number**

The description of `uart_receive_fifo_counter_number` is shown as below:

**Table 3-671. Function `uart_receive_fifo_counter_number`**

<b>Function name</b>	uart_receive_fifo_counter_number
<b>Function prototype</b>	uint8_t uart_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>	uart peripheral
<b>USARTx</b>	x=0,1
<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 3-672. 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/RFCs register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags
<b>USART_FLAG_PERR</b>	parity error flag
<b>USART_FLAG_FERR</b>	frame error flag
<b>USART_FLAG_NERR</b>	noise error flag
<b>USART_FLAG_ORERR</b>	overrun error
<b>USART_FLAG_IDLE</b>	idle line detected flag
<b>USART_FLAG_RBNE</b>	read data buffer not empty
<b>USART_FLAG_TC</b>	transmission completed
<b>USART_FLAG_TBE</b>	transmit data register empty
<b>USART_FLAG_LBD</b>	LIN break detected flag
<b>USART_FLAG_CTSF</b>	CTS change flag
<b>USART_FLAG_CTS</b>	CTS level
<b>USART_FLAG_RT</b>	receiver timeout flag
<b>USART_FLAG_EB</b>	end of block flag
<b>USART_FLAG_ABDE</b>	auto baudrate detection error
<b>USART_FLAG_ABD</b>	auto baudrate detection flag
<b>USART_FLAG_BSY</b>	busy flag
<b>USART_FLAG_AM</b>	address match flag
<b>USART_FLAG_SB</b>	send break flag

<i>USART_FLAG_RWU</i>	receiver wakeup from mute mode
<i>USART_FLAG_WU</i>	wakeup from deep-sleep mode flag
<i>USART_FLAG_TEA</i>	transmit enable acknowledge flag
<i>USART_FLAG_REA</i>	receive enable acknowledge flag
<i>USART_FLAG_EPERR</i>	early parity error flag
<i>USART_FLAG_RFE</i>	receive FIFO empty flag
<i>USART_FLAG_RFF</i>	receive FIFO full flag
<i>USART_FLAG_RFFINT</i>	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 3-673. Function usart\_flag\_clear**

<b>Function name</b>	usart_flag_clear
<b>Function prototype</b>	void usart_flag_clear(uint32_t usart_periph, usart_flag_enum flag);
<b>Function descriptions</b>	clear flag in STAT register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags
<i>USART_FLAG_PERR</i>	parity error flag
<i>USART_FLAG_FERR</i>	frame error flag
<i>USART_FLAG_NERR</i>	noise detected flag
<i>USART_FLAG_ORERR</i>	overrun error flag
<i>USART_FLAG_IDLE</i>	idle line detected flag
<i>USART_FLAG_TC</i>	transmission complete flag
<i>USART_FLAG_LBD</i>	LIN break detected flag
<i>USART_FLAG_CTSF</i>	CTS change flag
<i>USART_FLAG_RT</i>	receiver timeout flag

<b>USART_FLAG_EB</b>	end of block flag
<b>USART_FLAG_AM</b>	address match flag
<b>USART_FLAG_WU</b>	wakeup from deep-sleep mode flag
<b>USART_FLAG_EPERR</b>	early parity error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear USART0 flag */

uart_flag_clear(USART0,USART_FLAG_TC);
```

### uart\_interrupt\_enable

The description of `uart_interrupt_enable` is shown as below:

**Table 3-674. Function `uart_interrupt_enable`**

<b>Function name</b>	<code>uart_interrupt_enable</code>
<b>Function prototype</b>	<code>void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);</code>
<b>Function descriptions</b>	enable USART interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt type
<b>USART_INT_IDLE</b>	idle interrupt
<b>USART_INT_RBNE</b>	read data buffer not empty interrupt and overrun error interrupt enable interrupt
<b>USART_INT_TC</b>	transmission complete interrupt
<b>USART_INT_TBE</b>	transmit data register empty interrupt
<b>USART_INT_PERR</b>	parity error interrupt
<b>USART_INT_AM</b>	address match interrupt
<b>USART_INT_RT</b>	receiver timeout interrupt
<b>USART_INT_EB</b>	end of block interrupt
<b>USART_INT_LBD</b>	LIN break detection interrupt
<b>USART_INT_ERR</b>	error interrupt enable in multibuffer communication
<b>USART_INT_CTS</b>	CTS interrupt
<b>USART_INT_WU</b>	wakeup from deep-sleep mode interrupt
<b>USART_INT_RFF</b>	receive FIFO full interrupt enable

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 TBE interrupt */

uart_interrupt_enable(USART0, USART_INT_TBE);
```

### uart\_interrupt\_disable

The description of `uart_interrupt_disable` is shown as below:

**Table 3-675. Function `uart_interrupt_disable`**

Function name	uart_interrupt_disable
Function prototype	void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);
Function descriptions	disable USART interrupt
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	uart peripheral
USARTx	x=0,1
Input parameter{in}	
interrupt	USART interrupt flag
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:

```
/* disable USART0 TBE interrupt */

uart_interrupt_disable(USART0, USART_INT_TBE);
```

### **uart\_interrupt\_flag\_get**

The description of `uart_interrupt_flag_get` is shown as below:

**Table 3-676. Function `uart_interrupt_flag_get`**

<b>Function name</b>	uart_interrupt_flag_get
<b>Function prototype</b>	FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	get USART interrupt and flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag
<b>USART_INT_FLAG_EB</b>	end of block interrupt and flag
<b>USART_INT_FLAG_RT</b>	receiver timeout interrupt and flag
<b>USART_INT_FLAG_AM</b>	address match interrupt and flag
<b>USART_INT_FLAG_PERR</b>	parity error interrupt and flag
<b>USART_INT_FLAG_TE</b>	transmitter buffer empty interrupt and flag
<b>USART_INT_FLAG_TC</b>	transmission complete interrupt and flag
<b>USART_INT_FLAG_RBNE</b>	read data buffer not empty interrupt and flag
<b>USART_INT_FLAG_RBNE_ORERR</b>	read data buffer not empty interrupt and overrun error flag
<b>USART_INT_FLAG_IDLE</b>	IDLE line detected interrupt and flag
<b>USART_INT_FLAG_LBD</b>	LIN break detected interrupt and flag
<b>USART_INT_FLAG_WU</b>	wakeup from deep-sleep mode interrupt and flag
<b>USART_INT_FLAG_CTS</b>	CTS interrupt and flag
<b>USART_INT_FLAG_ER_NERR</b>	error interrupt and noise error flag

<b>USART_INT_FLAG_ER</b>	error interrupt and overrun error
<b>USART_INT_FLAG_ER</b>	error interrupt and frame error flag
<b>USART_INT_FLAG_RF</b>	receive FIFO full interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the USART0 interrupt flag status */

FlagStatus status;

status = usart_interrupt_flag_get(USART0, USART_INT_FLAG_RBNE);

usart_interrupt_flag_clear
```

The description of `usart_interrupt_flag_clear` is shown as below:

**Table 3-677. Function `usart_interrupt_flag_clear`**

<b>Function name</b>	<code>usart_interrupt_flag_clear</code>
<b>Function prototype</b>	<code>void usart_interrupt_flag_clear(uint32_t usart_periph,                                 usart_interrupt_flag_enum int_flag);</code>
<b>Function descriptions</b>	clear USART interrupt flag in STAT register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag
<b>USART_INT_FLAG_PE</b>	parity error flag
<b>USART_INT_FLAG_ER</b>	frame error flag
<b>USART_INT_FLAG_ER</b>	noise detected flag
<b>USART_INT_FLAG_RB</b>	read data buffer not empty interrupt and overrun error flag
<b>USART_INT_FLAG_ER</b>	error interrupt and overrun error
<b>USART_INT_FLAG_ID</b>	idle line detected flag

<i>LE</i>	
<i>USART_INT_FLAG_TC</i>	transmission complete flag
<i>USART_INT_FLAG_LB_D</i>	LIN break detected flag
<i>USART_INT_FLAG_CTS_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_AM_M</i>	address match flag
<i>USART_INT_FLAG_WU_U</i>	wakeup from deep-sleep mode flag
<i>USART_INT_FLAG_RF_F</i>	receive FIFO full interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the USART0 interrupt flag */
uart_interrupt_flag_clear(USART0, USART_INT_FLAG_TC);
```

## 3.24. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.24.1](#), the FWDGT firmware functions are introduced in chapter [3.24.2](#).

### 3.24.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

**Table 3-678. WWDGT Registers**

Registers	Descriptions
WWDGT_CTL	Control register
WWDGT_CFG	Configuration register
WWDGT_STAT	Status register

### 3.24.2. Descriptions of Peripheral functions

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

**Table 3-679. WWDGT firmware function**

Function name	Function description
wwdgt_deinit	reset the WWDGT counter configuration
wwdgt_enable	start the WWDGT counter
wwdgt_counter_update	configure the WWDGT counter value
wwdgt_config	configure counter value, window value, and prescaler divider value
wwdgt_interrupt_enable	enable early wakeup interrupt of WWDGT
wwdgt_flag_get	check early wakeup interrupt state of WWDGT
wwdgt_flag_clear	clear early wakeup interrupt state of WWDGT

#### **wwdgt\_deinit**

The description of wwdgt\_deinit is shown as below:

**Table 3-680. Function wwdgt\_deinit**

<b>Function name</b>	wwdgt_deinit
<b>Function prototype</b>	void wwdgt_deinit(void);
<b>Function descriptions</b>	reset the WWDGT counter configuration
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset the WWDGT configuration */
wwdgt_deinit();
```

#### **wwdgt\_enable**

The description of wwdgt\_enable is shown as below:

**Table 3-681. Function wwdgt\_enable**

<b>Function name</b>	wwdgt_enable
<b>Function prototype</b>	void wwdgt_enable (void);
<b>Function descriptions</b>	start the WWDGT counter
<b>Precondition</b>	-

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 3-682. Function wwdgt\_counter\_update**

Function name	wwdgt_counter_update
Function prototype	void wwdgt_counter_update(uint16_t counter_value);
Function descriptions	configure the WWDGT counter value
Precondition	-
Input parameter{in}	
counter_value	counter_value: 0x00000000 - 0x0000007F
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* update WWDGT counter to 0x7F */

wwdgt_counter_update(127);
```

### wwdgt\_config

The description of wwdgt\_config is shown as below:

**Table 3-683. Function wwdgt\_config**

Function name	wwdgt_config
Function prototype	void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler);
Function descriptions	configure counter value, window value, and prescaler divider value
Precondition	-
Input parameter{in}	
counter	counter: 0x00000000 - 0x0000007F
Input parameter{in}	
window	window: 0x00000000 - 0x0000007F

Input parameter{in}	
<b>prescaler</b>	wwdgt prescaler value
<i>WWDGT_CFG_PSC_DIV1</i>	the time base of WWDGT counter = (PCLK1/4096)/1
<i>WWDGT_CFG_PSC_DIV2</i>	the time base of WWDGT counter = (PCLK1/4096)/2
<i>WWDGT_CFG_PSC_DIV4</i>	the time base of WWDGT counter = (PCLK1/4096)/4
<i>WWDGT_CFG_PSC_DIV8</i>	the time base of WWDGT counter = (PCLK1/4096)/8
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* confire WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */

```

```
wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

### **wwdgt\_interrupt\_enable**

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

**Table 3-684. Function wwdgt\_interrupt\_enable**

<b>Function name</b>	wwdgt_interrupt_enable
<b>Function prototype</b>	void wwdgt_interrupt_enable(void);
<b>Function descriptions</b>	enable early wakeup interrupt of WWDGT
<b>Precondition</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable early wakeup interrupt of WWDGT */
```

```
wwdgt_interrupt_enable();
```

### **wwdgt\_flag\_get**

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

**Table 3-685. Function wwdgt\_flag\_get**

<b>Function name</b>	wwdgt_flag_get
<b>Function prototype</b>	FlagStatus wwdgt_flag_get(void);
<b>Function descriptions</b>	check early wakeup interrupt state of WWDGT
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* test if the counter value update has reached the 0x40 */

FlagStatus status;

status = wwdgt_flag_get ( );
```

### **wwdgt\_flag\_clear**

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

**Table 3-686. Function wwdgt\_flag\_clear**

<b>Function name</b>	wwdgt_flag_clear
<b>Function prototype</b>	void wwdgt_flag_clear(void);
<b>Function descriptions</b>	clear early wakeup interrupt state of WWDGT
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear early wakeup interrupt state of WWDGT */

wwdgt_flag_clear( );
```

## **3.25. USBFS**

Firmware function description of USBFS refers to document **GD32F3x0-Firmware-Library-USB User Manual\_V1.0**.

## 4. Revision history

**Table 4-1. Revision history**

Revision No.	Description	Date
1.0	Initial Release	Jun.1, 2019

## Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company under the intellectual property laws and treaties of the People's Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as its suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.