

**GigaDevice Semiconductor Inc.**

**GD32E10x**  
**ARM® Cortex™ -M4 32-bit MCU**

**Firmware Library**  
**User Guide**

Revision 1.5

(Dec. 2023)

# 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>20</b>  |
| <b>1.1. Rules of User Manual and Firmware Library .....</b>           | <b>20</b>  |
| 1.1.1. Peripherals.....   | 20         |
| 1.1.2. Naming rules.....  | 21         |
| <b>2. Firmware Library Overview .....</b>                             | <b>22</b>  |
| <b>2.1. File Structure of Firmware Library .....</b>                  | <b>22</b>  |
| 2.1.1. Examples Folder .....  | 23         |
| 2.1.2. Firmware Folder.....   | 23         |
| 2.1.3. Template Folder .....  | 23         |
| 2.1.4. Utilities Folder .....   | 26         |
| <b>2.2. File descriptions of Firmware Library .....</b>               | <b>27</b>  |
| <b>3. Firmware Library of Standard Peripherals .....</b>              | <b>28</b>  |
| <b>3.1. Overview of Firmware Library of Standard Peripherals.....</b> | <b>28</b>  |
| <b>3.2. ADC .....</b>   | <b>28</b>  |
| 3.2.1. Descriptions of Peripheral registers.....                      | 28         |
| 3.2.2. Descriptions of Peripheral functions .....                     | 29         |
| <b>3.3. BKP.....</b>  | <b>57</b>  |
| 3.3.1. Descriptions of Peripheral registers.....                      | 57         |
| 3.3.2. Descriptions of Peripheral functions .....                     | 57         |
| <b>3.4. CRC .....</b>   | <b>68</b>  |
| 3.4.1. Descriptions of Peripheral registers.....                      | 68         |
| 3.4.2. Descriptions of Peripheral functions .....                     | 68         |
| <b>3.5. CTC.....</b>  | <b>73</b>  |
| 3.5.1. Descriptions of Peripheral registers.....                      | 73         |
| 3.5.2. Descriptions of Peripheral functions .....                     | 73         |
| <b>3.6. DAC .....</b>   | <b>86</b>  |
| 3.6.1. Descriptions of Peripheral registers.....                      | 86         |
| 3.6.2. Descriptions of Peripheral functions .....                     | 86         |
| <b>3.7. DBG .....</b>   | <b>100</b> |
| 3.7.1. Descriptions of Peripheral registers.....                      | 100        |
| 3.7.2. Descriptions of Peripheral functions .....                     | 101        |

|  |            |
|--|------------|
| <b>3.8. DMA</b>                              | <b>105</b> |
| 3.8.1. Descriptions of Peripheral registers  | 106        |
| 3.8.2. Descriptions of Peripheral functions  | 106        |
| <b>3.9. EXMC</b>                             | <b>126</b> |
| 3.9.1. Descriptions of Peripheral registers  | 126        |
| 3.9.2. Descriptions of Peripheral functions  | 126        |
| <b>3.10. EXTI</b>                            | <b>131</b> |
| 3.10.1. Descriptions of Peripheral registers | 131        |
| 3.10.2. Descriptions of Peripheral functions | 132        |
| <b>3.11. FMC</b>                             | <b>139</b> |
| 3.11.1. Descriptions of Peripheral registers | 139        |
| 3.11.2. Descriptions of Peripheral functions | 139        |
| <b>3.12. FWDGT</b>                           | <b>159</b> |
| 3.12.1. Descriptions of Peripheral registers | 159        |
| 3.12.2. Descriptions of Peripheral functions | 160        |
| <b>3.13. GPIO</b>                            | <b>165</b> |
| 3.13.1. Descriptions of Peripheral registers | 165        |
| 3.13.2. Descriptions of Peripheral functions | 165        |
| <b>3.14. I2C</b>                             | <b>178</b> |
| 3.14.1. Descriptions of Peripheral registers | 178        |
| 3.14.2. Descriptions of Peripheral functions | 179        |
| <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>211</b> |
| 3.16.1. Descriptions of Peripheral registers | 211        |
| 3.16.2. Descriptions of Peripheral functions | 211        |
| <b>3.17. RCU</b>                             | <b>219</b> |
| 3.17.1. Descriptions of Peripheral registers | 219        |
| 3.17.2. Descriptions of Peripheral functions | 220        |
| <b>3.18. RTC</b>                             | <b>250</b> |
| 3.18.1. Descriptions of Peripheral registers | 251        |
| 3.18.2. Descriptions of Peripheral functions | 251        |
| <b>3.19. SPI</b>                             | <b>260</b> |
| 3.19.1. Descriptions of Peripheral registers | 260        |
| 3.19.2. Descriptions of Peripheral functions | 261        |
| <b>3.20. TIMER</b>                           | <b>287</b> |
| 3.20.1. Descriptions of Peripheral registers | 287        |
| 3.20.2. Descriptions of Peripheral functions | 287        |

---

|              |   |            |
|--------------|---|------------|
| <b>3.21.</b> | <b>USART.....</b>                         | <b>343</b> |
| 3.21.1.      | Descriptions of Peripheral registers..... | 343        |
| 3.21.2.      | Descriptions of Peripheral functions..... | 343        |
| <b>3.22.</b> | <b>WWDGT.....</b>                         | <b>378</b> |
| 3.22.1.      | Descriptions of Peripheral registers..... | 378        |
| 3.22.2.      | Descriptions of Peripheral functions..... | 378        |
| <b>4.</b>    | <b>Revision history.....</b>              | <b>384</b> |

## List of Figures

|   |    |
|---|----|
| Figure 2-1. File structure of firmware library of GD32E10x..... | 22 |
| Figure 2-2. Select peripheral example files.....                | 24 |
| Figure 2-3. Copy the peripheral example files .....             | 25 |
| Figure 2-4. Open the project file.....                          | 25 |
| Figure 2-5. Configure project files .....                       | 26 |
| Figure 2-6. Compile-debug-download .....                        | 26 |

## List of Tables

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

|  |    |
|--|----|
| Table 3-39. Function <code>adc_interrupt_flag_clear</code> .....                   | 55 |
| Table 3-40. Function <code>adc_interrupt_enable</code> .....                       | 55 |
| Table 3-41. Function <code>adc_interrupt_disable</code> .....                      | 56 |
| Table 3-42. BKP Registers .....  | 57 |
| Table 3-43. BKP firmware function .....  | 57 |
| Table 3-44. Function <code>bkp_deinit</code> .....                                 | 58 |
| Table 3-45. Function <code>bkp_data_write</code> .....                             | 58 |
| Table 3-46. Function <code>bkp_data_read</code> .....                              | 59 |
| Table 3-47. Function <code>bkp_rtc_calibration_output_enable</code> .....          | 60 |
| Table 3-48. Function <code>bkp_rtc_calibration_output_disable</code> .....         | 60 |
| Table 3-49. Function <code>bkp_rtc_signal_output_enable</code> .....               | 60 |
| Table 3-50. Function <code>bkp_rtc_signal_output_disable</code> .....              | 61 |
| Table 3-51. Function <code>bkp_rtc_output_select</code> .....                      | 61 |
| Table 3-52. Function <code>bkp_rtc_clock_output_select</code> .....                | 62 |
| Table 3-53. Function <code>bkp_rtc_clock_calibration_direction_select</code> ..... | 63 |
| Table 3-54. Function <code>bkp_rtc_calibration_value_set</code> .....              | 63 |
| Table 3-55. Function <code>bkp_tamper_detection_enable</code> .....                | 64 |
| Table 3-56. Function <code>bkp_tamper_detection_disable</code> .....               | 64 |
| Table 3-57. Function <code>bkp_tamper_active_level_set</code> .....                | 65 |
| Table 3-58. Function <code>bkp_interrupt_enable</code> .....                       | 65 |
| Table 3-59. Function <code>bkp_interrupt_disable</code> .....                      | 66 |
| Table 3-60. Function <code>bkp_flag_get</code> .....                               | 66 |
| Table 3-61. Function <code>bkp_flag_clear</code> .....                             | 67 |
| Table 3-62. Function <code>bkp_interrupt_flag_get</code> .....                     | 67 |
| Table 3-63. Function <code>bkp_interrupt_flag_clear</code> .....                   | 68 |
| Table 3-64. CRC Registers .....  | 68 |
| Table 3-65. CRC firmware function .....  | 69 |
| Table 3-66. Function <code>crc_deinit</code> .....                                 | 69 |
| Table 3-67. Function <code>crc_data_register_reset</code> .....                    | 69 |
| Table 3-68. Function <code>crc_data_register_read</code> .....                     | 70 |
| Table 3-69. Function <code>crc_free_data_register_read</code> .....                | 70 |
| Table 3-70. Function <code>crc_free_data_register_write</code> .....               | 71 |
| Table 3-71. Function <code>crc_single_data_calculate</code> .....                  | 71 |
| Table 3-72. Function <code>crc_block_data_calculate</code> .....                   | 72 |
| Table 3-73. CTC Registers .....  | 73 |
| Table 3-74. CTC firmware function .....  | 73 |
| Table 3-75. Function <code>ctc_deinit</code> .....                                 | 74 |
| Table 3-76. Function <code>ctc_counter_enable</code> .....                         | 74 |
| Table 3-77. Function <code>ctc_counter_disable</code> .....                        | 75 |
| Table 3-78. Function <code>ctc_irc48m_trim_value_config</code> .....               | 75 |
| Table 3-79. Function <code>ctc_software_refsource_pulse_generate</code> .....      | 76 |
| Table 3-80. Function <code>ctc_hardware_trim_mode_config</code> .....              | 76 |
| Table 3-81. Function <code>ctc_refsource_polarity_config</code> .....              | 77 |
| Table 3-82. Function <code>ctc_refsource_signal_select</code> .....                | 77 |

|  |     |
|--|-----|
| Table 3-83. Function ctc_refsource_prescaler_config .....          | 78  |
| Table 3-84. Function ctc_clock_limit_value_config .....            | 79  |
| Table 3-85. Function ctc_counter_reload_value_config .....         | 79  |
| Table 3-86. Function ctc_counter_capture_value_read .....          | 80  |
| Table 3-87. Function ctc_counter_direction_read .....              | 80  |
| Table 3-88. Function ctc_counter_reload_value_read .....           | 81  |
| Table 3-89. Function ctc_irc48m_trim_value_read .....              | 81  |
| Table 3-90. Function ctc_interrupt_enable .....                    | 82  |
| Table 3-91. Function ctc_interrupt_disable .....                   | 82  |
| Table 3-92. Function ctc_interrupt_flag_get .....                  | 83  |
| Table 3-93. Function ctc_interrupt_flag_clear .....                | 84  |
| Table 3-94. Function ctc_flag_get .....                            | 84  |
| Table 3-95. Function ctc_flag_clear .....                          | 85  |
| Table 3-96. DAC Registers .....                                    | 86  |
| Table 3-97. DAC firmware function .....                            | 86  |
| Table 3-98. Function dac_deinit .....                              | 87  |
| Table 3-99. Function dac_enable .....                              | 87  |
| Table 3-100. Function dac_disable .....                            | 88  |
| Table 3-101. Function dac_dma_enable .....                         | 89  |
| Table 3-102. Function dac_dma_disable .....                        | 89  |
| Table 3-103. Function dac_output_buffer_enable .....               | 90  |
| Table 3-104. Function dac_output_buffer_disable .....              | 90  |
| Table 3-105. Function dac_output_value_get .....                   | 91  |
| Table 3-106. Function dac_data_set .....                           | 92  |
| Table 3-107. Function dac_trigger_enable .....                     | 92  |
| Table 3-108. Function dac_trigger_disable .....                    | 93  |
| Table 3-109. Function dac_trigger_source_config .....              | 93  |
| Table 3-110. Function dac_software_trigger_enable .....            | 94  |
| Table 3-111. Function dac_wave_mode_config .....                   | 95  |
| Table 3-112. Function dac_ifsr_noise_config .....                  | 96  |
| Table 3-113. Function dac_triangle_noise_config .....              | 96  |
| Table 3-114. Function dac_concurrent_enable .....                  | 97  |
| Table 3-115. Function dac_concurrent_disable .....                 | 98  |
| Table 3-116. Function dac_concurrent_software_trigger_enable ..... | 98  |
| Table 3-117. Function dac_concurrent_output_buffer_enable .....    | 99  |
| Table 3-118. Function dac_concurrent_output_buffer_disable .....   | 99  |
| Table 3-119. Function dac_concurrent_data_set .....                | 100 |
| Table 3-120. DBG Registers .....                                   | 101 |
| Table 3-121. DBG firmware function .....                           | 101 |
| Table 3-122. Enum dbg_periph_enum .....                            | 101 |
| Table 3-123. Function dbg_id_get .....                             | 102 |
| Table 3-124. Function dbg_low_power_enable .....                   | 102 |
| Table 3-125. Function dbg_low_power_disable .....                  | 103 |
| Table 3-126. Function dbg_periph_enable .....                      | 103 |



|   |     |
|---|-----|
| Table 3-127. Function dbg_periph_disable .....                    | 104 |
| Table 3-128. Function dbg_trace_pin_enable .....                  | 105 |
| Table 3-129. Function dbg_trace_pin_disable .....                 | 105 |
| Table 3-130. DMA Registers .....                                  | 106 |
| Table 3-131. DMA firmware function .....                          | 106 |
| Table 3-132. Structure dma_parameter_struct .....                 | 107 |
| Table 3-133. Function dma_deinit .....                            | 107 |
| Table 3-134. Function dma_struct_para_init .....                  | 108 |
| Table 3-135. Function dma_init .....                              | 108 |
| Table 3-136. Function dma_circulation_enable .....                | 109 |
| Table 3-137. Function dma_circulation_disable .....               | 110 |
| Table 3-138. Function dma_memory_to_memory_enable .....           | 110 |
| Table 3-139. Function dma_memory_to_memory_disable .....          | 111 |
| Table 3-140. Function dma_channel_enable .....                    | 112 |
| Table 3-141. Function dma_channel_disable .....                   | 112 |
| Table 3-142. Function dma_periph_address_config .....             | 113 |
| Table 3-143. Function dma_memory_address_config .....             | 114 |
| Table 3-144. Function dma_transfer_number_config .....            | 114 |
| Table 3-145. Function dma_transfer_number_get .....               | 115 |
| Table 3-146. Function dma_priority_config .....                   | 116 |
| Table 3-147. Function dma_memory_width_config .....               | 116 |
| Table 3-148. Function dma_periph_width_config .....               | 117 |
| Table 3-149. Function dma_memory_increase_enable .....            | 118 |
| Table 3-150. Function dma_memory_increase_disable .....           | 119 |
| Table 3-151. Function dma_periph_increase_enable .....            | 119 |
| Table 3-152. Function dma_periph_increase_disable .....           | 120 |
| Table 3-153. Function dma_transfer_direction_config .....         | 120 |
| Table 3-154. Function dma_flag_get .....                          | 121 |
| Table 3-155. Function dma_flag_clear .....                        | 122 |
| Table 3-156. Function dma_interrupt_flag_get .....                | 122 |
| Table 3-157. Function dma_interrupt_flag_clear .....              | 123 |
| Table 3-158. Function dma_interrupt_enable .....                  | 124 |
| Table 3-159. Function dma_interrupt_disable .....                 | 125 |
| Table 3-160. EXMC Registers .....                                 | 126 |
| Table 3-161. EXMC firmware function .....                         | 126 |
| Table 3-162. Structure exmc_norsram_timing_parameter_struct ..... | 126 |
| Table 3-163. Structure exmc_norsram_parameter_struct .....        | 127 |
| Table 3-164. Function exmc_norsram_deinit .....                   | 127 |
| Table 3-165. Function exmc_norsram_init .....                     | 128 |
| Table 3-166. Function exmc_norsram_struct_para_init .....         | 129 |
| Table 3-167. Function exmc_norsram_enable .....                   | 129 |
| Table 3-168. Function exmc_norsram_disable .....                  | 130 |
| Table 3-169. Function exmc_norsram_page_size_config .....         | 130 |
| Table 3-170. EXTI Registers .....                                 | 131 |

|   |     |
|---|-----|
| Table 3-171. EXTI firmware function .....                   | 132 |
| Table 3-172. Enum exti_line_enum .....                      | 132 |
| Table 3-173. Enum exti_mode_enum .....                      | 133 |
| Table 3-174. Enum exti_trig_type_enum .....                 | 133 |
| Table 3-175. Function exti_deinit .....                     | 133 |
| Table 3-176. Function exti_init .....                       | 133 |
| Table 3-177. Function exti_interrupt_enable .....           | 134 |
| Table 3-178. Function exti_interrupt_disable .....          | 135 |
| Table 3-179. Function exti_event_enable .....               | 135 |
| Table 3-180. Function exti_event_disable .....              | 135 |
| Table 3-181. Function exti_software_interrupt_enable .....  | 136 |
| Table 3-182. Function exti_software_interrupt_disable ..... | 136 |
| Table 3-183. Function exti_flag_get .....                   | 137 |
| Table 3-184. Function exti_flag_clear .....                 | 137 |
| Table 3-185. Function exti_interrupt_flag_get .....         | 138 |
| Table 3-186. Function exti_interrupt_flag_clear .....       | 138 |
| Table 3-187. FMC Registers .....                            | 139 |
| Table 3-188. FMC firmware function .....                    | 139 |
| Table 3-189. Enum fmc_state_enum .....                      | 140 |
| Table 3-190. Enum fmc_int_enum .....                        | 141 |
| Table 3-191. Enum fmc_flag_enum .....                       | 141 |
| Table 3-192. Enum fmc_interrupt_flag_enum .....             | 141 |
| Table 3-193. Function fmc_wscnt_set .....                   | 141 |
| Table 3-194. Function fmc_prefetch_enable .....             | 142 |
| Table 3-195. Function fmc_prefetch_disable .....            | 142 |
| Table 3-196. Function fmc_ibus_enable .....                 | 143 |
| Table 3-197. Function fmc_ibus_disable .....                | 143 |
| Table 3-198. Function fmc_dbus_enable .....                 | 144 |
| Table 3-199. Function fmc_dbus_disable .....                | 144 |
| Table 3-200. Function fmc_ibus_reset .....                  | 145 |
| Table 3-201. Function fmc_dbus_reset .....                  | 145 |
| Table 3-202. Function fmc_program_width_set .....           | 146 |
| Table 3-203. Function fmc_unlock .....                      | 146 |
| Table 3-204. Function fmc_lock .....                        | 147 |
| Table 3-205. Function fmc_page_erase .....                  | 147 |
| Table 3-206. Function fmc_mass_erase .....                  | 148 |
| Table 3-207. Function fmc_doubleword_program .....          | 148 |
| Table 3-208. Function fmc_word_program .....                | 149 |
| Table 3-209. Function ob_unlock .....                       | 149 |
| Table 3-210. Function ob_lock .....                         | 150 |
| Table 3-211. Function ob_erase .....                        | 150 |
| Table 3-212. Function ob_write_protection_enable .....      | 151 |
| Table 3-213. Function ob_security_protection_config .....   | 151 |
| Table 3-214. Function ob_user_write .....                   | 152 |

|  |     |
|--|-----|
| Table 3-215. Function ob_data_program.....                     | 152 |
| Table 3-216. Function ob_user_get.....                         | 153 |
| Table 3-217. Function ob_data_program.....                     | 153 |
| Table 3-218. Function ob_write_protection_get .....            | 154 |
| Table 3-219. Function ob_security_protection_flag_get .....    | 154 |
| Table 3-220. Function fmc_interrupt_enable .....               | 155 |
| Table 3-221. Function fmc_interrupt_disable .....              | 155 |
| Table 3-222. Function fmc_flag_get.....                        | 156 |
| Table 3-223. Function fmc_flag_clear.....                      | 156 |
| Table 3-224. Function fmc_interrupt_flag_get.....              | 157 |
| Table 3-225. Function fmc_interrupt_flag_clear.....            | 158 |
| Table 3-226. Function fmc_state_get.....                       | 158 |
| Table 3-227. Function fmc_ready_wait.....                      | 159 |
| Table 3-228. FWDGT Registers.....                              | 159 |
| Table 3-229. FWDGT firmware function .....                     | 160 |
| Table 3-230. Function fwdgt_write_ensable .....                | 160 |
| Table 3-231. Function fwdgt_write_disable .....                | 161 |
| Table 3-232. Function fwdgt_enable .....                       | 161 |
| Table 3-233. Function fwdgt_prescaler_value_config .....       | 161 |
| Table 3-234. Function fwdgt_reload_value_config .....          | 162 |
| Table 3-235. Function fwdgt_counter_reload .....               | 163 |
| Table 3-236. Function fwdgt_config.....                        | 163 |
| Table 3-237. Function fwdgt_flag_get fwdgt_write_disable ..... | 164 |
| Table 3-238. GPIO Registers.....                               | 165 |
| Table 3-239. GPIO firmware function .....                      | 165 |
| Table 3-240. Function gpio_deinit.....                         | 166 |
| Table 3-241. Function gpio_afio_deinit .....                   | 166 |
| Table 3-242. Function gpio_init.....                           | 167 |
| Table 3-243. Function gpio_bit_set.....                        | 168 |
| Table 3-244. Function gpio_bit_reset .....                     | 169 |
| Table 3-245. Function gpio_bit_write.....                      | 169 |
| Table 3-246. Function gpio_port_write .....                    | 170 |
| Table 3-247. Function gpio_input_bit_get.....                  | 170 |
| Table 3-248. Function gpio_input_port_get .....                | 171 |
| Table 3-249. Function gpio_output_bit_get .....                | 172 |
| Table 3-250. Function gpio_output_port_get.....                | 172 |
| Table 3-251. Function gpio_pin_remap_config .....              | 173 |
| Table 3-252. Function gpio_exti_source_select .....            | 174 |
| Table 3-253. Function gpio_event_output_config.....            | 175 |
| Table 3-254. Function gpio_event_output_enable .....           | 176 |
| Table 3-255. Function gpio_event_output_disable .....          | 176 |
| Table 3-256. Function gpio_pin_lock.....                       | 177 |
| Table 3-257. Function gpio_compensation_config .....           | 177 |
| Table 3-258. Function gpio_compensation_flag_get .....         | 178 |

|  |     |
|--|-----|
| Table 3-259. I2C Registers .....                               | 179 |
| Table 3-260. I2C firmware function .....                       | 179 |
| Table 3-261. Enum i2c_flag_enum .....                          | 180 |
| Table 3-262. Enum i2c_interrupt_flag_enum .....                | 181 |
| Table 3-263. Enum i2c_interrupt_enum .....                     | 181 |
| Table 3-264. Function i2c_deinit .....                         | 182 |
| Table 3-265. Function i2c_clock_config .....                   | 182 |
| Table 3-266. Function i2c_mode_addr_config .....               | 183 |
| Table 3-267. Function i2c_smbus_type_config .....              | 184 |
| Table 3-268. Function i2c_ack_config .....                     | 184 |
| Table 3-269. Function i2c_ackpos_config .....                  | 185 |
| Table 3-270. Function i2c_master_addressing .....              | 185 |
| Table 3-271. Function i2c_dualaddr_enable .....                | 186 |
| Table 3-272. Function i2c_dualaddr_disable .....               | 187 |
| Table 3-273. Function i2c_enable .....                         | 187 |
| Table 3-274. Function i2c_disable .....                        | 188 |
| Table 3-275. Function i2c_start_on_bus .....                   | 188 |
| Table 3-276. Function i2c_stop_on_bus .....                    | 189 |
| Table 3-277. Function i2c_data_transmit .....                  | 189 |
| Table 3-278. Function i2c_data_receive .....                   | 190 |
| Table 3-279. Function i2c_dma_config .....                     | 190 |
| Table 3-280. Function i2c_dma_last_transfer_config .....       | 191 |
| Table 3-281. Function i2c_stretch_scl_low_config .....         | 191 |
| Table 3-282. Function i2c_slave_response_to_gcall_config ..... | 192 |
| Table 3-283. Function i2c_software_reset_config .....          | 193 |
| Table 3-284. Function i2c_pec_config .....                     | 193 |
| Table 3-285. Function i2c_pec_transfer_config .....            | 194 |
| Table 3-286. Function i2c_pec_value_get .....                  | 194 |
| Table 3-287. Function i2c_smbus_alert_config .....             | 195 |
| Table 3-288. Function i2c_smbus_arp_config .....               | 196 |
| Table 3-289. Function i2c_sam_enable .....                     | 196 |
| Table 3-290. Function i2c_sam_disable .....                    | 197 |
| Table 3-291. Function i2c_sam_timeout_enable .....             | 197 |
| Table 3-292. Function i2c_sam_timeout_disable .....            | 198 |
| Table 3-293. Function i2c_flag_get .....                       | 198 |
| Table 3-294. Function i2c_flag_clear .....                     | 199 |
| Table 3-295. Function i2c_interrupt_enable .....               | 200 |
| Table 3-296. Function i2c_interrupt_disable .....              | 201 |
| Table 3-297. Function i2c_interrupt_flag_get .....             | 202 |
| Table 3-298. Function i2c_interrupt_flag_clear .....           | 203 |
| Table 3-299. NVIC Registers .....                              | 204 |
| Table 3-300. SysTick Registers .....                           | 205 |
| Table 3-301. IRQn_Type .....                                   | 205 |
| Table 3-302. MISC firmware function .....                      | 207 |

|  |     |
|--|-----|
| Table 3-303. Function nvic_priority_group_set .....        | 207 |
| Table 3-304. Function nvic_irq_enable .....                | 208 |
| Table 3-305. Function nvic_irq_disable .....               | 208 |
| Table 3-306. Function nvic_vector_table_set .....          | 209 |
| Table 3-307. Function system_lowpower_set .....            | 209 |
| Table 3-308. Function system_lowpower_reset .....          | 210 |
| Table 3-309. Function systick_clksource_set .....          | 211 |
| Table 3-310. PMU Registers .....                           | 211 |
| Table 3-311. PMU firmware function .....                   | 212 |
| Table 3-312. Function pmu_deinit .....                     | 212 |
| Table 3-313. Function pmu_lvd_select .....                 | 212 |
| Table 3-314. Function pmu_ldo_output_select .....          | 213 |
| Table 3-315. Function pmu_lvd_disable .....                | 214 |
| Table 3-316. Function pmu_to_sleepmode .....               | 214 |
| Table 3-317. Function pmu_to_deepsleepmode .....           | 215 |
| Table 3-318. Function pmu_to_standbymode .....             | 215 |
| Table 3-319. Function pmu_wakeup_pin_enable .....          | 216 |
| Table 3-320. Function pmu_wakeup_pin_disable .....         | 216 |
| Table 3-321. Function pmu_backup_write_enable .....        | 217 |
| Table 3-322. Function pmu_backup_write_disable .....       | 217 |
| Table 3-323. Function pmu_flag_get .....                   | 218 |
| Table 3-324. Function pmu_flag_clear .....                 | 218 |
| Table 3-325. RCU Registers .....                           | 219 |
| Table 3-326. RCU firmware function .....                   | 220 |
| Table 3-327. rcu_periph_enum .....                         | 221 |
| Table 3-328. rcu_periph_sleep_enum .....                   | 222 |
| Table 3-329. rcu_periph_reset_enum .....                   | 222 |
| Table 3-330. rcu_flag_enum .....                           | 223 |
| Table 3-331. rcu_int_flag_enum .....                       | 224 |
| Table 3-332. rcu_int_flag_clear_enum .....                 | 224 |
| Table 3-333. rcu_int_enum .....                            | 225 |
| Table 3-334. rcu_osci_type_enum .....                      | 225 |
| Table 3-335. rcu_clock_freq_enum .....                     | 225 |
| Table 3-336. Function rcu_deinit .....                     | 226 |
| Table 3-337. Function rcu_periph_clock_enable .....        | 226 |
| Table 3-338. Function rcu_periph_clock_disable .....       | 227 |
| Table 3-339. Function rcu_periph_clock_sleep_enable .....  | 227 |
| Table 3-340. Function rcu_periph_clock_sleep_disable ..... | 228 |
| Table 3-341. Function rcu_periph_reset_enable .....        | 228 |
| Table 3-342. Function rcu_periph_reset_disable .....       | 229 |
| Table 3-343. Function rcu_bkp_reset_enable .....           | 229 |
| Table 3-344. Function rcu_bkp_reset_disable .....          | 230 |
| Table 3-345. Function rcu_system_clock_source_config ..... | 230 |
| Table 3-346. Function rcu_system_clock_source_get .....    | 231 |

|  |     |
|--|-----|
| Table 3-347. Function rcu_ahb_clock_config .....             | 231 |
| Table 3-348. Function rcu_apb1_clock_config .....            | 232 |
| Table 3-349. Function rcu_apb2_clock_config .....            | 232 |
| Table 3-350. Function rcu_ckout0_config .....                | 233 |
| Table 3-351. Function rcu_pll_config .....                   | 234 |
| Table 3-352. Function rcu_pllprel_config.....                | 235 |
| Table 3-353. Function rcu_predv0_config.....                 | 235 |
| Table 3-354. Function rcu_predv1_config.....                 | 236 |
| Table 3-355. Function rcu_pll1_config .....                  | 236 |
| Table 3-356. Function rcu_pll2_config .....                  | 237 |
| Table 3-357. Function rcu_adc_clock_config .....             | 237 |
| Table 3-358. Function rcu_usb_clock_config .....             | 238 |
| Table 3-359. Function rcu_rtc_clock_config.....              | 239 |
| Table 3-360. Function rcu_i2s1_clock_config .....            | 240 |
| Table 3-361. Function rcu_i2s2_clock_config .....            | 240 |
| Table 3-362. Function rcu_ck48m_clock_config .....           | 241 |
| Table 3-363. Function rcu_flag_get.....                      | 242 |
| Table 3-364. Function rcu_all_reset_flag_clear .....         | 242 |
| Table 3-365. Function rcu_interrupt_flag_get.....            | 243 |
| Table 3-366. Function rcu_interrupt_flag_clear.....          | 243 |
| Table 3-367. Function rcu_interrupt_enable .....             | 244 |
| Table 3-368. Function rcu_interrupt_disable .....            | 244 |
| Table 3-369. Function rcu_lxtal_drive_capability_config..... | 245 |
| Table 3-370. Function rcu_osci_stab_wait.....                | 245 |
| Table 3-371. Function rcu_osci_on.....                       | 246 |
| Table 3-372. Function rcu_osci_off.....                      | 246 |
| Table 3-373. Function rcu_osci_bypass_mode_enable.....       | 247 |
| Table 3-374. Function rcu_osci_bypass_mode_disable.....      | 247 |
| Table 3-375. Function rcu_hxtal_clock_monitor_enable.....    | 248 |
| Table 3-376. Function rcu_hxtal_clock_monitor_disable.....   | 248 |
| Table 3-377. Function rcu_irc8m_adjust_value_set.....        | 249 |
| Table 3-378. Function rcu_deepsleep_voltage_set.....         | 249 |
| Table 3-379. Function rcu_clock_freq_get.....                | 250 |
| Table 3-380. RTC Registers .....                             | 251 |
| Table 3-381. RTC firmware function.....                      | 251 |
| Table 3-382. Function rtc_configuration_mode_enter .....     | 252 |
| Table 3-383. Function rtc_configuration_mode_exit .....      | 252 |
| Table 3-384. Function rtc_counter_set.....                   | 252 |
| Table 3-385. Function rtc_prescaler_set .....                | 253 |
| Table 3-386. Function rtc_lwoff_wait .....                   | 254 |
| Table 3-387. Function rtc_register_sync_wait.....            | 254 |
| Table 3-388. Function rtc_alarm_config .....                 | 255 |
| Table 3-389. Function rtc_counter_get.....                   | 255 |
| Table 3-390. Function rtc_divider_get .....                  | 256 |

|  |     |
|--|-----|
| Table 3-391. Function rtc_flag_get.....                      | 256 |
| Table 3-392. Function rtc_flag_clear.....                    | 257 |
| Table 3-393. Function rtc_interrupt_flag_get.....            | 257 |
| Table 3-394. Function rtc_interrupt_flag_clear.....          | 258 |
| Table 3-395. Function rtc_interrupt_enable .....             | 259 |
| Table 3-396. Function rtc_interrupt_disable .....            | 259 |
| Table 3-397. SPI/I2S Registers.....                          | 260 |
| Table 3-398. SPI/I2S firmware function.....                  | 262 |
| Table 3-399. spi_parameter_struct.....                       | 263 |
| Table 3-400. Function spi_i2s_deinit .....                   | 263 |
| Table 3-401. Function spi_struct_para_init.....              | 264 |
| Table 3-402. Function spi_init .....                         | 264 |
| Table 3-403. Function spi_enable .....                       | 265 |
| Table 3-404. Function spi_disable .....                      | 265 |
| Table 3-405. Function i2s_init.....                          | 266 |
| Table 3-406. Function i2s_psc_config .....                   | 267 |
| Table 3-407. Function i2s_enable.....                        | 268 |
| Table 3-408. Function i2s_disable.....                       | 269 |
| Table 3-409. Function spi_nss_output_enable .....            | 269 |
| Table 3-410. Function spi_nss_output_disable .....           | 270 |
| Table 3-411. Function spi_nss_internal_high .....            | 270 |
| Table 3-412. Function spi_nss_internal_low .....             | 271 |
| Table 3-413. Function spi_dma_enable .....                   | 271 |
| Table 3-414. Function spi_dma_disable .....                  | 272 |
| Table 3-415. Function spi_i2s_data_frame_format_config ..... | 273 |
| Table 3-416. Function spi_i2s_data_transmit .....            | 273 |
| Table 3-417. Function spi_i2s_data_receive .....             | 274 |
| Table 3-418. Function spi_bidirectional_transfer_config..... | 274 |
| Table 3-419. Function spi_crc_polynomial_set .....           | 275 |
| Table 3-420. Function spi_crc_polynomial_get .....           | 275 |
| Table 3-421. Function spi_crc_on .....                       | 276 |
| Table 3-422. Function spi_crc_off.....                       | 276 |
| Table 3-423. Function spi_crc_next .....                     | 277 |
| Table 3-424. Function spi_crc_get .....                      | 277 |
| Table 3-425. Function spi_ti_mode_enable.....                | 278 |
| Table 3-426. Function spi_ti_mode_disable.....               | 279 |
| Table 3-427. Function spi_nssp_mode_enable .....             | 279 |
| Table 3-428. Function spi_nssp_mode_disable .....            | 280 |
| Table 3-429. Function spi_quad_enable.....                   | 280 |
| Table 3-430. Function spi_quad_disable.....                  | 281 |
| Table 3-431. Function spi_quad_write_enable .....            | 281 |
| Table 3-432. Function spi_quad_read_enable .....             | 282 |
| Table 3-433. Function spi_quad_io23_output_enable .....      | 282 |
| Table 3-434. Function spi_quad_io23_output_disable .....     | 283 |



|  |     |
|--|-----|
| Table 3-435. Function spi_i2s_interrupt_enable .....               | 283 |
| Table 3-436. Function spi_i2s_interrupt_disable .....              | 284 |
| Table 3-437. Function spi_i2s_interrupt_flag_get .....             | 284 |
| Table 3-438. Function spi_i2s_flag_get .....                       | 285 |
| Table 3-439. Function spi_crc_error_clear.....                     | 286 |
| Table 3-440. TIMERx Registers.....                                 | 287 |
| Table 3-441. TIMERx firmware function.....                         | 288 |
| Table 3-442. Structure timer_parameter_struct.....                 | 290 |
| Table 3-443. Structure timer_break_parameter_struct .....          | 290 |
| Table 3-444. Structure timer_oc_parameter_struct.....              | 291 |
| Table 3-445. Structure timer_ic_parameter_struct.....              | 291 |
| Table 3-446. Function timer_deinit.....                            | 291 |
| Table 3-447. Function timer_struct_para_init .....                 | 292 |
| Table 3-448. Function timer_init.....                              | 292 |
| Table 3-449. Function timer_enable.....                            | 293 |
| Table 3-450. Function timer_disable .....                          | 294 |
| Table 3-451. Function timer_auto_reload_shadow_enable .....        | 294 |
| Table 3-452. Function timer_auto_reload_shadow_disable .....       | 295 |
| Table 3-453. Function timer_update_event_enable .....              | 295 |
| Table 3-454. Function timer_update_event_disable .....             | 296 |
| Table 3-455. Function timer_counter_alignment .....                | 296 |
| Table 3-456. Function timer_counter_up_direction .....             | 297 |
| Table 3-457. timer_counter_down_direction .....                    | 297 |
| Table 3-458. Function timer_prescaler_config .....                 | 298 |
| Table 3-459. Function timer_repetition_value_config.....           | 299 |
| Table 3-460. Function timer_autoreload_value_config.....           | 299 |
| Table 3-461. Function timer_counter_value_config.....              | 300 |
| Table 3-462. Function timer_counter_read .....                     | 300 |
| Table 3-463. Function timer_prescaler_read .....                   | 301 |
| Table 3-464. Function timer_single_pulse_mode_config.....          | 301 |
| Table 3-465. Function timer_update_source_config.....              | 302 |
| Table 3-466. Function timer_dma_enable .....                       | 303 |
| Table 3-467. Function timer_dma_disable .....                      | 304 |
| Table 3-468. Function timer_channel_dma_request_source_select..... | 304 |
| Table 3-469. Function timer_dma_transfer_config .....              | 305 |
| Table 3-470. Function timer_event_software_generate .....          | 307 |
| Table 3-471. Function timer_break_struct_para_init .....           | 308 |
| Table 3-472. Function timer_break_config.....                      | 308 |
| Table 3-473. Function timer_break_enable .....                     | 309 |
| Table 3-474. Function timer_break_disable .....                    | 310 |
| Table 3-475. Function timer_automatic_output_enable .....          | 310 |
| Table 3-476. Function timer_automatic_output_disable .....         | 311 |
| Table 3-477. Function timer_primary_output_config.....             | 311 |
| Table 3-478. Function timer_channel_control_shadow_config.....     | 312 |



|  |     |
|--|-----|
| Table 3-479. Function timer_channel_control_shadow_update_config .....         | 312 |
| Table 3-480. Function timer_channel_output_struct_para_init .....              | 313 |
| Table 3-481. Function timer_channel_output_config .....                        | 314 |
| Table 3-482. Function timer_channel_output_mode_config .....                   | 315 |
| Table 3-483. Function timer_channel_output_pulse_value_config .....            | 316 |
| Table 3-484. Function timer_channel_output_shadow_config .....                 | 316 |
| Table 3-485. Function timer_channel_output_fast_config .....                   | 317 |
| Table 3-486. Function timer_channel_output_clear_config .....                  | 318 |
| Table 3-487. Function timer_channel_output_polarity_config .....               | 319 |
| Table 3-488. Function timer_channel_complementary_output_polarity_config ..... | 320 |
| Table 3-489. Function timer_channel_output_state_config .....                  | 321 |
| Table 3-490. Function timer_channel_complementary_output_state_config .....    | 321 |
| Table 3-491. Function timer_channel_input_struct_para_init .....               | 322 |
| Table 3-492. Function timer_input_capture_config .....                         | 323 |
| Table 3-493. Function timer_channel_input_capture_prescaler_config .....       | 324 |
| Table 3-494. Function timer_channel_capture_value_register_read .....          | 324 |
| Table 3-495. Function timer_input_pwm_capture_config .....                     | 325 |
| Table 3-496. Function timer_hall_mode_config .....                             | 326 |
| Table 3-497. Function timer_input_trigger_source_select .....                  | 327 |
| Table 3-498. Function timer_master_output_trigger_source_select .....          | 328 |
| Table 3-499. Function timer_slave_mode_select .....                            | 329 |
| Table 3-500. Function timer_master_slave_mode_config .....                     | 330 |
| Table 3-501. Function timer_external_trigger_config .....                      | 330 |
| Table 3-502. Function timer_quadrature_decoder_mode_config .....               | 331 |
| Table 3-503. Function timer_internal_clock_config .....                        | 332 |
| Table 3-504. Function timer_internal_trigger_as_external_clock_config .....    | 333 |
| Table 3-505. Function timer_external_trigger_as_external_clock_config .....    | 333 |
| Table 3-506. Function timer_external_clock_mode0_config .....                  | 334 |
| Table 3-507. Function timer_external_clock_mode1_config .....                  | 335 |
| Table 3-508. Function timer_external_clock_mode1_disable .....                 | 336 |
| Table 3-509. Function timer_write_chxval_register_config .....                 | 337 |
| Table 3-510. Function timer_output_value_selection_config .....                | 337 |
| Table 3-511. Function timer_interrupt_enable .....                             | 338 |
| Table 3-512. Function timer_interrupt_disable .....                            | 339 |
| Table 3-513. Function timer_interrupt_flag_get .....                           | 340 |
| Table 3-514. Function timer_interrupt_flag_clear .....                         | 340 |
| Table 3-515. Function timer_flag_get .....                                     | 341 |
| Table 3-516. Function timer_flag_clear .....                                   | 342 |
| Table 3-517. USART Registers .....   | 343 |
| Table 3-518. USART firmware function .....                                     | 344 |
| Table 3-519. usart_flag_enum .....   | 345 |
| Table 3-520. usart_interrupt_flag_enum .....                                   | 345 |
| Table 3-521. usart_interrupt_enum .....  | 346 |
| Table 3-522. usart_invert_enum .....   | 346 |

|  |     |
|--|-----|
| Table 3-523. Function <code>usart_deinit</code> .....                            | 347 |
| Table 3-524. Function <code>usart_baudrate_set</code> .....                      | 347 |
| Table 3-525. Function <code>usart_parity_config</code> .....                     | 348 |
| Table 3-526. Function <code>usart_word_length_set</code> .....                   | 348 |
| Table 3-527. Function <code>usart_stop_bit_set</code> .....                      | 349 |
| Table 3-528. Function <code>usart_enable</code> .....                            | 350 |
| Table 3-529. Function <code>usart_disable</code> .....                           | 350 |
| Table 3-530. Function <code>usart_transmit_config</code> .....                   | 351 |
| Table 3-531. Function <code>usart_receive_config</code> .....                    | 351 |
| Table 3-532. Function <code>usart_data_first_config</code> .....                 | 352 |
| Table 3-533. Function <code>usart_invert_config</code> .....                     | 353 |
| Table 3-534. Function <code>usart_receiver_timeout_enable</code> .....           | 353 |
| Table 3-535. Function <code>usart_receiver_timeout_disable</code> .....          | 354 |
| Table 3-536. Function <code>usart_receiver_timeout_threshold_config</code> ..... | 354 |
| Table 3-537. Function <code>usart_data_transmit</code> .....                     | 355 |
| Table 3-538. Function <code>usart_data_receive</code> .....                      | 356 |
| Table 3-539. Function <code>usart_address_config</code> .....                    | 356 |
| Table 3-540. Function <code>usart_mute_mode_enable</code> .....                  | 357 |
| Table 3-541. Function <code>usart_mute_mode_disable</code> .....                 | 357 |
| Table 3-542. Function <code>usart_mute_mode_wakeup_config</code> .....           | 358 |
| Table 3-543. Function <code>usart_lin_mode_enable</code> .....                   | 358 |
| Table 3-544. Function <code>usart_lin_mode_disable</code> .....                  | 359 |
| Table 3-545. Function <code>usart_lin_break_dection_length_config</code> .....   | 360 |
| Table 3-546. Function <code>usart_send_break</code> .....                        | 360 |
| Table 3-547. Function <code>usart_halfduplex_enable</code> .....                 | 361 |
| Table 3-548. Function <code>usart_halfduplex_disable</code> .....                | 361 |
| Table 3-549. Function <code>usart_synchronous_clock_enable</code> .....          | 362 |
| Table 3-550. Function <code>usart_synchronous_clock_disable</code> .....         | 362 |
| Table 3-551. Function <code>usart_synchronous_clock_config</code> .....          | 363 |
| Table 3-552. Function <code>usart_guard_time_config</code> .....                 | 364 |
| Table 3-553. Function <code>usart_smartcard_mode_enable</code> .....             | 364 |
| Table 3-554. Function <code>usart_smartcard_mode_disable</code> .....            | 365 |
| Table 3-555. Function <code>usart_smartcard_mode_nack_enable</code> .....        | 365 |
| Table 3-556. Function <code>usart_smartcard_mode_nack_disable</code> .....       | 366 |
| Table 3-557. Function <code>usart_smartcard_autoretry_config</code> .....        | 366 |
| Table 3-558. Function <code>usart_block_length_config</code> .....               | 367 |
| Table 3-559. Function <code>usart_irda_mode_enable</code> .....                  | 367 |
| Table 3-560. Function <code>usart_irda_mode_disable</code> .....                 | 368 |
| Table 3-561. Function <code>usart_prescaler_config</code> .....                  | 368 |
| Table 3-562. Function <code>usart_irda_lowpower_config</code> .....              | 369 |
| Table 3-563. Function <code>usart_hardware_flow_rts_config</code> .....          | 370 |
| Table 3-564. Function <code>usart_hardware_flow_cts_config</code> .....          | 370 |
| Table 3-565. Function <code>usart_dma_receive_config</code> .....                | 371 |
| Table 3-566. Function <code>usart_dma_transmit_config</code> .....               | 372 |

|   |     |
|---|-----|
| Table 3-567. Function <code>usart_hardware_flow_coherence_config</code> ..... | 372 |
| Table 3-568. Function <code>usart_flag_get</code> .....                       | 373 |
| Table 3-569. Function <code>usart_flag_clear</code> .....                     | 374 |
| Table 3-570. Function <code>usart_interrupt_enable</code> .....               | 374 |
| Table 3-571. Function <code>usart_interrupt_disable</code> .....              | 375 |
| Table 3-572. Function <code>usart_interrupt_flag_get</code> .....             | 376 |
| Table 3-573. Function <code>usart_interrupt_flag_clear</code> .....           | 377 |
| Table 3-574. WWDGT Registers .....  | 378 |
| Table 3-575. WWDGT firmware function.....                                     | 378 |
| Table 3-576. Function <code>wwdgt_deinit</code> .....                         | 379 |
| Table 3-577. Function <code>wwdgt_enable</code> .....                         | 379 |
| Table 3-578. Function <code>wwdgt_counter_update</code> .....                 | 380 |
| Table 3-579. Function <code>wwdgt_config</code> .....                         | 380 |
| Table 3-580. Function <code>wwdgt_interrupt_enable</code> .....               | 381 |
| Table 3-581. Function <code>wwdgt_flag_get</code> .....                       | 381 |
| Table 3-582. Function <code>wwdgt_flag_clear</code> .....                     | 382 |
| Table 4-1. Revision history .....   | 384 |

## 1. Introduction

This manual introduces firmware library of GD32E10x 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 GD32E10x 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 |
| BKP         | Backup registers            |
| CRC         | CRC calculation unit        |
| CTC         | Clock trim controller       |
| DAC         | Digital-to-analog converter |
| DBG         | Debug                       |

| Peripherals | Descriptions   |
|-------------|--|
| DMA         | Direct memory access controller                          |
| EXMC        | External memory controller                               |
| EXTI        | Interrupt/event controller                               |
| FMC         | Flash memory controller                                  |
| FWDGT       | Free watchdog timer                                      |
| GPIO/AFIO   | General-purpose and alternate-function I/Os              |
| I2C         | Inter-integrated circuit interface                       |
| MISC        | Nested Vectored Interrupt Controller                     |
| PMU         | Power management unit                                    |
| RCU         | Reset and clock unit                                     |
| RTC         | Real-time Clock  |
| SPI/I2S     | Serial peripheral interface/Inter-IC sound               |
| TIMER       | TIMER  |
| 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:

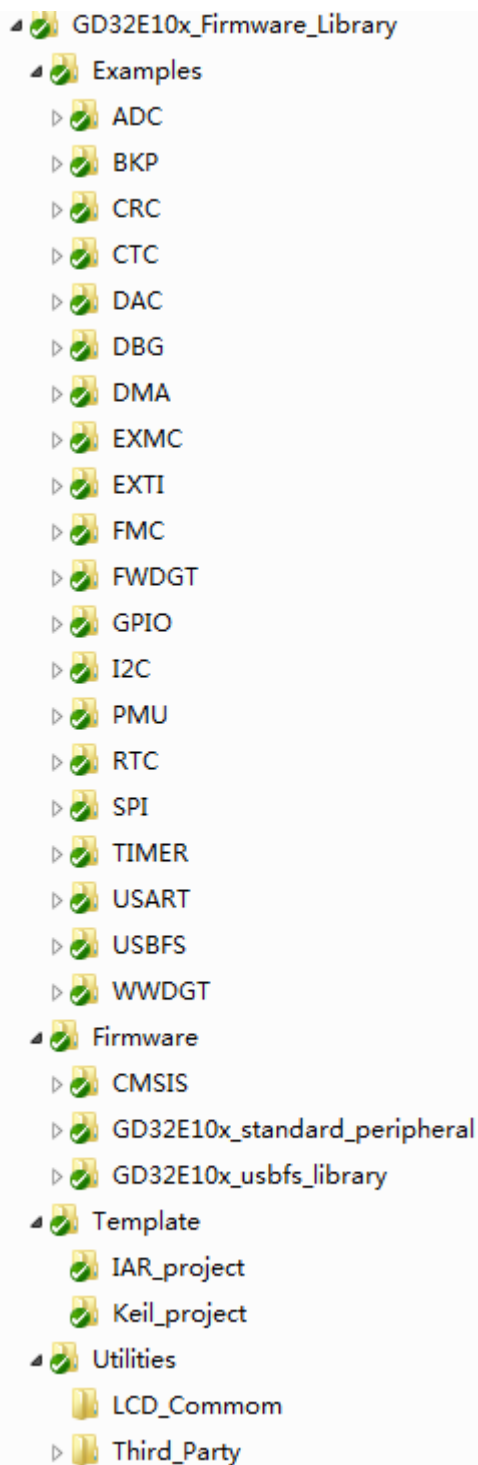
- The peripherals are shortened in XXX format, such as: ADC. More shorten information of peripherals refer to [Peripherals](#);
- The name of sourcefile and header file are started with “gd32e10x\_”, such as: gd32e10x\_adc.h;
- The constants used only in one file should be defined in the used file; the 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

GD32E10x\_Firmware\_Library, the file structure is shown as below:

**Figure 2-1. File structure of firmware library of GD32E10x**



### 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;
- `gd32e10x_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);
- `gd32e10x_it.c`: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- `gd32e10x_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 GD32E10x and system configuration file;
- GD32E10x\_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;
- GD32E10x\_usbfs\_library subfolder includes all the related files about USBFS peripheral: 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.

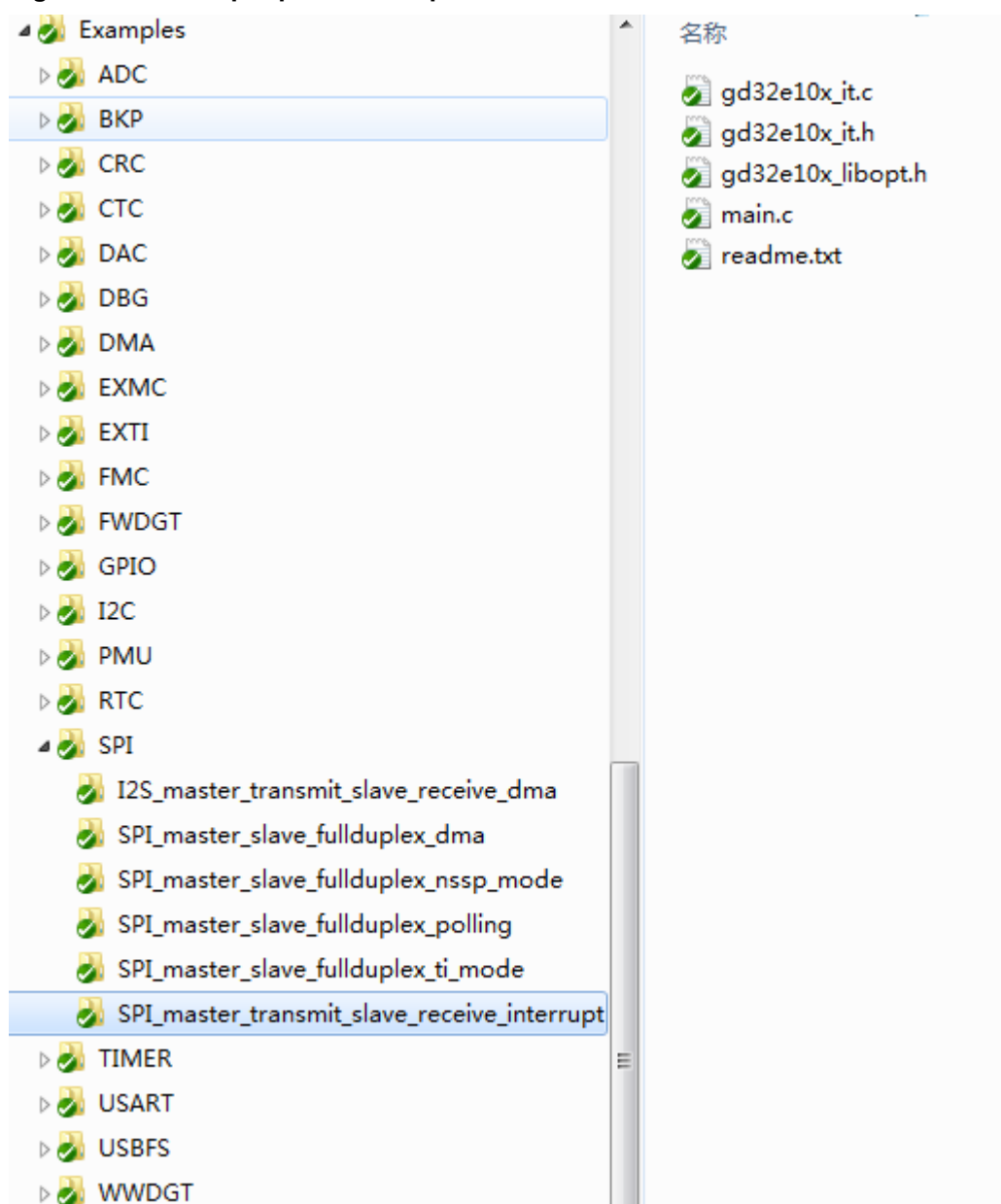
### 2.1.3. Template Folder

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 formware examples, the steps are shown as below:

#### Select files

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

Figure 2-2. Select peripheral example files

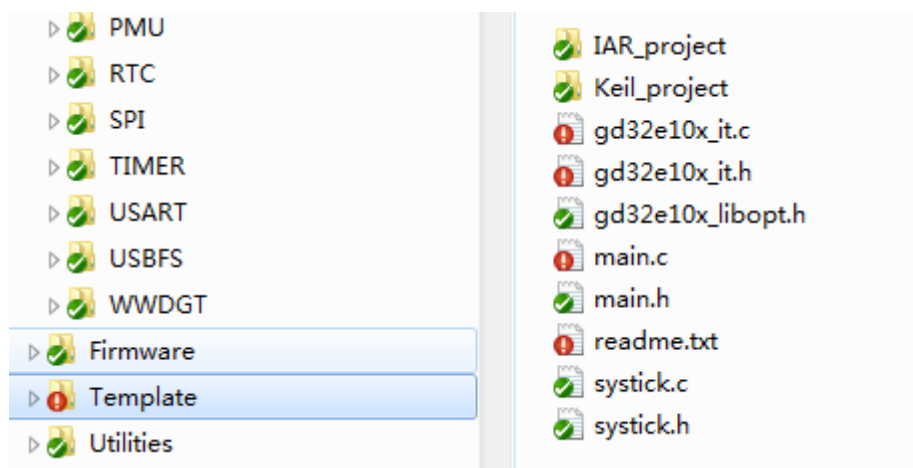


### Copy files

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



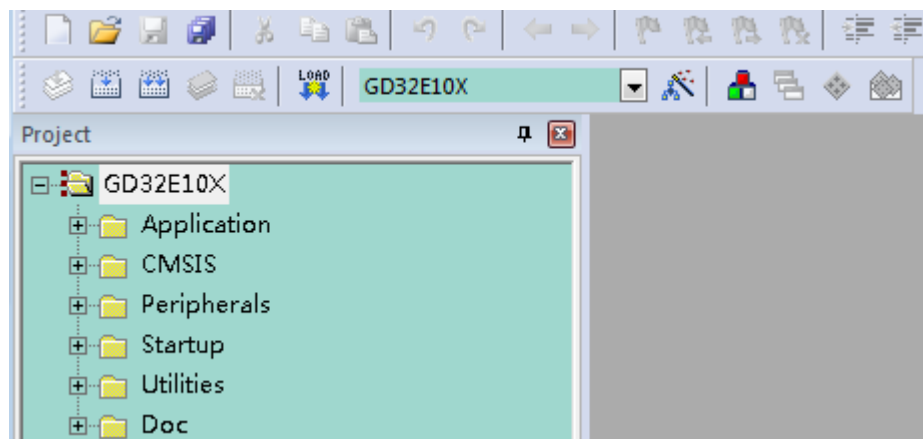
Figure 2-3. Copy the peripheral example files



### Open a project

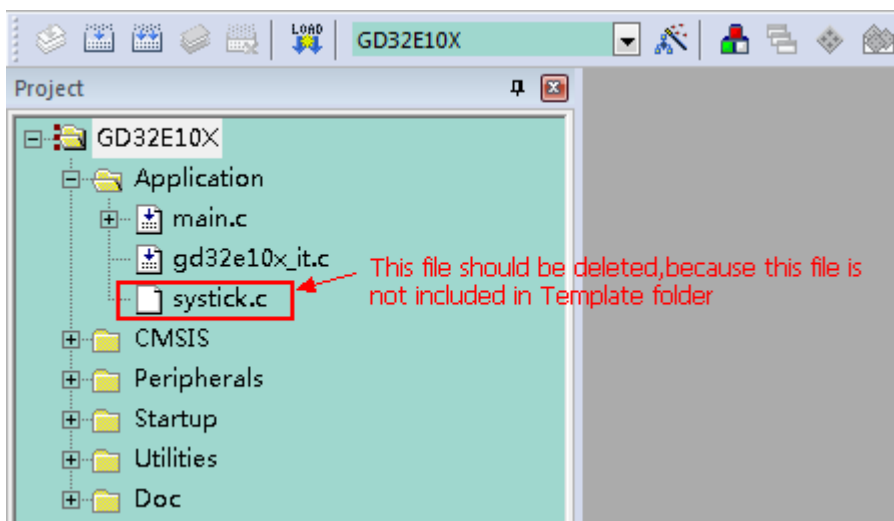
GD provides project in Keil and IAR, users can open project in different IDEs according to their need, such as "Keil\_project", open \Template\Keil\_project\Project.uvproj, shown as below:

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:

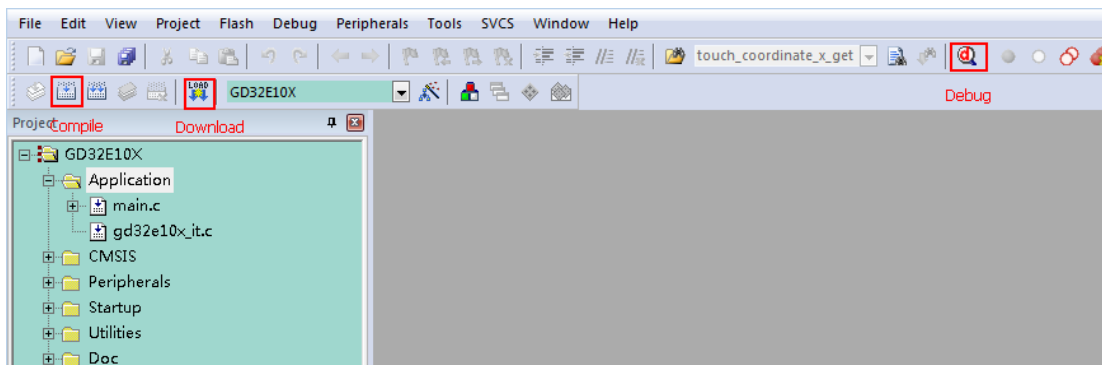
**Figure 2-5. Configure project files**



## 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:

- LCD\_Commom and Third\_Party subfolders include files for USB tests;
- gd32e10x\_eval.h and gd32e10x\_lcd\_eval.h are related header files of the evaluation board about running the firmware examples;
- gd32e10x\_eval.c and gd32e10x\_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   |
|-------------------|--|
| gd32e10x_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.  |
| gd32e10x_it.h     | Header file, including all the prototypes of interrupt service routines.   |
| gd32e10x_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. |
| gd32e10x_xxx.h    | The header file of peripheral PPP, including functions about peripheral PPP, and the variables used for functions.   |
| gd32e10x_xxx.c    | The C source file for driving peripheral PPP.  |
| systick.h         | The header file of systick.c, including prototypes of systick configuration function and delay function.   |
| systick.c         | The source file about systick configuration function and delay function.   |
| readme.txt        | Description document about how to configure and how to use the firmware example.   |

## 3. Firmware Library of Standard Peripherals

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

The description format of firmware functions are shown as below:

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

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

### 3.2. ADC

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

#### 3.2.1. Descriptions of Peripheral registers

ADC registers are listed in the table shown as below:

**Table 3-2. ADC Registers**

| <b>Registers</b>   | <b>Descriptions</b>                             |
|--------------------|---|
| ADC_STAT           | Status register                                 |
| ADC_CTL0           | Control register 0                              |
| ADC_CTL1           | Control register 1                              |
| ADC_SAMPT0         | Sample time register 0                          |
| ADC_SAMPT1         | Sample time register 1                          |
| ADC_IOFFx (x=0..3) | 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_mode_config                    | configure the ADC sync mode   |
| adc_special_function_config        | enable or disable ADC special function                                  |
| adc_data_alignment_config          | configure ADC data alignment  |
| adc_enable                         | enable ADC interface  |
| adc_disable                        | disable ADC interface   |
| adc_calibration_enable             | ADC calibration and reset calibration                                   |
| adc_tempsensor_vrefint_enable      | enable the temperature sensor and Vrefint channel                       |
| adc_tempsensor_vrefint_disable     | disable the temperature sensor and Vrefint channel                      |
| adc_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_dma_mode_enable                | enable DMA request  |
| adc_dma_mode_disable               | disable DMA request   |
| adc_discontinuous_mode_config      | configure ADC discontinuous mode  |
| adc_channel_length_config          | configure the length of regular channel group or inserted channel group |
| adc_regular_channel_config         | configure ADC regular channel   |
| adc_inserted_channel_config        | configure ADC inserted channel  |
| adc_inserted_channel_offset_config | configure ADC inserted channel offset                                   |
| adc_external_trigger_source_config | configure ADC external trigger source                                   |
| adc_external_trigger_config        | enable ADC external trigger   |
| adc_software_trigger_enable        | enable ADC software trigger   |
| adc_regular_data_read              | read ADC regular group data register                                    |
| adc_inserted_data_read             | read ADC inserted group data register                                   |
| adc_sync_mode_convert_value_read   | read the last ADC0 and ADC1 conversion result data in sync mode         |

| Function name                            | Function description   |
|--|--|
| adc_watchdog_single_channel_enable       | configure ADC analog watchdog single channel                         |
| adc_watchdog_group_channel_enable        | configure ADC analog watchdog group channel                          |
| adc_watchdog_disable                     | disable ADC analog watchdog  |
| adc_watchdog_threshold_config            | configure ADC analog watchdog threshold                              |
| adc_flag_get                             | get the ADC flag bits  |
| adc_flag_clear                           | clear the ADC flag bits  |
| adc_regular_software_startconv_flag_get  | get the bit state of ADCx software start conversion                  |
| adc_inserted_software_startconv_flag_get | get the bit state of ADCx software inserted channel start conversion |
| adc_interrupt_flag_get                   | get the ADC interrupt bits   |
| adc_interrupt_flag_clear                 | clear the ADC flag   |
| adc_interrupt_enable                     | enable ADC interrupt   |
| adc_interrupt_disable                    | disable ADC interrupt  |

## adc\_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(uint32_t adc_periph);              |
| Function descriptions | reset ADCx peripheral                              |
| Precondition          | -  |
| The called functions  | rcu_periph_reset_enable / rcu_periph_reset_disable |
| Input parameter{in}   |  |
| adc_periph            | ADC peripheral                                     |
| ADCx(x=0,1)           | ADC peripheral selection                           |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

Example:

```
/* reset ADC0 */
adc_deinit(ADC0);
```

## adc\_mode\_config

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

**Table 3-5. Function `adc_mode_config`**

|  |  |
|--|--|
| <b>Function name</b>   | <code>adc_mode_config</code>   |
| <b>Function prototype</b>  | <code>void adc_mode_config(uint32_t mode);</code>                        |
| <b>Function descriptions</b>                                     | configure the ADCs sync mode   |
| <b>Precondition</b>  | -  |
| <b>The called functions</b>                                      | -  |
| <b>Input parameter{in}</b>                                       |  |
| <b>mode</b>  | ADC mode   |
| <code>ADC_MODE_FREE</code>                                       | all the ADC work independently   |
| <code>ADC_DUAL_REGULAR_PARALLEL_INSERTED_PARALLEL</code>         | ADC0 and ADC1 work in combined regular parallel + inserted parallel mode |
| <code>ADC_DUAL_REGULAR_PARALLEL_INSERTED_TRIGGER_ROTATION</code> | ADC0 and ADC1 work in combined regular parallel + trigger rotation mode  |
| <code>ADC_DUAL_INSERTED_PARALLEL_REGULAR_FOLLOWUP_FAST</code>    | ADC0 and ADC1 work in combined inserted parallel + follow-up fast mode   |
| <code>ADC_DUAL_INSERTED_PARALLEL_REGULAR_FOLLOWUP_SLOW</code>    | ADC0 and ADC1 work in combined inserted parallel + follow-up slow mode   |
| <code>ADC_DUAL_INSERTED_PARALLEL</code>                          | ADC0 and ADC1 work in inserted parallel mode only                        |
| <code>ADC_DUAL_REGULAR_PARALLEL</code>                           | ADC0 and ADC1 work in regular parallel mode only                         |
| <code>ADC_DUAL_REGULAR_FOLLOWUP_FAST</code>                      | ADC0 and ADC1 work in follow-up fast mode only                           |
| <code>ADC_DUAL_REGULAR_FOLLOWUP_SLOW</code>                      | ADC0 and ADC1 work in follow-up slow mode only                           |
| <code>ADC_DUAL_INSERTED_TRIGGER_ROTATION</code>                  | ADC0 and ADC1 work in trigger rotation mode only                         |
| <b>Output parameter{out}</b>                                     |  |
| -  | -  |
| <b>Return value</b>  |  |
| -  | -  |

Example:

```
/* configure the ADC sync mode */
adc_mode_config(ADC_MODE_FREE);
```

## adc\_special\_function\_config

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

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

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

Example:

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

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

## adc\_data\_alignment\_config

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

**Table 3-7. Function adc\_data\_alignment\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | adc_data_alignment_config   |
| <b>Function prototype</b>    | void adc_data_alignment_config(uint32_t adc_periph, 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>adc_periph</b>            | ADC peripheral           |
| <i>ADCx(x=0,1)</i>           | ADC peripheral selection |
| <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 ADC0 data alignment */
adc_data_alignment_config(ADC0, ADC_DATAALIGN_RIGHT);
```

## adc\_enable

The description of adc\_enable is shown as below:

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

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

Example:

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

## adc\_disable

The description of adc\_disable is shown as below:

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

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

Example:

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

## adc\_calibration\_enable

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

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

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

Example:

```
/* ADC0 calibration and reset calibration */
adc_calibration_enable(ADC0);
```

## adc\_tempsensor\_vrefint\_enable

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

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

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

Example:

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

```
adc_tempsensor_vrefint_enable();
```

## adc\_tempsensor\_vrefint\_disable

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

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

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

Example:

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

```
adc_tempsensor_vrefint_disable();
```

## adc\_resolution\_config

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

**Table 3-13. Function adc\_resolution\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | adc_resolution_config  |
| <b>Function prototype</b>    | void adc_resolution_config(uint32_t adc_periph , uint32_t resolution); |
| <b>Function descriptions</b> | configure ADC resolution   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>adc_periph</b>            | ADC peripheral   |
| <i>ADCx(x=0,1)</i>           | ADC peripheral selection   |
| <b>Input parameter{in}</b>   |  |
| <b>resolution</b>            | ADC resolution   |
| <i>ADC_RESOLUTION_12B</i>    | 12-bit ADC resolution  |
| <i>ADC_RESOLUTION_10B</i>    | 10-bit ADC resolution  |
| <i>ADC_RESOLUTION_8B</i>     | 8-bit ADC resolution   |
| <i>ADC_RESOLUTION_6B</i>     | 6-bit ADC resolution   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure ADC0 data alignment */
adc_data_alignment_config(ADC0, ADC_DATAALIGN_RIGHT);
```

## adc\_oversample\_mode\_config

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

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

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | adc_oversample_mode_config  |
| <b>Function prototype</b>    | void adc_oversample_mode_config(uint32_t adc_periph, uint32_t mode, uint16_t shift, uint8_t ratio); |
| <b>Function descriptions</b> | configure ADC oversample mode   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |

| Input parameter{in}                      |  |
|--|--|
| <b>adc_periph</b>                        | ADC peripheral   |
| <i>ADCx(x=0,1)</i>                       | ADC peripheral selection   |
| Input parameter{in}                      |  |
| <b>mode</b>                              | ADC oversampling mode  |
| <i>ADC_OVERSAMPLING<br/>_ALL_CONVERT</i> | all oversampled conversions for a channel are done consecutively after a trigger |
| <i>ADC_OVERSAMPLING<br/>_ONE_CONVERT</i> | each oversampled conversion for a channel needs a trigger                        |
| Input parameter{in}                      |  |
| <b>shift</b>                             | ADC oversampling shift   |
| <i>ADC_OVERSAMPLING<br/>_SHIFT_NONE</i>  | no oversampling shift  |
| <i>ADC_OVERSAMPLING<br/>_SHIFT_1B</i>    | 1-bit oversampling shift   |
| <i>ADC_OVERSAMPLING<br/>_SHIFT_2B</i>    | 2-bit oversampling shift   |
| <i>ADC_OVERSAMPLING<br/>_SHIFT_3B</i>    | 3-bit oversampling shift   |
| <i>ADC_OVERSAMPLING<br/>_SHIFT_4B</i>    | 4-bit oversampling shift   |
| <i>ADC_OVERSAMPLING<br/>_SHIFT_5B</i>    | 5-bit oversampling shift   |
| <i>ADC_OVERSAMPLING<br/>_SHIFT_6B</i>    | 6-bit oversampling shift   |
| <i>ADC_OVERSAMPLING<br/>_SHIFT_7B</i>    | 7-bit oversampling shift   |
| <i>ADC_OVERSAMPLING<br/>_SHIFT_8B</i>    | 8-bit oversampling shift   |
| Input parameter{in}                      |  |
| <b>ratio</b>                             | ADC oversampling ratio   |
| <i>ADC_OVERSAMPLING<br/>_RATIO_MUL2</i>  | oversampling ratio multiple 2  |
| <i>ADC_OVERSAMPLING<br/>_RATIO_MUL4</i>  | oversampling ratio multiple 4  |
| <i>ADC_OVERSAMPLING<br/>_RATIO_MUL8</i>  | oversampling ratio multiple 8  |
| <i>ADC_OVERSAMPLING<br/>_RATIO_MUL16</i> | oversampling ratio multiple 16   |
| <i>ADC_OVERSAMPLING<br/>_RATIO_MUL32</i> | oversampling ratio multiple 32   |
| <i>ADC_OVERSAMPLING<br/>_RATIO_MUL64</i> | oversampling ratio multiple 64   |

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

Example:

```
/* configure ADC1 oversample mode: 16 times sample, 4 bits shift */
```

```
adc_oversample_mode_config(ADC1, 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 3-15. Function adc\_oversample\_mode\_enable**

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

Example:

```
/* enable ADC0 oversample mode */
```

```
adc_oversample_mode_enable (ADC0);
```

## adc\_oversample\_mode\_disable

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

**Table 3-16. Function adc\_oversample\_mode\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | adc_oversample_mode_disable                            |
| <b>Function prototype</b>    | void adc_oversample_mode_disable(uint32_t adc_periph); |
| <b>Function descriptions</b> | disable ADC oversample mode                            |

|                              |                          |
|------------------------------|--------------------------|
| <b>Precondition</b>          | -                        |
| <b>The called functions</b>  | -                        |
| <b>Input parameter{in}</b>   |                          |
| <b>adc_periph</b>            | ADC peripheral           |
| <i>ADCx(x=0,1)</i>           | ADC peripheral selection |
| <b>Output parameter{out}</b> |                          |
| -                            | -                        |
| <b>Return value</b>          |                          |
| -                            | -                        |

Example:

```
/* disable ADC0 oversample mode */
```

```
adc_oversample_mode_disable (ADC0);
```

## adc\_dma\_mode\_enable

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

**Table 3-17. Function adc\_dma\_mode\_enable**

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

Example:

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

```
adc_dma_mode_enable(ADC0);
```

## adc\_dma\_mode\_disable

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

**Table 3-18. Function adc\_dma\_mode\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | adc_dma_mode_disable                            |
| <b>Function prototype</b>    | void adc_dma_mode_disable(uint32_t adc_periph); |
| <b>Function descriptions</b> | disable ADCx DMA request                        |

|                              |                          |
|------------------------------|--------------------------|
| <b>Precondition</b>          | -                        |
| <b>The called functions</b>  | -                        |
| <b>Input parameter{in}</b>   |                          |
| <b>adc_periph</b>            | ADC peripheral           |
| <i>ADCx(x=0,1)</i>           | ADC peripheral selection |
| <b>Output parameter{out}</b> |                          |
| -                            | -                        |
| <b>Return value</b>          |                          |
| -                            | -                        |

Example:

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

```
adc_dma_mode_disable(ADC0);
```

## adc\_discontinuous\_mode\_config

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

**Table 3-19. Function adc\_discontinuous\_mode\_config**

|  |  |
|--|--|
| <b>Function name</b>                     | adc_discontinuous_mode_config  |
| <b>Function prototype</b>                | void adc_discontinuous_mode_config(uint32_t adc_periph, uint8_t adc_channel_group, uint8_t length);                                    |
| <b>Function descriptions</b>             | configure ADC discontinuous mode   |
| <b>Precondition</b>                      | -  |
| <b>The called functions</b>              | -  |
| <b>Input parameter{in}</b>               |  |
| <b>adc_periph</b>                        | ADC peripheral   |
| <i>ADCx(x=0,1)</i>                       | ADC peripheral selection   |
| <b>Input parameter{in}</b>               |  |
| <b>adc_channel_group</b>                 | select the channel group   |
| <i>ADC_REGULAR_CHANNEL</i>               | regular channel group  |
| <i>ADC_INSERTED_CHANNEL</i>              | inserted channel group   |
| <i>ADC_CHANNEL_DISCONTINUOUS_DISABLE</i> | disable discontinuous mode of regular and inserted channel   |
| <b>Input parameter{in}</b>               |  |
| <b>length</b>                            | number of conversions in discontinuous mode, the number can be 1..8 for regular channel, the number has no effect for inserted channel |
| <b>Output parameter{out}</b>             |  |
| -  | -  |
| <b>Return value</b>                      |  |
| -  | -  |



Example:

```
/* configure ADC0 regular channel group discontinuous mode */  
  
adc_discontinuous_mode_config(ADC0, ADC_REGULAR_CHANNEL, 6);
```

## adc\_channel\_length\_config

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

**Table 3-20. Function adc\_channel\_length\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | adc_channel_length_config  |
| <b>Function prototype</b>    | void adc_channel_length_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t length); |
| <b>Function descriptions</b> | configure the length of regular channel group or inserted channel group                          |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>adc_periph</b>            | ADC peripheral   |
| ADCx(x=0,1)                  | ADC peripheral selection   |
| <b>Input parameter{in}</b>   |  |
| <b>adc_channel_group</b>     | select the channel group   |
| ADC_REGULAR_CHANNEL          | regular channel group  |
| ADC_INSERTED_CHANNEL         | inserted channel group   |
| <b>Input parameter{in}</b>   |  |
| <b>length</b>                | the length of the channel, regular channel 1-16, inserted channel 1-4                            |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure the length of ADC0 regular channel */  
  
adc_channel_length_config(ADC0, ADC_REGULAR_CHANNEL, 4);
```

## adc\_regular\_channel\_config

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

**Table 3-21. Function adc\_regular\_channel\_config**

|                           |  |
|---------------------------|--|
| <b>Function name</b>      | adc_regular_channel_config   |
| <b>Function prototype</b> | void adc_regular_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time); |

|                                      |   |
|--------------------------------------|---|
| <b>Function descriptions</b>         | configure ADC regular channel   |
| <b>Precondition</b>                  | -   |
| <b>The called functions</b>          | -   |
| <b>Input parameter{in}</b>           |   |
| <b>adc_periph</b>                    | ADC peripheral  |
| <i>ADCx(x=0,1)</i>                   | ADC peripheral selection  |
| <b>Input parameter{in}</b>           |   |
| <b>rank</b>                          | the regular group sequence rank, this parameter must be between 0 to 15 |
| <b>Input parameter{in}</b>           |   |
| <b>adc_channel</b>                   | the selected ADC channel  |
| <i>ADC_CHANNEL_x(x=0..17)</i>        | ADC Channelx (x=0..17)(x=16 and x=17 are only for ADC0)                 |
| <b>Input parameter{in}</b>           |   |
| <b>sample_time</b>                   | the sample time value   |
| <i>ADC_SAMPLETIME_1<br/>POINT5</i>   | 1.5 cycles  |
| <i>ADC_SAMPLETIME_7<br/>POINT5</i>   | 7.5 cycles  |
| <i>ADC_SAMPLETIME_1<br/>3POINT5</i>  | 13.5 cycles   |
| <i>ADC_SAMPLETIME_2<br/>8POINT5</i>  | 28.5 cycles   |
| <i>ADC_SAMPLETIME_4<br/>1POINT5</i>  | 41.5 cycles   |
| <i>ADC_SAMPLETIME_5<br/>5POINT5</i>  | 55.5 cycles   |
| <i>ADC_SAMPLETIME_7<br/>1POINT5</i>  | 71.5 cycles   |
| <i>ADC_SAMPLETIME_2<br/>39POINT5</i> | 239.5 cycles  |
| <b>Output parameter{out}</b>         |   |
| -                                    | -   |
| <b>Return value</b>                  |   |
| -                                    | -   |

Example:

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

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

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

The description of `adc_inserted_channel_config` is shown as below:

**Table 3-22. 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(uint32_t adc_periph, uint8_t rank, uint8_t adc_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>adc_periph</b>                     | ADC peripheral   |
| <code>ADCx(0,1)</code>                | ADC peripheral selection   |
| <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>adc_channel</b>                    | the selected ADC channel   |
| <code>ADC_CHANNEL_x(x=0..17)</code>   | ADC Channelx (x=0..17)(x=16 and x=17 are only for ADC0)  |
| <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  |
| <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 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 3-23. Function adc\_inserted\_channel\_offset\_config**

|                                |  |
|--------------------------------|--|
| <b>Function name</b>           | adc_inserted_channel_offset_config   |
| <b>Function prototype</b>      | void adc_inserted_channel_offset_config(uint32_t adc_periph, uint8_t inserted_channel, uint16_t offset); |
| <b>Function descriptions</b>   | configure ADC inserted channel offset  |
| <b>Precondition</b>            | -  |
| <b>The called functions</b>    | -  |
| <b>Input parameter{in}</b>     |  |
| <b>adc_periph</b>              | ADC peripheral   |
| ADCx(x=0,1)                    | ADC peripheral selection   |
| <b>Input parameter{in}</b>     |  |
| <b>inserted_channel</b>        | insert channel select  |
| ADC_INSERTED_CHANNEL_x(x=0..3) | inserted channel, x=0,1,2,3  |
| <b>Input parameter{in}</b>     |  |
| <b>offset</b>                  | the offset data, this parameter must be between 0 to 4095  |
| <b>Output parameter{out}</b>   |  |
| -                              | -  |
| <b>Return value</b>            |  |
| -                              | -  |

Example:

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

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

## adc\_external\_trigger\_source\_config

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

**Table 3-24. 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(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t external_trigger_source); |
| <b>Function descriptions</b> | configure ADC external trigger source  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>adc_periph</b>            | ADC peripheral   |
| ADCx(x=0,1)                  | ADC peripheral selection   |
| <b>Input parameter{in}</b>   |  |

|   |   |
|---|---|
| <b>adc_channel_group</b>                  | select the channel group                        |
| <i>ADC_REGULAR_CHANNEL</i>                | regular channel group                           |
| <i>ADC_INSERTED_CHANNEL</i>               | inserted channel group                          |
| <b>Input parameter{in}</b>                |   |
| <b>external_trigger_source</b>            | regular or inserted group trigger source        |
| <i>ADC0_1_EXTTRIGGER_REGULAR_T0_CH0</i>   | TIMER0 CH0 event select for regular channel     |
| <i>ADC0_1_EXTTRIGGER_REGULAR_T0_CH1</i>   | TIMER0 CH1 event select for regular channel     |
| <i>ADC0_1_EXTTRIGGER_REGULAR_T0_CH2</i>   | TIMER0 CH2 event select for regular channel     |
| <i>ADC0_1_EXTTRIGGER_REGULAR_T1_CH1</i>   | TIMER1 CH1 event select for regular channel     |
| <i>ADC0_1_EXTTRIGGER_REGULAR_T2_TRGO</i>  | TIMER2 TRGO event select for regular channel    |
| <i>ADC0_1_EXTTRIGGER_REGULAR_T3_CH3</i>   | TIMER3 CH3 event select for regular channel     |
| <i>ADC0_1_EXTTRIGGER_REGULAR_T7_TRGO</i>  | TIMER7 TRGO event select for regular channel    |
| <i>ADC0_1_EXTTRIGGER_REGULAR_EXTI_11</i>  | external interrupt line 11 for regular channel  |
| <i>ADC0_1_EXTTRIGGER_REGULAR_NONE</i>     | software trigger for regular channel            |
| <i>ADC0_1_EXTTRIGGER_INSERTED_T0_TRGO</i> | TIMER0 TRGO event select for inserted channel   |
| <i>ADC0_1_EXTTRIGGER_INSERTED_T0_CH3</i>  | TIMER0 CH3 event select for inserted channel    |
| <i>ADC0_1_EXTTRIGGER_INSERTED_T1_TRGO</i> | TIMER1 TRGO event select for inserted channel   |
| <i>ADC0_1_EXTTRIGGER_INSERTED_T1_CH0</i>  | TIMER1 CH0 event select for inserted channel    |
| <i>ADC0_1_EXTTRIGGER_INSERTED_T2_CH3</i>  | TIMER2 CH3 event select for inserted channel    |
| <i>ADC0_1_EXTTRIGGER_INSERTED_T3_TRGO</i> | TIMER3 TRGO event select for inserted channel   |
| <i>ADC0_1_EXTTRIGGER_INSERTED_EXTI_15</i> | external interrupt line 15 for inserted channel |
| <i>ADC0_1_EXTTRIGGER_INSERTED_T7_CH3</i>  | TIMER7 CH3 event select for inserted channel    |
| <i>ADC0_1_EXTTRIGGER_INSERTED_NONE</i>    | software trigger for inserted channel           |

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

Example:

```
/* configure ADC0 regular channel external trigger source */
```

```
adc_external_trigger_source_config(ADC0,ADC_REGULAR_CHANNEL,  
ADC0_1_EXTTRIG_REGULAR_T0_CH0);
```

## adc\_external\_trigger\_config

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

**Table 3-25. Function adc\_external\_trigger\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | adc_external_trigger_config   |
| <b>Function prototype</b>    | void adc_external_trigger_config(uint32_t adc_periph, uint8_t adc_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>adc_periph</b>            | ADC peripheral  |
| <i>ADCx(x=0,1)</i>           | ADC peripheral selection  |
| <b>Input parameter{in}</b>   |   |
| <b>adc_channel_group</b>     | select the channel group  |
| <i>ADC_REGULAR_CHANNEL</i>   | regular channel group   |
| <i>ADC_INSERTED_CHANNEL</i>  | inserted channel group  |
| <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 ADC0 inserted channel group external trigger */
```

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

## adc\_software\_trigger\_enable

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

**Table 3-26. Function adc\_software\_trigger\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | adc_software_trigger_enable   |
| <b>Function prototype</b>    | void adc_software_trigger_enable(uint32_t adc_periph, uint8_t adc_channel_group); |
| <b>Function descriptions</b> | enable ADC software trigger   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>adc_periph</b>            | ADC peripheral  |
| <i>ADCx(x=0,1)</i>           | ADC peripheral selection  |
| <b>Input parameter{in}</b>   |   |
| <b>adc_channel_group</b>     | select the channel group  |
| <i>ADC_REGULAR_CHANNEL</i>   | regular channel group   |
| <i>ADC_INSERTED_CHANNEL</i>  | inserted channel group  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable ADC0 regular channel group software trigger */
```

```
adc_software_trigger_enable(ADC0, ADC_REGULAR_CHANNEL);
```

## adc\_regular\_data\_read

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

**Table 3-27. Function adc\_regular\_data\_read**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | adc_regular_data_read                                |
| <b>Function prototype</b>    | uint16_t adc_regular_data_read(uint32_t adc_periph); |
| <b>Function descriptions</b> | read ADC regular group data register                 |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>adc_periph</b>            | ADC peripheral                                       |
| <i>ADCx(x=0,1)</i>           | ADC peripheral selection                             |
| <b>Output parameter{out}</b> |  |
| -                            | -  |

| Return value |                                 |
|--------------|---------------------------------|
| uint16_t     | ADC conversion value (0-0xFFFF) |

Example:

```
/* read ADC0 regular group data register */
```

```
uint16_t adc_value = 0;
```

```
adc_value = adc_regular_data_read(ADC0);
```

## adc\_inserted\_data\_read

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

**Table 3-28. Function adc\_inserted\_data\_read**

| Function name  | adc_inserted_data_read  |
|--|---|
| Function prototype   | uint16_t adc_inserted_data_read(uint32_t adc_periph, uint8_t inserted_channel); |
| Function descriptions  | read ADC inserted group data register   |
| Precondition   | -   |
| The called functions   | -   |
| Input parameter{in}  |   |
| adc_periph   | ADC peripheral  |
| ADCx(x=0,1)  | ADC peripheral selection  |
| Input parameter{in}  |   |
| inserted_channel   | insert channel select   |
| ADC_INSERTED_CHANNEL_0<br>ADC_INSERTED_CHANNEL_1<br>ADC_INSERTED_CHANNEL_2<br>ADC_INSERTED_CHANNEL_3 | inserted Channelx, x=0,1,2,3  |
| Output parameter{out}  |   |
| -  | -   |
| Return value   |   |
| uint16_t   | ADC conversion value (0-0xFFFF)   |

Example:

```
/* read ADC0 inserted group data register */
```

```
uint16_t adc_value = 0;
```

```
adc_value = adc_inserted_data_read (ADC0, ADC_INSERTED_CHANNEL_0);
```

## adc\_sync\_mode\_convert\_value\_read

The description of adc\_sync\_mode\_convert\_value\_read is shown as below:

**Table 3-29. Function adc\_sync\_mode\_convert\_value\_read**

| Function name      | adc_sync_mode_convert_value_read                 |
|--------------------|--|
| Function prototype | uint32_t adc_sync_mode_convert_value_read(void); |



|                              |   |
|------------------------------|---|
| <b>Function descriptions</b> | read the last ADC0 and ADC1 conversion result data in sync mode |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| uint32_t                     | ADC conversion value (0-0xFFFFFFFF)                             |

Example:

```
/* read the last ADC0 and ADC1 conversion result data in sync mode */

uint32_t adc_value = 0;

adc_value = adc_sync_mode_convert_value_read ();
```

## adc\_watchdog\_single\_channel\_enable

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

**Table 3-30. Function adc\_watchdog\_single\_channel\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | adc_watchdog_single_channel_enable   |
| <b>Function prototype</b>    | void adc_watchdog_single_channel_enable(uint32_t adc_periph, uint8_t adc_channel); |
| <b>Function descriptions</b> | configure ADC analog watchdog single channel                                       |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| adc_periph                   | ADC peripheral   |
| ADCx(x=0,1)                  | ADC peripheral selection   |
| <b>Input parameter{in}</b>   |  |
| adc_channel                  | the selected ADC channel   |
| ADC_CHANNEL_x(x=0..17)       | ADC Channelx(x=0..17) (x=16 and x=17 are only for ADC0)                            |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure ADC0 analog watchdog single channel */

adc_watchdog_single_channel_enable(ADC0, ADC_CHANNEL_1);
```

## adc\_watchdog\_group\_channel\_enable

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

**Table 3-31. Function adc\_watchdog\_group\_channel\_enable**

|                                     |   |
|-------------------------------------|---|
| <b>Function name</b>                | adc_watchdog_group_channel_enable   |
| <b>Function prototype</b>           | void adc_watchdog_group_channel_enable(uint32_t adc_periph, uint8_t adc_channel_group); |
| <b>Function descriptions</b>        | configure ADC analog watchdog group channel   |
| <b>Precondition</b>                 | -   |
| <b>The called functions</b>         | -   |
| <b>Input parameter{in}</b>          |   |
| <b>adc_periph</b>                   | ADC peripheral  |
| <i>ADCx(x=0,1)</i>                  | ADC peripheral selection  |
| <b>Input parameter{in}</b>          |   |
| <b>adc_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 ADC0 analog watchdog group channel */
```

```
adc_watchdog_group_channel_enable(ADC0, ADC_REGULAR_CHANNEL);
```

## adc\_watchdog\_disable

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

**Table 3-32. Function adc\_watchdog\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | adc_watchdog_disable                            |
| <b>Function prototype</b>    | void adc_watchdog_disable(uint32_t adc_periph); |
| <b>Function descriptions</b> | disable ADC analog watchdog                     |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>adc_periph</b>            | ADC peripheral                                  |
| <i>ADCx(0,1)</i>             | ADC peripheral selection                        |

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

Example:

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

```
adc_watchdog_disable(ADC0);
```

## adc\_watchdog\_threshold\_config

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

**Table 3-33. Function adc\_watchdog\_threshold\_config**

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

Example:

```
/* configure ADC0 analog watchdog threshold */
```

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

## adc\_flag\_get

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

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

| Function name         | adc_flag_get  |
|-----------------------|---|
| Function prototype    | FlagStatus adc_flag_get(uint32_t adc_periph , uint32_t adc_flag); |
| Function descriptions | get the ADC flag bits   |

|                              |                                       |
|------------------------------|---------------------------------------|
| <b>Precondition</b>          | -                                     |
| <b>The called functions</b>  | -                                     |
| <b>Input parameter{in}</b>   |                                       |
| <b>adc_periph</b>            | ADC peripheral                        |
| <i>ADCx(x=0,1)</i>           | ADC peripheral selection              |
| <b>Input parameter{in}</b>   |                                       |
| <b>adc_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   |
| <b>Output parameter{out}</b> |                                       |
| -                            | -                                     |
| <b>Return value</b>          |                                       |
| <b>FlagStatus</b>            | SET or RESET                          |

Example:

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

```
FlagStatus flag_value;
```

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

## adc\_flag\_clear

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

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

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | adc_flag_clear   |
| <b>Function prototype</b>    | void adc_flag_clear(uint32_t adc_periph, uint32_t adc_flag); |
| <b>Function descriptions</b> | clear the ADC flag bits                                      |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>adc_periph</b>            | ADC peripheral   |
| <i>ADCx(x=0,1)</i>           | ADC peripheral selection                                     |
| <b>Input parameter{in}</b>   |  |
| <b>adc_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   |
| <b>Output parameter{out}</b> |                                       |
| -                            | -                                     |
| <b>Return value</b>          |                                       |
| -                            | -                                     |

Example:

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

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

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

The description of `adc_regular_software_startconv_flag_get` is shown as below:

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

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | <code>adc_regular_software_startconv_flag_get</code>                                  |
| <b>Function prototype</b>    | <code>FlagStatus adc_regular_software_startconv_flag_get(uint32_t adc_periph);</code> |
| <b>Function descriptions</b> | get the bit state of ADCx software regular channel start conversion                   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>adc_periph</b>            | ADC peripheral  |
| <i>ADCx(x=0,1)</i>           | ADC peripheral selection  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>FlagStatus</b>            | SET or RESET  |

Example:

```
/* get the bit state of ADC0 software regular channel start conversion */
```

```
FlagStatus flag_value;
```

```
flag_value = adc_regular_software_startconv_flag_get(ADC0);
```

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

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

**Table 3-37. Function adc\_inserted\_software\_startconv\_flag\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | adc_inserted_software_startconv_flag_get                                  |
| <b>Function prototype</b>    | FlagStatus adc_inserted_software_startconv_flag_get(uint32_t adc_periph); |
| <b>Function descriptions</b> | get the bit state of ADCx software inserted channel start conversion      |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>adc_periph</b>            | ADC peripheral  |
| <i>ADCx(x=0,1)</i>           | ADC peripheral selection  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>FlagStatus</b>            | SET or RESET  |

Example:

```
/* get the bit state of ADC0 software inserted channel start conversion */
```

```
FlagStatus flag_value;
```

```
flag_value = adc_inserted_software_startconv_flag_get(ADC0);
```

## adc\_interrupt\_flag\_get

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

**Table 3-38. Function adc\_interrupt\_flag\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | adc_interrupt_flag_get  |
| <b>Function prototype</b>    | FlagStatus adc_interrupt_flag_get(uint32_t adc_periph, uint32_t adc_interrupt); |
| <b>Function descriptions</b> | get the ADC interrupt bits  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>adc_periph</b>            | ADC peripheral  |
| <i>ADCx(x=0,1)</i>           | ADC peripheral selection  |
| <b>Input parameter{in}</b>   |   |
| <b>adc_interrupt</b>         | the adc interrupt bits  |
| <i>ADC_INT_WDE</i>           | analog watchdog interrupt   |
| <i>ADC_INT_EOC</i>           | end of group conversion interrupt   |
| <i>ADC_INT_EOIC</i>          | end of inserted group conversion interrupt                                      |
| <b>Output parameter{out}</b> |   |

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

Example:

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

```
FlagStatus flag_value;
```

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

## adc\_interrupt\_flag\_clear

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

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

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | adc_interrupt_flag_clear  |
| <b>Function prototype</b>    | void adc_interrupt_flag_clear(uint32_t adc_periph, uint32_t adc_interrupt); |
| <b>Function descriptions</b> | clear the ADC interrupt bits  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>adc_periph</b>            | ADC peripheral  |
| <i>ADCx(x=0,1)</i>           | ADC peripheral selection  |
| <b>Input parameter{in}</b>   |   |
| <b>adc_interrupt</b>         | the adc interrupt bits  |
| <i>ADC_INT_WDE</i>           | analog watchdog interrupt   |
| <i>ADC_INT_EOC</i>           | end of group conversion interrupt   |
| <i>ADC_INT_EOIC</i>          | end of inserted group conversion interrupt                                  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

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

```
adc_interrupt_flag_clear(ADC0, ADC_INT_WDE);
```

## adc\_interrupt\_enable

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

**Table 3-40. Function adc\_interrupt\_enable**

|                           |   |
|---------------------------|---|
| <b>Function name</b>      | adc_interrupt_enable  |
| <b>Function prototype</b> | void adc_interrupt_enable(uint32_t adc_periph, uint32_t adc_interrupt); |

|                              |  |
|------------------------------|--|
| <b>Function descriptions</b> | enable ADC interrupt                       |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>adc_periph</b>            | ADC peripheral                             |
| <i>ADCx(x=0,1)</i>           | ADC peripheral selection                   |
| <b>Input parameter{in}</b>   |  |
| <b>adc_interrupt</b>         | the adc interrupt                          |
| <i>ADC_INT_WDE</i>           | analog watchdog interrupt                  |
| <i>ADC_INT_EOC</i>           | end of group conversion interrupt          |
| <i>ADC_INT_EOIC</i>          | end of inserted group conversion interrupt |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable ADC0 analog watchdog interrupt */
```

```
adc_interrupt_enable(ADC0, ADC_INT_WDE);
```

## adc\_interrupt\_disable

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

**Table 3-41. Function adc\_interrupt\_disable**

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

Example:



```
/* disable ADC0 interrupt */
```

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

## 3.3. BKP

The Backup registers are located in the Backup domain that remains powered-on by  $V_{BAT}$  even if  $V_{DD}$  power is shut down, they are forty two 16-bit (84 bytes) registers for data protection of user application data, and the wake-up action from Standby mode or system reset do not affect these registers. The BKP registers are listed in chapter [3.3.1](#), the BKP firmware functions are introduced in chapter [3.3.2](#).

### 3.3.1. Descriptions of Peripheral registers

BKP registers are listed in the table shown as below:

**Table 3-42. BKP Registers**

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

### 3.3.2. Descriptions of Peripheral functions

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

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

| Function name                              | Function description   |
|--|--|
| bkp_deinit                                 | reset data registers   |
| bkp_data_write                             | write data register  |
| bkp_data_read                              | read data register   |
| bkp_rtc_calibration_output_enable          | enable RTC clock calibration output  |
| bkp_rtc_calibration_output_disable         | disable RTC clock calibration output   |
| bkp_rtc_signal_output_enable               | enable RTC alarm or second signal output                                       |
| bkp_rtc_signal_output_disable              | disable RTC alarm or second signal output                                      |
| bkp_rtc_output_select                      | select RTC output, the RTC output can be select as alarm pulse or second pulse |
| bkp_rtc_clock_output_select                | select RTC clock output  |
| bkp_rtc_clock_calibration_direction_select | select RTC clock calibration direction   |
| bkp_rtc_calibration_value_set              | set RTC clock calibration value  |
| bkp_tamper_detection_enable                | enable tamper detection  |

| Function name                | Function description           |
|------------------------------|--------------------------------|
| bkp_tamper_detection_disable | disable tamper detection       |
| bkp_tamper_active_level_set  | set tamper pin active level    |
| bkp_interrupt_enable         | enable tamper interrupt        |
| bkp_interrupt_disable        | disable tamper interrupt       |
| bkp_flag_get                 | get bkp flag state             |
| bkp_flag_clear               | clear bkp flag state           |
| bkp_interrupt_flag_get       | get bkp interrupt flag state   |
| bkp_interrupt_flag_clear     | clear bkp interrupt flag state |

## bkp\_deinit

The description of bkp\_deinit is shown as below:

**Table 3-44. Function bkp\_deinit**

|                       |  |
|-----------------------|--|
| Function name         | bkp_deinit                                   |
| Function prototype    | void bkp_deinit(void);                       |
| Function descriptions | reset data registers                         |
| Precondition          | -  |
| The called functions  | rcu_bkp_reset_enable / rcu_bkp_reset_disable |
| Input parameter{in}   |  |
| -                     | -  |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

Example:

```
/* reset BKP registers */
bkp_deinit ();
```

## bkp\_data\_write

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

**Table 3-45. Function bkp\_data\_write**

|                       |   |
|-----------------------|---|
| Function name         | bkp_data_write  |
| Function prototype    | void bkp_data_write(bkp_data_register_enum register_number, uint16_t data); |
| Function descriptions | write data register   |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |
| register_number       | refer to bkp_data_register_enum   |

|                                       |   |
|---------------------------------------|---|
| <i>BKP_DATA_x</i> ( <i>x</i> = 0..41) | bkp data register number <i>x</i>         |
| <b>Input parameter{in}</b>            |   |
| <b>data</b>                           | the data to be write in BKP data register |
| 0-0xffff                              | data value                                |
| <b>Output parameter{out}</b>          |   |
| -                                     | -   |
| <b>Return value</b>                   |   |
| -                                     | -   |

Example:

```
/* write BKP data register */
bkp_data_write (BKP_DATA_0, 0x1226);
```

## bkp\_data\_read

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

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

|                                       |  |
|---------------------------------------|--|
| <b>Function name</b>                  | bkp_data_read  |
| <b>Function prototype</b>             | uint16_t bkp_data_read (bkp_data_register_enum register_number); |
| <b>Function descriptions</b>          | read data register   |
| <b>Precondition</b>                   | -  |
| <b>The called functions</b>           | -  |
| <b>Input parameter{in}</b>            |  |
| <b>register_number</b>                | refer to bkp_data_register_enum                                  |
| <i>BKP_DATA_x</i> ( <i>x</i> = 0..41) | bkp data register number <i>x</i>                                |
| <b>Output parameter{out}</b>          |  |
| -                                     | -  |
| <b>Return value</b>                   |  |
| uint16_t                              | 0-0xffff   |

Example:

```
/* read BKP data register */
uint16_t data;
data = bkp_data_read (BKP_DATA_0);
```

## bkp\_rtc\_calibration\_output\_enable

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

**Table 3-47. Function bkp\_rtc\_calibration\_output\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | bkp_rtc_calibration_output_enable             |
| <b>Function prototype</b>    | void bkp_rtc_calibration_output_enable(void); |
| <b>Function descriptions</b> | enable RTC clock calibration output           |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable RTC clock calibration output */
bkp_rtc_calibration_output_enable();
```

## bkp\_rtc\_calibration\_output\_disable

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

**Table 3-48. Function bkp\_rtc\_calibration\_output\_disable**

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

Example:

```
/* disable RTC clock calibration output */
bkp_rtc_calibration_output_disable();
```

## bkp\_rtc\_signal\_output\_enable

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

**Table 3-49. Function bkp\_rtc\_signal\_output\_enable**

|                      |                              |
|----------------------|------------------------------|
| <b>Function name</b> | bkp_rtc_signal_output_enable |
|----------------------|------------------------------|

|                              |   |
|------------------------------|---|
| <b>Function prototype</b>    | void bkp_rtc_signal_output_enable (void); |
| <b>Function descriptions</b> | enable RTC alarm or second signal output  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable RTC alarm or second signal output */
bkp_rtc_signal_output_enable();
```

## bkp\_rtc\_signal\_output\_disable

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

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

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | bkp_rtc_signal_output_disable              |
| <b>Function prototype</b>    | void bkp_rtc_signal_output_disable (void); |
| <b>Function descriptions</b> | disable RTC alarm or second signal output  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable RTC alarm or second signal output */
bkp_rtc_signal_output_disable();
```

## bkp\_rtc\_output\_select

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

**Table 3-51. Function bkp\_rtc\_output\_select**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | bkp_rtc_output_select  |
| <b>Function prototype</b>    | void bkp_rtc_output_select (uint16_t outputsel);                         |
| <b>Function descriptions</b> | select RTC output, the RTC output can be select as alarm pulse or second |

|                                 |  |
|---------------------------------|--|
|                                 | pulse  |
| <b>Precondition</b>             | -  |
| <b>The called functions</b>     | -  |
| <b>Input parameter{in}</b>      |  |
| <b>outputsel</b>                | RTC output selection                           |
| <i>RTC_OUTPUT_ALARM_PULSE</i>   | RTC alarm pulse is selected as the RTC output  |
| <i>RTC_OUTPUT_SECONDS_PULSE</i> | RTC second pulse is selected as the RTC output |
| <b>Output parameter{out}</b>    |  |
| -                               | -  |
| <b>Return value</b>             |  |
| -                               | -  |

Example:

```
/* select RTC output alarm signal output */
```

```
bkp_rtc_output_select (RTC_OUTPUT_ALARM_PULSE);
```

## bkp\_rtc\_clock\_output\_select

The description of bkp\_rtc\_clock\_output\_select is shown as below:

**Table 3-52. Function bkp\_rtc\_clock\_output\_select**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | bkp_rtc_clock_output_select  |
| <b>Function prototype</b>    | void bkp_rtc_clock_output_select(uint16_t clocksel);                                     |
| <b>Function descriptions</b> | select RTC clock output, the RTC clock output can be select as divided 64 or no division |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>clocksel</b>              | RTC clock output selection   |
| <i>RTC_CLOCK_DIV_64</i>      | RTC clock divided 64 is selected as the RTC clock output                                 |
| <i>RTC_CLOCK_DIV_1</i>       | RTC clock is selected as the RTC clock output  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* select RTC clock divided 64 to output */
```

```
bkp_rtc_clock_output_select (RTC_CLOCK_DIV_64);
```

## bkp\_rtc\_clock\_calibration\_direction\_select

The description of bkp\_rtc\_clock\_calibration\_direction\_select is shown as below:

**Table 3-53. Function bkp\_rtc\_clock\_calibration\_direction\_select**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | bkp_rtc_clock_calibration_direction_select   |
| <b>Function prototype</b>    | void bkp_rtc_clock_calibration_direction_select(uint16_t direction);   |
| <b>Function descriptions</b> | select RTC clock calibration direction, the RTC clock calibration direction can be select as slowed down or speed up |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>direction</b>             | RTC clock calibration direction  |
| <i>RTC_CLOCK_SLOW_DOWN</i>   | RTC clock slow down  |
| <i>RTC_CLOCK_SPEED_UP</i>    | RTC clock speed up   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* set RTC clock slowed down */
```

```
bkp_rtc_clock_calibration_direction_select (RTC_CLOCK_SLOWED_DOWN);
```

## bkp\_rtc\_calibration\_value\_set

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

**Table 3-54. Function bkp\_rtc\_calibration\_value\_set**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | bkp_rtc_calibration_value_set                      |
| <b>Function prototype</b>    | void bkp_rtc_calibration_value_set(uint8_t value); |
| <b>Function descriptions</b> | set RTC clock calibration value                    |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>value</b>                 | RTC clock calibration value                        |
| <i>0x00 - 0x7F</i>           | value  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

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

```
bkp_rtc_calibration_value_set (0x7f);
```

## bkp\_tamper\_detection\_enable

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

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

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

Example:

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

```
bkp_tamper_detection_enable();
```

## bkp\_tamper\_detection\_disable

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

**Table 3-56. Function bkp\_tamper\_detection\_disable**

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

Example:

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



bkp\_tamper\_detection\_disable();

## bkp\_tamper\_active\_level\_set

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

**Table 3-57. Function bkp\_tamper\_active\_level\_set**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | bkp_tamper_active_level_set                        |
| <b>Function prototype</b>    | void bkp_tamper_active_level_set (uint16_t level); |
| <b>Function descriptions</b> | set tamper pin active level                        |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>level</b>                 | tamper pin active level                            |
| TAMPER_PIN_ACTIVE_HIGH       | the tamper pin is active high                      |
| TAMPER_PIN_ACTIVE_LOW        | the tamper pin is active low                       |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

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

```
bkp_tamper_active_level_set (TAMPER_PIN_ACTIVE_HIGH);
```

## bkp\_interrupt\_enable

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

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

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

Example:

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

```
bkp_interrupt_enable ();
```

## bkp\_interrupt\_disable

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

**Table 3-59. Function bkp\_interrupt\_disable**

|                              |                                    |
|------------------------------|------------------------------------|
| <b>Function name</b>         | bkp_interrupt_disable              |
| <b>Function prototype</b>    | void bkp_interrupt_disable (void); |
| <b>Function descriptions</b> | disable tamper 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 tamper pin interrupt */
```

```
bkp_interrupt_disable ();
```

## bkp\_flag\_get

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

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

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

Example:

```
/* get BKP flag state */
```

```
FlagStatus status;
```

```
status = bkp_flag_get ();
```

## bkp\_flag\_clear

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

**Table 3-61. Function bkp\_flag\_clear**

|                              |                            |
|------------------------------|----------------------------|
| <b>Function name</b>         | bkp_flag_clear             |
| <b>Function prototype</b>    | void bkp_flag_clear(void); |
| <b>Function descriptions</b> | clear bkp flag state       |
| <b>Precondition</b>          | -                          |
| <b>The called functions</b>  | -                          |
| <b>Input parameter{in}</b>   |                            |
| -                            | -                          |
| <b>Output parameter{out}</b> |                            |
| -                            | -                          |
| <b>Return value</b>          |                            |
| -                            | -                          |

Example:

```
/* clear BKP flag state */
```

```
bkp_flag_clear ();
```

## bkp\_interrupt\_flag\_get

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

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

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

Example:

```
/* get BKP interrupt flag state */
```

bkp\_interrupt\_flag\_get ();

## bkp\_interrupt\_flag\_clear

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

**Table 3-63. Function bkp\_interrupt\_flag\_clear**

|                              |                                      |
|------------------------------|--------------------------------------|
| <b>Function name</b>         | bkp_interrupt_flag_clear             |
| <b>Function prototype</b>    | void bkp_interrupt_flag_clear(void); |
| <b>Function descriptions</b> | clear bkp interrupt flag state       |
| <b>Precondition</b>          | -                                    |
| <b>The called functions</b>  | -                                    |
| <b>Input parameter{in}</b>   |                                      |
| -                            | -                                    |
| <b>Output parameter{out}</b> |                                      |
| -                            | -                                    |
| <b>Return value</b>          |                                      |
| -                            | -                                    |

Example:

```
/* clear BKP interrupt flag state */
```

```
bkp_interrupt_flag_clear ();
```

## 3.4. CRC

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

### 3.4.1. Descriptions of Peripheral registers

CRC registers are listed in the table shown as below:

**Table 3-64. CRC Registers**

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

### 3.4.2. Descriptions of Peripheral functions

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

**Table 3-65. CRC firmware function**

| Function name                | Function description   |
|------------------------------|--|
| crc_deinit                   | deinit CRC calculation unit                                      |
| crc_data_register_reset      | reset data register to the initializaiton value of data register |
| crc_data_register_read       | read the value of the data register                              |
| crc_free_data_register_read  | read the value of the free data register                         |
| crc_free_data_register_write | write data to the free data register                             |
| crc_single_data_calculate    | calculate the CRC value of a 32-bit data                         |
| crc_block_data_calculate     | calculate the CRC value of an array of 32-bit values             |

## crc\_deinit

The description of crc\_deinit is shown as below:

**Table 3-66. Function crc\_deinit**

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

Example:

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

## crc\_data\_register\_reset

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

**Table 3-67. Function crc\_data\_register\_reset**

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

|              |   |
|--------------|---|
| -            | - |
| Return value |   |
| -            | - |

Example:

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

```
crc_data_register_reset ();
```

## crc\_data\_register\_read

The description of `crc_data_register_read` is shown as below:

**Table 3-68. Function `crc_data_register_read`**

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

Example:

```
/* read crc data register */
```

```
uint32_t crc_value = 0;
```

```
crc_value = crc_data_register_read();
```

## crc\_free\_data\_register\_read

The description of `crc_free_data_register_read` is shown as below:

**Table 3-69. Function `crc_free_data_register_read`**

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

|                     |  |
|---------------------|--|
| -                   | -  |
| <b>Return value</b> |  |
| uint8_t             | 8-bit value of the free data register (0-0xFF) |

Example:

```
/* read crc free data register */

uint8_t crc_value = 0;

crc_value = crc_free_data_register_read();
```

## crc\_free\_data\_register\_write

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

**Table 3-70. 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 data to 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\_single\_data\_calculate

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

**Table 3-71. 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> | calculate the CRC value of a 32-bit data            |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>sdata</b>                 | specify 32-bit data                                 |
| <b>Output parameter{out}</b> |   |

|                     |   |
|---------------------|---|
| -                   | -   |
| <b>Return value</b> |   |
| <b>uint32_t</b>     | 32-bit CRC calculate value (0-0xFFFFFFFF) |

Example:

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

uint32_t val = 0, valcrc = 0;

val = (uint32_t)0xabcd1234;

rcu_periph_clock_enable(RCU_CRC);

valcrc = crc_single_data_calculate(val);
```

## crc\_block\_data\_calculate

The description of `crc_block_data_calculate` is shown as below:

**Table 3-72. Function `crc_block_data_calculate`**

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

Example:

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

/* 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);
```



## 3.5. CTC

The CTC unit trims the frequency of the IRC48M which is based on an external accurate reference signal source. It can adjust the calibration value to provide a precise IRC48M clock automatically or manually. The CTC registers are listed in chapter [3.5.1](#), the CTC firmware functions are introduced in chapter [3.5.2](#)

### 3.5.1. Descriptions of Peripheral registers

CTC registers are listed in the table shown as below:

**Table 3-73. 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 |

### 3.5.2. Descriptions of Peripheral functions

CTC registers are listed in the table shown as below:

**Table 3-74. CTC firmware function**

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

| Function name            | Function description      |
|--------------------------|---------------------------|
| ctc_interrupt_disable    | disable the CTC interrupt |
| ctc_interrupt_flag_get   | get CTC interrupt flag    |
| ctc_interrupt_flag_clear | clear CTC interrupt flag  |
| ctc_flag_get             | get CTC flag              |
| ctc_flag_clear           | clear CTC flag            |

## ctc\_deinit

The description of ctc\_deinit is shown as below:

**Table 3-75. Function ctc\_deinit**

|                       |  |
|-----------------------|--|
| Function name         | ctc_deinit   |
| Function prototype    | void ctc_deinit (void)                             |
| Function descriptions | Reset CTC peripheral                               |
| Precondition          | -  |
| The called functions  | rcu_periph_reset_enable / rcu_periph_reset_disable |
| Input parameter{in}   |  |
| -                     | -  |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

Example:

```
/* reset CTC */
ctc_deinit();
```

## ctc\_counter\_enable

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

**Table 3-76. Function ctc\_counter\_enable**

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

Example:

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

```
ctc_counter_enable ();
```

## ctc\_counter\_disable

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

**Table 3-77. Function ctc\_counter\_disable**

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

Example:

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

```
ctc_counter_disable ();
```

## ctc\_irc48m\_trim\_value\_config

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

**Table 3-78. Function ctc\_irc48m\_trim\_value\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | ctc_irc48m_trim_value_config                           |
| <b>Function prototype</b>    | void ctc_irc48m_trim_value_config(uint8_t trim_value); |
| <b>Function descriptions</b> | configure the IRC48M trim value                        |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| 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 3-79. 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 3-80. 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     |
| <i>CTC_HARDWARE_TRIM_MODE_ENABLE</i>  | hardware automatically trim mode enable                |
| <i>CTC_HARDWARE_TRIM_MODE_DISABLE</i> | hardware automatically trim mode disable               |
| <b>Output parameter{out}</b>          |  |
| -                                     | -  |
| <b>Return value</b>                   |  |
| -                                     | -  |

Example:

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

```
ctc_hardware_trim_mode_config (CTC_HARDWARE_TRIM_MODE_ENABLE);
```

## ctc\_refsource\_polarity\_config

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

**Table 3-81. Function ctc\_refsource\_polarity\_config**

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

Example:

```
/* set reference source polarity */
```

```
ctc_refsource_polarity_config (CTC_REFSOURCE_POLARITY_RISING);
```

## ctc\_refsource\_signal\_select

The description of ctc\_refsource\_signal\_select is shown as below:

**Table 3-82. Function ctc\_refsource\_signal\_select**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | ctc_refsource_signal_select                      |
| <b>Function prototype</b>    | void ctc_refsource_signal_select(uint32_t refs); |
| <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                          |
| CTC_REFSOURCE_GPIO           | GPIO is selected                                 |

|                              |                   |
|------------------------------|-------------------|
| <i>CTC_REFSOURCE_LXTAL</i>   | LXTAL is selected |
| <b>Output parameter{out}</b> |                   |
| -                            | -                 |
| <b>Return value</b>          |                   |
| -                            | -                 |

Example:

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

```
ctc_refsource_signal_select (CTC_REFSOURCE_LXTAL);
```

## ctc\_refsource\_prescaler\_config

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

**Table 3-83. Function ctc\_refsource\_prescaler\_config**

|                                 |  |
|---------------------------------|--|
| <b>Function name</b>            | ctc_refsource_prescaler_config                           |
| <b>Function prototype</b>       | void ctc_refsource_prescaler_config(uint32_t prescaler); |
| <b>Function descriptions</b>    | configure reference signal source prescaler              |
| <b>Precondition</b>             | -  |
| <b>The called functions</b>     | -  |
| <b>Input parameter{in}</b>      |  |
| <b>prescaler</b>                | Prescaler factor   |
| <i>CTC_REFSOURCE_PSC_OFF</i>    | reference signal not divided                             |
| <i>CTC_REFSOURCE_PSC_DIV2</i>   | reference signal divided by 2                            |
| <i>CTC_REFSOURCE_PSC_DIV4</i>   | reference signal divided by 4                            |
| <i>CTC_REFSOURCE_PSC_DIV8</i>   | reference signal divided by 8                            |
| <i>CTC_REFSOURCE_PSC_DIV16</i>  | reference signal divided by 16                           |
| <i>CTC_REFSOURCE_PSC_DIV32</i>  | reference signal divided by 32                           |
| <i>CTC_REFSOURCE_PSC_DIV64</i>  | reference signal divided by 64                           |
| <i>CTC_REFSOURCE_PSC_DIV128</i> | reference signal divided by 128                          |
| <b>Output parameter{out}</b>    |  |
| -                               | -  |
| <b>Return value</b>             |  |
| -                               | -  |

Example:

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

ctc_refsource_prescaler_config(CTC_REFSOURCE_PSC_DIV2);
```

## ctc\_clock\_limit\_value\_config

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

**Table 3-84. Function ctc\_clock\_limit\_value\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | ctc_clock_limit_value_config                            |
| <b>Function prototype</b>    | void ctc_clock_limit_value_config(uint8_t limit_value); |
| <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 3-85. 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\_capture\_value\_read

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

**Table 3-86. 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>          |  |
| uint16_t                     | 读取计数器捕获值(0x0000 - 0xFFFF)                      |

Example:

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

```
uint16_t ctc_value = 0;
```

```
ctc_value = ctc_counter_capture_value_read ();
```

## ctc\_counter\_direction\_read

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

**Table 3-87. 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> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| FlagStatus                   | SET(向下计数) / RESET(向上计数)                      |

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 3-88. Function ctc\_counter\_reload\_value\_read**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | ctc_counter_reload_value_read                 |
| <b>Function prototype</b>    | uint16_t ctc_counter_reload_value_read(void); |
| <b>Function descriptions</b> | read CTC counter reload value                 |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| uint16_t                     | 读取计数器重载值的16位数据 (0x0000 - 0xFFFF)              |

Example:

```
/* read CTC counter reload value */
```

```
uint16_t ctc_reload_value = 0;
```

```
ctc_reload_value = ctc_counter_reload_value_read ();
```

## ctc\_irc48m\_trim\_value\_read

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

**Table 3-89. Function ctc\_irc48m\_trim\_value\_read**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | ctc_irc48m_trim_value_read                |
| <b>Function prototype</b>    | uint8_t ctc_irc48m_trim_value_read(void); |
| <b>Function descriptions</b> | read the IRC48M trim value                |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| uint8_t                      | 6位IRC48M校准值 (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 3-90. Function ctc\_interrupt\_enable**

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

Example:

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

ctc_interrupt_enable (CTC_INT_CKOK);
```

## ctc\_interrupt\_disable

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

**Table 3-91. 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                                   |
| <i>CTC_INT_CKOK</i>          | clock trim OK interrupt                         |

|                              |                              |
|------------------------------|------------------------------|
| <i>CTC_INT_CKWARN</i>        | clock trim warning interrupt |
| <i>CTC_INT_ERR</i>           | error interrupt              |
| <i>CTC_INT_EREf</i>          | expect reference interrupt   |
| <b>Output parameter{out}</b> |                              |
| -                            | -                            |
| <b>Return value</b>          |                              |
| -                            | -                            |

Example:

```
/* disable CTC clock trim OK interrupt */
ctc_interrupt_disable (CTC_INT_CKOK);
```

## ctc\_interrupt\_flag\_get

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

**Table 3-92. Function ctc\_interrupt\_flag\_get**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | ctc_interrupt_flag_get                                 |
| <b>Function prototype</b>    | FlagStatus ctc_interrupt_flag_get(uint32_t interrupt); |
| <b>Function descriptions</b> | get CTC interrupt flag                                 |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>interrupt</b>             | CTC interrupt flag                                     |
| <i>CTC_INT_FLAG_CKOK</i>     | clock trim OK interrupt                                |
| <i>CTC_INT_FLAG_CKWARN</i>   | clock trim warning interrupt                           |
| <i>CTC_INT_FLAG_ERR</i>      | error interrupt  |
| <i>CTC_INT_FLAG_EREf</i>     | expect reference interrupt                             |
| <i>CTC_INT_FLAG_CKErr</i>    | clock trim error bit interrupt                         |
| <i>CTC_INT_FLAG_REFMISS</i>  | reference sync pulse miss interrupt                    |
| <i>CTC_INT_FLAG_TRIMERR</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 3-93. Function ctc\_interrupt\_flag\_clear**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | ctc_interrupt_flag_clear                           |
| <b>Function prototype</b>    | void ctc_interrupt_flag_clear(uint32_t interrupt); |
| <b>Function descriptions</b> | clear CTC interrupt flag                           |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>interrupt</b>             | CTC interrupt flag                                 |
| CTC_INT_FLAG_CKOK            | clock trim OK interrupt                            |
| CTC_INT_FLAG_CKWARN          | clock trim warning interrupt                       |
| CTC_INT_FLAG_ERR             | error interrupt                                    |
| CTC_INT_FLAG_EREFS           | expect reference interrupt                         |
| CTC_INT_FLAG_CKEERR          | clock trim error bit interrupt                     |
| CTC_INT_FLAG_REFMISS         | reference sync pulse miss interrupt                |
| CTC_INT_FLAG_TRIMERR         | trim value error interrupt                         |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

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

```
ctc_interrupt_flag_clear (CTC_INT_FLAG_CKOK);
```

## ctc\_flag\_get

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

**Table 3-94. 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                   |
| <i>CTC_FLAG_CKOK</i>         | clock trim OK interrupt flag      |
| <i>CTC_FLAG_CKWARN</i>       | clock trim warning interrupt flag |
| <i>CTC_FLAG_ERR</i>          | error interrupt flag              |
| <i>CTC_FLAG_EREFP</i>        | expect reference interrupt flag   |
| <i>CTC_FLAG_CKERR</i>        | clock trim error bit              |
| <i>CTC_FLAG_REFMIS</i>       | reference sync pulse miss flag    |
| <i>CTC_FLAG_TRIMERR</i>      | trim value error flag             |
| <b>Output parameter{out}</b> |                                   |
| -                            | -                                 |
| <b>Return value</b>          |                                   |
| <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 3-95. 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                      |
| <i>CTC_FLAG_CKOK</i>         | clock trim OK interrupt flag         |
| <i>CTC_FLAG_CKWARN</i>       | clock trim warning interrupt flag    |
| <i>CTC_FLAG_ERR</i>          | error interrupt flag                 |
| <i>CTC_FLAG_EREFP</i>        | expect reference interrupt flag      |
| <i>CTC_FLAG_CKERR</i>        | clock trim error bit                 |
| <i>CTC_FLAG_REFMIS</i>       | reference sync pulse miss flag       |
| <i>CTC_FLAG_TRIMERR</i>      | trim value error flag                |
| <b>Output parameter{out}</b> |                                      |
| -                            | -                                    |
| <b>Return value</b>          |                                      |
| -                            | -                                    |

Example:

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

```
ctc_flag_clear (CTC_FLAG_CKOK);
```

## 3.6. DAC

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

### 3.6.1. Descriptions of Peripheral registers

DAC registers are listed in the table shown as below:

**Table 3-96. DAC Registers**

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

### 3.6.2. Descriptions of Peripheral functions

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

**Table 3-97. DAC firmware function**

| Function name             | Function description      |
|---------------------------|---------------------------|
| dac_deinit                | deinitialize DAC          |
| dac_enable                | enable DAC                |
| dac_disable               | disable DAC               |
| dac_dma_enable            | enable DAC DMA function   |
| dac_dma_disable           | disable DAC DMA function  |
| dac_output_buffer_enable  | enable DAC output buffer  |
| dac_output_buffer_disable | disable DAC output buffer |
| dac_output_value_get      | get DAC output value      |

| Function name                                       | Function description                                |
|---|---|
| <code>dac_data_set</code>                           | set DAC data holding register value                 |
| <code>dac_trigger_enable</code>                     | enable DAC trigger                                  |
| <code>dac_trigger_disable</code>                    | disable DAC trigger                                 |
| <code>dac_trigger_source_config</code>              | configure DAC trigger source                        |
| <code>dac_software_trigger_enable</code>            | enable DAC software trigger                         |
| <code>dac_wave_mode_config</code>                   | configure DAC wave mode                             |
| <code>dac_lfsr_noise_config</code>                  | configure DAC LFSR noise mode                       |
| <code>dac_triangle_noise_config</code>              | configure DAC triangle noise mode                   |
| <code>dac_concurrent_enable</code>                  | enable DAC concurrent mode                          |
| <code>dac_concurrent_disable</code>                 | disable DAC concurrent mode                         |
| <code>dac_concurrent_software_trigger_enable</code> | enable DAC concurrent software trigger              |
| <code>dac_concurrent_output_buffer_enable</code>    | enable DAC concurrent buffer function               |
| <code>dac_concurrent_output_buffer_disable</code>   | disable DAC concurrent buffer function              |
| <code>dac_concurrent_data_set</code>                | set DAC concurrent mode data holding register value |

## **dac\_deinit**

The description of `dac_deinit` is shown as below:

**Table 3-98. Function `dac_deinit`**

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

Example:

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

## **dac\_enable**

The description of `dac_enable` is shown as below:

**Table 3-99. Function `dac_enable`**

|                      |                         |
|----------------------|-------------------------|
| <b>Function name</b> | <code>dac_enable</code> |
|----------------------|-------------------------|

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

Example:

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

## **dac\_disable**

The description of dac\_disable is shown as below:

**Table 3-100. Function dac\_disable**

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

Example:

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



## **dac\_dma\_enable**

The description of `dac_dma_enable` is shown as below:

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

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

Example:

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

## **dac\_dma\_disable**

The description of `dac_dma_disable` is shown as below:

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

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

Example:

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

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

The description of `dac_output_buffer_enable` is shown as below:

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

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

Example:

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

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

The description of `dac_output_buffer_disable` is shown as below:

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

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

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

Example:

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

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

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

**Table 3-105. Function dac\_output\_value\_get**

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

Example:

```
/* get the DAC0_OUT0 last data output value */
uint16_t data=0;
data = dac_output_value_get(DAC0, DAC_OUT0);
```

## **dac\_data\_set**

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

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

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

Example:

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

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

## **`dac_trigger_enable`**

The description of `dac_trigger_enable` is shown as below:

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

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

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

Example:

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

## **dac\_trigger\_disable**

The description of `dac_trigger_disable` is shown as below:

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

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

Example:

```
/* disable DAC0_OUT0 trigger */
dac_trigger_disable(DAC0, DAC_OUT0);
```

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

The description of `dac_trigger_source_config` is shown as below:

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

|                           |  |
|---------------------------|--|
| <b>Function name</b>      | <code>dac_trigger_source_config</code>   |
| <b>Function prototype</b> | <code>void dac_trigger_source_config(uint32_t dac_periph, uint8_t dac_out, uint32_t triggersource);</code> |

|                                |  |
|--------------------------------|--|
| <b>Function descriptions</b>   | configure DAC trigger source           |
| <b>Precondition</b>            | -                                      |
| <b>The called functions</b>    | -                                      |
| <b>Input parameter{in}</b>     |  |
| <b>dac_periph</b>              | DAC peripheral                         |
| <i>DACx</i>                    | DAC peripheral selection (x = 0)       |
| <b>Input parameter{in}</b>     |  |
| <b>dac_out</b>                 | DAC output                             |
| <i>DAC_OUTx</i>                | DAC output channel selection (x = 0,1) |
| <b>Input parameter{in}</b>     |  |
| <b>triggersource</b>           | external trigger of DAC                |
| <i>DAC_TRIGGER_T5_TRG</i><br>0 | TIMER5 TRGO                            |
| <i>DAC_TRIGGER_T2_TRG</i><br>0 | TIMER2 TRGO                            |
| <i>DAC_TRIGGER_T6_TRG</i><br>0 | TIMER6 TRGO                            |
| <i>DAC_TRIGGER_T4_TRG</i><br>0 | TIMER4 TRGO                            |
| <i>DAC_TRIGGER_T1_TRG</i><br>0 | TIMER1 TRGO                            |
| <i>DAC_TRIGGER_T3_TRG</i><br>0 | TIMER3 TRGO                            |
| <i>DAC_TRIGGER_EXTI_9</i>      | EXTI interrupt line9 event             |
| <i>DAC_TRIGGER_SOFTWARE</i>    | software trigger                       |
| <b>Output parameter{out}</b>   |  |
| -                              | -                                      |
| <b>Return value</b>            |  |
| -                              | -                                      |

Example:

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

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

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

The description of `dac_software_trigger_enable` is shown as below:

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

|                           |  |
|---------------------------|--|
| <b>Function name</b>      | <code>dac_software_trigger_enable</code>   |
| <b>Function prototype</b> | <code>void dac_software_trigger_enable(uint32_t dac_periph, uint8_t dac_out);</code> |

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

Example:

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

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

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

**Table 3-111. Function dac\_wave\_mode\_config**

|                               |  |
|-------------------------------|--|
| <b>Function name</b>          | dac_wave_mode_config   |
| <b>Function prototype</b>     | void dac_wave_mode_config(uint32_t dac_periph, uint8_t dac_out, 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>dac_periph</b>             | DAC peripheral   |
| <i>DACx</i>                   | DAC peripheral selection (x = 0)   |
| <b>Input parameter{in}</b>    |  |
| <b>dac_out</b>                | DAC output   |
| <i>DAC_OUTx</i>               | DAC output channel selection (x = 0,1)   |
| <b>Input parameter{in}</b>    |  |
| <b>wave_mode</b>              | DAC wave mode  |
| <i>DAC_WAVE_DISABLE</i>       | wave mode disable  |
| <i>DAC_WAVE_MODE_LFSR</i>     | LFSR noise mode  |
| <i>DAC_WAVE_MODE_TRIANGLE</i> | triangle noise mode  |
| <b>Output parameter{out}</b>  |  |
| -                             | -  |

| Return value |   |
|--------------|---|
| -            | - |

Example:

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

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

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

The description of `dac_lfsr_noise_config` is shown as below:

**Table 3-112. Function `dac_lfsr_noise_config`**

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

Example:

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

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

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

The description of `dac_triangle_noise_config` is shown as below:

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

| Function name      | <code>dac_triangle_noise_config</code>  |
|--------------------|---|
| Function prototype | <code>void dac_triangle_noise_config(uint32_t dac_periph, uint8_t dac_out,</code> |



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

Example:

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

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

## **dac\_concurrent\_enable**

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

**Table 3-114. Function dac\_concurrent\_enable**

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

Example:

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

`dac_concurrent_enable(DAC0);`

## **dac\_concurrent\_disable**

The description of `dac_concurrent_disable` is shown as below:

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

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

Example:

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

```
dac_concurrent_disable(DAC0);
```

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

The description of `dac_concurrent_software_trigger_enable` is shown as below:

**Table 3-116. Function `dac_concurrent_software_trigger_enable`**

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

Example:

```
/* enable DAC0 concurrent software trigger */
```

`dac_concurrent_software_trigger_enable(DAC0);`

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

The description of `dac_concurrent_output_buffer_enable` is shown as below:

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

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

Example:

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

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

The description of `dac_concurrent_output_buffer_disable` is shown as below:

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

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

Example:

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

`dac_concurrent_output_buffer_disable(DAC0);`

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

The description of `dac_concurrent_data_set` is shown as below:

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

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

Example:

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

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

## **3.7. DBG**

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

### **3.7.1. Descriptions of Peripheral registers**

DBG registers are listed in the table shown as below:

**Table 3-120. DBG Registers**

| Registers | Descriptions         |
|-----------|----------------------|
| DBG_ID    | DBG ID code register |
| DBG_CTL   | DBG control register |

## 3.7.2. Descriptions of Peripheral functions

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

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

| Function name         | Function description                                      |
|-----------------------|---|
| 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 |
| dbg_trace_pin_enable  | enable trace pin assignment                               |
| dbg_trace_pin_disable | disable trace pin assignment                              |

## Enum dbg\_periph\_enum

**Table 3-122. 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_TIMER3_HOLD  | hold TIMER3 counter when core is halted  |
| DBG_I2C0_HOLD    | hold I2C0 smbus when core is halted      |
| DBG_I2C1_HOLD    | hold I2C1 smbus when core is halted      |
| DBG_TIMER4_HOLD  | hold TIMER4 counter when core is halted  |
| DBG_TIMER5_HOLD  | hold TIMER5 counter when core is halted  |
| DBG_TIMER6_HOLD  | hold TIMER6 counter when core is halted  |
| DBG_TIMER7_HOLD  | hold TIMER7 counter when core is halted  |
| DBG_TIMER11_HOLD | hold TIMER11 counter when core is halted |
| DBG_TIMER12_HOLD | hold TIMER12 counter when core is halted |
| DBG_TIMER13_HOLD | hold TIMER13 counter when core is halted |
| DBG_TIMER8_HOLD  | hold TIMER8 counter when core is halted  |
| DBG_TIMER9_HOLD  | hold TIMER9 counter when core is halted  |
| DBG_TIMER10_HOLD | hold TIMER10 counter when core is halted |

## dbg\_id\_get

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

**Table 3-123. 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 3-124. 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>   |   |
| dbg_low_power                | low power mode  |
| DBG_LOW_POWER_SLEEP          | keep debugger connection during sleep mode              |
| DBG_LOW_POWER_DEEPSLEEP      | keep debugger connection during deepsleep mode          |
| DBG_LOW_POWER_STANDBY        | keep debugger connection during standby mode            |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |

Example:

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

```
dbg_low_power_enable(DBG_LOW_POWER_SLEEP);
```

## dbg\_low\_power\_disable

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

**Table 3-125. Function dbg\_low\_power\_disable**

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

Example:

```
/* 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 3-126. 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>   |  |

| dbg_periph            | Peripheral refer to <a href="#">Table 3-122. Enum dbg_periph_enum</a>      |
|-----------------------|--|
| DBG_FWDGT_HOLD        | debug FWDGT kept when core is halted                                       |
| DBG_WWDGT_HOLD        | debug WWDGT kept when core is halted                                       |
| DBG_I2Cx_HOLD         | x=0,1, hold I2Cx smbus when core is halted                                 |
| DBG_TIMERx_HOLD       | x=0,1,2,3,4,5,6,7,8,9,10,11,12,13, hold TIMERx counter when core is halted |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

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 3-127. Function dbg\_periph\_disable**

| Function name         | dbg_periph_disable   |
|-----------------------|--|
| Function prototype    | void dbg_periph_disable(dbg_periph_enum dbg_periph);                       |
| Function descriptions | Disable peripheral behavior when the mcu is in debug mode                  |
| Precondition          | -  |
| The called functions  | -  |
| Input parameter{in}   |  |
| dbg_periph            | peripheral refer to <a href="#">Table 3-122. Enum dbg_periph_enum</a>      |
| DBG_FWDGT_HOLD        | debug FWDGT kept when core is halted                                       |
| DBG_WWDGT_HOLD        | debug WWDGT kept when core is halted                                       |
| DBG_I2Cx_HOLD         | x=0,1, hold I2Cx smbus when core is halted                                 |
| DBG_TIMERx_HOLD       | x=0,1,2,3,4,5,6,7,8,9,10,11,12,13, hold TIMERx counter when core is halted |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

Example:

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

```
dbg_periph_disable(DBG_TIMER0_HOLD);
```

## dbg\_trace\_pin\_enable

The description of dbg\_trace\_pin\_enable is shown as below:



**Table 3-128. Function dbg\_trace\_pin\_enable**

|                              |                                  |
|------------------------------|----------------------------------|
| <b>Function name</b>         | dbg_trace_pin_enable             |
| <b>Function prototype</b>    | void dbg_trace_pin_enable(void); |
| <b>Function descriptions</b> | Enable trace pin assignment      |
| <b>Precondition</b>          | -                                |
| <b>The called functions</b>  | -                                |
| <b>Input parameter{in}</b>   |                                  |
| -                            | -                                |
| <b>Output parameter{out}</b> |                                  |
| -                            | -                                |
| <b>Return value</b>          |                                  |
| -                            | -                                |

Example:

```
/* enable trace pin assignment */
dbg_trace_pin_enable();
```

## dbg\_trace\_pin\_disable

The description of dbg\_trace\_pin\_disable is shown as below:

**Table 3-129. Function dbg\_trace\_pin\_disable**

|                              |                                   |
|------------------------------|-----------------------------------|
| <b>Function name</b>         | dbg_trace_pin_disable             |
| <b>Function prototype</b>    | void dbg_trace_pin_disable(void); |
| <b>Function descriptions</b> | Disable trace pin assignment      |
| <b>Precondition</b>          | -                                 |
| <b>The called functions</b>  | -                                 |
| <b>Input parameter{in}</b>   |                                   |
| -                            | -                                 |
| <b>Output parameter{out}</b> |                                   |
| -                            | -                                 |
| <b>Return value</b>          |                                   |
| -                            | -                                 |

Example:

```
/* disable trace pin assignment */
dbg_trace_pin_disable();
```

## 3.8. DMA

The direct memory access (DMA) controller provides a hardware method of transferring data between peripherals and/or memory without intervention from the CPU, thereby freeing up

bandwidth for other system functions. The DMA registers are listed in chapter [3.8.1](#), the DMA firmware functions are introduced in chapter [3.8.2](#).

## 3.8.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

**Table 3-130. DMA Registers**

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

## 3.8.2. Descriptions of Peripheral functions

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

**Table 3-131. 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                      |

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

## Structure `dma_parameter_struct`

**Table 3-132. Structure `dma_parameter_struct`**

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

## `dma_deinit`

The description of `dma_deinit` is shown as below:

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

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

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

Example:

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

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

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

**Table 3-134. Function dma\_struct\_para\_init**

| Function name         | dma_struct_para_init  |
|-----------------------|---|
| Function prototype    | void dma_struct_para_init(dma_parameter_struct* init_struct);   |
| Function descriptions | initialize the parameters of DMA struct with the default values |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |
| *init_struct          | a spi_parameter_struct address                                  |
| Output parameter{out} |   |
| -                     | -   |
| Return value          |   |
| -                     | -   |

Example:

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

## **dma\_init**

The description of dma\_init is shown as below:

**Table 3-135. Function dma\_init**

| Function name         | dma_init  |
|-----------------------|---|
| Function prototype    | void dma_init(uint32_t dma_periph, dma_channel_enum channelx, dma_parameter_struct* init_struct); |
| Function descriptions | initialize DMA channel  |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |
| dma_periph            | DMA peripheral  |
| DMAx(x=0,1)           | DMA peripheral selection  |

| Input parameter{in}                       |  |
|---|--|
| <b>channelx</b>                           | DMA channel  |
| <i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i> | DMA channel selection  |
| Input parameter{in}                       |  |
| <b>init_struct</b>                        | Structure for initialization, the structure members can refer to <a href="#">Table 3-132. Structure dma_parameter_struct</a> |
| Output parameter{out}                     |  |
| -   | -  |
| Return value                              |  |
| -   | -  |

Example:

```

/* DMA0 channel0 initialize */
dma_parameter_struct dma_init_struct;
dma_deinit(DMA0, DMA_CH0);
dma_struct_para_init(&dma_init_struct);

dma_init_struct.direction = DMA_PERIPHERAL_TO_MEMORY;
dma_init_struct.memory_addr = (uint32_t)g_destbuf;
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_struct.memory_width = DMA_MEMORY_WIDTH_8BIT;
dma_init_struct.number = TRANSFER_NUM;
dma_init_struct.periph_addr = (uint32_t)BANK0_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_8BIT;
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_init(DMA0, DMA_CH0, &dma_init_struct);

```

## **dma\_circulation\_enable**

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

**Table 3-136. Function dma\_circulation\_enable**

| <b>Function name</b>         | dma_circulation_enable   |
|------------------------------|--|
| <b>Function prototype</b>    | void dma_circulation_enable(uint32_t dma_periph, dma_channel_enum channelx); |
| <b>Function descriptions</b> | enable DMA circulation mode  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| Input parameter{in}          |  |
| <b>dma_periph</b>            | DMA peripheral   |
| <i>DMAx(x=0, 1)</i>          | DMA peripheral selection   |
| Input parameter{in}          |  |

|   |                       |
|---|-----------------------|
| <b>channelx</b>                           | DMA channel           |
| <i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i> | DMA channel selection |
| <b>Output parameter{out}</b>              |                       |
| -   | -                     |
| <b>Return value</b>                       |                       |
| -   | -                     |

Example:

```
/* enable DMA0 channel0 circulation mode */
dma_circulation_enable(DMA0, DMA_CH0);
```

## **dma\_circulation\_disable**

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

**Table 3-137. Function dma\_circulation\_disable**

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

Example:

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

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

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

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

|                      |                             |
|----------------------|-----------------------------|
| <b>Function name</b> | dma_memory_to_memory_enable |
|----------------------|-----------------------------|

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

Example:

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

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

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

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

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

Example:

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

## **dma\_channel\_enable**

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

**Table 3-140. Function dma\_channel\_enable**

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

Example:

```
/* enable DMA0 channel0 */
dma_channel_enable(DMA0, DMA_CH0);
```

## **dma\_channel\_disable**

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

**Table 3-141. Function dma\_channel\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | dma_channel_disable   |
| <b>Function prototype</b>    | void dma_channel_disable(uint32_t dma_periph, dma_channel_enum channelx); |
| <b>Function descriptions</b> | disable DMA channel   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>dma_periph</b>            | DMA peripheral  |



|   |                          |
|---|--------------------------|
| <i>DMAx(x=0, 1)</i>                       | DMA peripheral selection |
| <b>Input parameter{in}</b>                |                          |
| <b>channelx</b>                           | DMA channel              |
| <i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i> | DMA channel selection    |
| <b>Output parameter{out}</b>              |                          |
| -   | -                        |
| <b>Return value</b>                       |                          |
| -   | -                        |

Example:

```
/* disable DMA0 channel0 */
dma_channel_disable(DMA0, DMA_CH0);
```

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

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

**Table 3-142. Function dma\_periph\_address\_config**

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

Example:

```
#define BANK0_WRITE_START_ADDR ((uint32_t)0x08004000)

dma_periph_address_config(DMA0, DMA_CH0, BANK0_WRITE_START_ADDR);
```

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

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

**Table 3-143. Function dma\_memory\_address\_config**

|  |  |
|--|--|
| <b>Function name</b>                       | dma_memory_address_config  |
| <b>Function prototype</b>                  | void dma_memory_address_config(uint32_t dma_periph,<br>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>dma_periph</b>                          | DMA peripheral   |
| <i>DMAx(x=0, 1)</i>                        | DMA peripheral selection   |
| <b>Input parameter{in}</b>                 |  |
| <b>channelx</b>                            | DMA channel  |
| <i>DMA_CHx(DMA0: x=0..6; DMA1: x=0..4)</i> | DMA channel selection  |
| <b>Input parameter{in}</b>                 |  |
| <b>address</b>                             | memory base address  |
| <b>Output parameter{out}</b>               |  |
| -  | -  |
| <b>Return value</b>                        |  |
| -  | -  |

Example:

```
uint8_t g_destbuf[TRANSFER_NUM];
```

```
dma_memory_address_config(DMA0, DMA_CH0, (uint32_t) g_destbuf);
```

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

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

**Table 3-144. Function dma\_transfer\_number\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | dma_transfer_number_config   |
| <b>Function prototype</b>    | void dma_transfer_number_config(uint32_t dma_periph,<br>dma_channel_enum channelx, uint32_t number); |
| <b>Function descriptions</b> | set the number of remaining data to be transferred by the DMA  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>dma_periph</b>            | DMA peripheral   |
| <i>DMAx(x=0, 1)</i>          | DMA peripheral selection   |
| <b>Input parameter{in}</b>   |  |

|   |                       |
|---|-----------------------|
| <b>channelx</b>                           | DMA channel           |
| <i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i> | DMA channel selection |
| <b>Input parameter{in}</b>                |                       |
| <b>number</b>                             | data transfer number  |
| <i>0-0xffff</i>                           | number                |
| <b>Output parameter{out}</b>              |                       |
| -   | -                     |
| <b>Return value</b>                       |                       |
| -   | -                     |

Example:

```
#define TRANSFER_NUM                0x400

dma_transfer_number_config(DMA0, DMA_CH0, TRANSFER_NUM);
```

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

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

**Table 3-145. Function dma\_transfer\_number\_get**

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

Example:

```
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 3-146. Function dma\_priority\_config**

|   |  |
|---|--|
| <b>Function name</b>                      | dma_priority_config  |
| <b>Function prototype</b>                 | void dma_priority_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t priority); |
| <b>Function descriptions</b>              | configure priority level of DMA channel  |
| <b>Precondition</b>                       | -  |
| <b>The called functions</b>               | -  |
| <b>Input parameter{in}</b>                |  |
| <b>dma_periph</b>                         | DMA peripheral   |
| <i>DMAx(x=0,1)</i>                        | DMA peripheral selection   |
| <b>Input parameter{in}</b>                |  |
| <b>channelx</b>                           | DMA channel  |
| <i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i> | DMA channel selection  |
| <b>Input parameter{in}</b>                |  |
| <b>priority</b>                           | priority Level of this channel   |
| <i>DMA_PRIORITY_LOW</i>                   | low priority   |
| <i>DMA_PRIORITY_MEDIUM</i>                | medium priority  |
| <i>DMA_PRIORITY_HIGH</i>                  | high priority  |
| <i>DMA_PRIORITY_ULTRA_HIGH</i>            | ultra high priority  |
| <b>Output parameter{out}</b>              |  |
| -   | -  |
| <b>Return value</b>                       |  |
| -   | -  |

Example:

```
dma_priority_config(DMA0, DMA_CH0, DMA_PRIORITY_ULTRA_HIGH);
```

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

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

**Table 3-147. Function dma\_memory\_width\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | dma_memory_width_config   |
| <b>Function prototype</b>    | void dma_memory_width_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t mwidth); |
| <b>Function descriptions</b> | configure transfer data size of memory  |
| <b>Precondition</b>          | -   |

|   |   |
|---|---|
| <b>The called functions</b>               | -                                       |
| <b>Input parameter{in}</b>                |   |
| <b>dma_periph</b>                         | DMA peripheral                          |
| <i>DMAx(x=0, 1)</i>                       | DMA peripheral selection                |
| <b>Input parameter{in}</b>                |   |
| <b>channelx</b>                           | DMA channel                             |
| <i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i> | DMA channel selection                   |
| <b>Input parameter{in}</b>                |   |
| <b>mwidth</b>                             | transfer data width of memory           |
| <i>DMA_MEMORY_WIDTH_8BIT</i>              | transfer data width of memory is 8-bit  |
| <i>DMA_MEMORY_WIDTH_16BIT</i>             | transfer data width of memory is 16-bit |
| <i>DMA_MEMORY_WIDTH_32BIT</i>             | transfer data width of memory is 32-bit |
| <b>Output parameter{out}</b>              |   |
| -   | -                                       |
| <b>Return value</b>                       |   |
| -   | -                                       |

Example:

```
dma_memory_width_config(DMA0, DMA_CH0, DMA_MEMORY_WIDTH_8BIT);
```

## **dma\_periph\_width\_config**

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

**Table 3-148. Function dma\_periph\_width\_config**

|   |   |
|---|---|
| <b>Function name</b>                      | dma_periph_width_config   |
| <b>Function prototype</b>                 | void dma_periph_width_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t pwidth); |
| <b>Function descriptions</b>              | configure transfer data size of peripheral  |
| <b>Precondition</b>                       | -   |
| <b>The called functions</b>               | -   |
| <b>Input parameter{in}</b>                |   |
| <b>dma_periph</b>                         | DMA peripheral  |
| <i>DMAx(x=0, 1)</i>                       | DMA peripheral selection  |
| <b>Input parameter{in}</b>                |   |
| <b>channelx</b>                           | DMA channel   |
| <i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i> | DMA channel selection   |
| <b>Input parameter{in}</b>                |   |
| <b>pwidth</b>                             | transfer data width of peripheral   |

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

Example:

```
dma_periph_width_config(DMA0, DMA_CH0, DMA_PERIPHERAL_WIDTH_8BIT);
```

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

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

**Table 3-149. Function dma\_memory\_increase\_enable**

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

Example:

```
dma_memory_increase_enable(DMA0, DMA_CH0);
```

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

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

**Table 3-150. Function dma\_memory\_increase\_disable**

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

Example:

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

## dma\_periph\_increase\_enable

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

**Table 3-151. Function dma\_periph\_increase\_enable**

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

Example:

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

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

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

**Table 3-152. Function dma\_periph\_increase\_disable**

|   |  |
|---|--|
| <b>Function name</b>                      | dma_periph_increase_disable  |
| <b>Function prototype</b>                 | void dma_periph_increase_disable(uint32_t dma_periph,<br>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>dma_periph</b>                         | DMA peripheral   |
| <i>DMAx(x=0, 1)</i>                       | DMA peripheral selection   |
| <b>Input parameter{in}</b>                |  |
| <b>channelx</b>                           | DMA channel  |
| <i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i> | DMA channel selection  |
| <b>Output parameter{out}</b>              |  |
| -   | -  |
| <b>Return value</b>                       |  |
| -   | -  |

Example:

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

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

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

**Table 3-153. Function dma\_transfer\_direction\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | dma_transfer_direction_config  |
| <b>Function prototype</b>    | void dma_transfer_direction_config(uint32_t dma_periph,<br>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>dma_periph</b>            | DMA peripheral   |
| <i>DMAx(x=0, 1)</i>          | DMA peripheral selection   |
| <b>Input parameter{in}</b>   |  |



|   |  |
|---|--|
| <b>channelx</b>                           | DMA channel                              |
| <i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i> | DMA channel selection                    |
| <b>Input parameter{in}</b>                |  |
| <b>direction</b>                          | specify the direction of data transfer   |
| <i>DMA_PERIPHERAL_TO_MEMORY</i>           | read from peripheral and write to memory |
| <i>DMA_MEMORY_TO_PERIPHERAL</i>           | read from memory and write to peripheral |
| <b>Output parameter{out}</b>              |  |
| -   | -  |
| <b>Return value</b>                       |  |
| -   | -  |

Example:

```
dma_transfer_direction_config(DMA0, DMA_CH0, DMA_PERIPHERAL_TO_MEMORY);
```

## dma\_flag\_get

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

**Table 3-154. Function dma\_flag\_get**

|   |   |
|---|---|
| <b>Function name</b>                      | dma_flag_get  |
| <b>Function prototype</b>                 | FlagStatus dma_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag); |
| <b>Function descriptions</b>              | check DMA flag is set or not  |
| <b>Precondition</b>                       | -   |
| <b>The called functions</b>               | -   |
| <b>Input parameter{in}</b>                |   |
| <b>dma_periph</b>                         | DMA peripheral  |
| <i>DMAx(x=0, 1)</i>                       | DMA peripheral selection  |
| <b>Input parameter{in}</b>                |   |
| <b>channelx</b>                           | DMA channel   |
| <i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i> | DMA channel selection   |
| <b>Input parameter{in}</b>                |   |
| <b>flag</b>                               | specify get which flag  |
| <i>DMA_FLAG_G</i>                         | global interrupt flag of channel  |
| <i>DMA_FLAG_FTF</i>                       | full transfer finish flag of channel  |
| <i>DMA_FLAG_HTF</i>                       | half transfer finish flag of channel  |
| <i>DMA_FLAG_ERR</i>                       | error flag of channel   |
| <b>Output parameter{out}</b>              |   |
| -   | -   |
| <b>Return value</b>                       |   |

|                   |              |
|-------------------|--------------|
| <b>FlagStatus</b> | SET or RESET |
|-------------------|--------------|

Example:

```
FlagStatus flag = RESET;
```

```
flag = dma_flag_get(DMA0, DMA_CH0, DMA_FLAG_FTF);
```

## **dma\_flag\_clear**

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

**Table 3-155. Function dma\_flag\_clear**

|   |   |
|---|---|
| <b>Function name</b>                      | dma_flag_clear  |
| <b>Function prototype</b>                 | void dma_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag); |
| <b>Function descriptions</b>              | clear the flag of a DMA channel   |
| <b>Precondition</b>                       | -   |
| <b>The called functions</b>               | -   |
| <b>Input parameter{in}</b>                |   |
| <b>dma_periph</b>                         | DMA peripheral  |
| <i>DMAx(x=0, 1)</i>                       | DMA peripheral selection  |
| <b>Input parameter{in}</b>                |   |
| <b>channelx</b>                           | DMA channel   |
| <i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i> | DMA channel selection   |
| <b>Input parameter{in}</b>                |   |
| <b>flag</b>                               | specify get which flag  |
| <i>DMA_FLAG_G</i>                         | global interrupt flag of channel  |
| <i>DMA_FLAG_FTF</i>                       | full transfer finish flag of channel  |
| <i>DMA_FLAG_HTF</i>                       | half transfer finish flag of channel  |
| <i>DMA_FLAG_ERR</i>                       | error flag of channel   |
| <b>Output parameter{out}</b>              |   |
| -   | -   |
| <b>Return value</b>                       |   |
| -   | -   |

Example:

```
dma_flag_clear(DMA0, DMA_CH0, DMA_FLAG_FTF);
```

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

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

**Table 3-156. Function dma\_interrupt\_flag\_get**

|                      |                        |
|----------------------|------------------------|
| <b>Function name</b> | dma_interrupt_flag_get |
|----------------------|------------------------|

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

Example:

```
if(dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_G);
}
```

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

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

**Table 3-157. Function dma\_interrupt\_flag\_clear**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | dma_interrupt_flag_clear  |
| <b>Function prototype</b>    | void dma_interrupt_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag); |
| <b>Function descriptions</b> | clear the interrupt flag of a DMA channel   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>dma_periph</b>            | DMA peripheral  |
| <i>DMAx(x=0, 1)</i>          | DMA peripheral selection  |

| Input parameter{in}                       |  |
|---|--|
| <b>channelx</b>                           | DMA channel                                    |
| <i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i> | DMA channel selection                          |
| Input parameter{in}                       |  |
| <b>flag</b>                               | specify get which flag                         |
| <i>DMA_INT_FLAG_G</i>                     | global interrupt flag of channel               |
| <i>DMA_INT_FLAG_FTF</i>                   | full transfer finish interrupt flag of channel |
| <i>DMA_INT_FLAG_HTF</i>                   | half transfer finish interrupt flag of channel |
| <i>DMA_INT_FLAG_ER</i>                    | error interrupt flag of channel                |
| Output parameter{out}                     |  |
| -   | -  |
| Return value                              |  |
| -   | -  |

Example:

```
if(dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_G);
}
```

## **dma\_interrupt\_enable**

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

**Table 3-158. Function dma\_interrupt\_enable**

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

|                              |   |
|------------------------------|---|
| <i>DMA_INT_HTF</i>           | half transfer finish interrupt of channel |
| <i>DMA_INT_ERR</i>           | error interrupt of channel                |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* DMA0 channel0 interrupt configuration */
dma_interrupt_enable(DMA0, DMA_CH0, DMA_INT_FTF);
```

## **dma\_interrupt\_disable**

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

**Table 3-159. Function dma\_interrupt\_disable**

|  |  |
|--|--|
| <b>Function name</b>                       | dma_interrupt_disable  |
| <b>Function prototype</b>                  | void dma_interrupt_disable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source); |
| <b>Function descriptions</b>               | disable DMA interrupt  |
| <b>Precondition</b>                        | -  |
| <b>The called functions</b>                | -  |
| <b>Input parameter{in}</b>                 |  |
| <b>dma_periph</b>                          | DMA peripheral   |
| <i>DMAx(x=0, 1)</i>                        | DMA peripheral selection   |
| <b>Input parameter{in}</b>                 |  |
| <b>channelx</b>                            | DMA channel  |
| <i>DMA_CHx(DMA0: x=0..6; DMA1: x=0..4)</i> | DMA channel selection  |
| <b>Input parameter{in}</b>                 |  |
| <b>source</b>                              | DMA interrupt source   |
| <i>DMA_INT_FTF</i>                         | full transfer finish interrupt of channel  |
| <i>DMA_INT_HTF</i>                         | half transfer finish interrupt of channel  |
| <i>DMA_INT_ERR</i>                         | error interrupt of channel   |
| <b>Output parameter{out}</b>               |  |
| -  | -  |
| <b>Return value</b>                        |  |
| -  | -  |

Example:

```
/* DMA0 channel0 interrupt configuration */
dma_interrupt_disable(DMA0, DMA_CH0, DMA_INT_FTF);
```

## 3.9. EXMC

The external memory controller EXMC, is used as a translator for MCU to access a variety of external memory. The EXMC registers are listed in chapter [3.9.1](#), the EXMC firmware functions are introduced in chapter [3.9.2](#).

### 3.9.1. Descriptions of Peripheral registers

EXMC registers are listed in the table shown as below:

**Table 3-160. EXMC Registers**

| Registers    | Descriptions  |
|--------------|---|
| EXMC_SNCTL   | SRAM/NOR Flash control registers                    |
| EXMC_SNTCFG  | SRAM/NOR Flash timing configuration registers       |
| EXMC_SNWTCFG | SRAM/NOR Flash write timing configuration registers |

### 3.9.2. Descriptions of Peripheral functions

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

**Table 3-161. EXMC firmware function**

| Function name                 | Function description                                |
|-------------------------------|---|
| exmc_norsram_deinit           | deinitialize EXMC NOR/SRAM bank                     |
| exmc_norsram_init             | initialize EXMC NOR/SRAM bank                       |
| exmc_norsram_struct_para_init | initialize the struct exmc_norsram_parameter_struct |
| exmc_norsram_enable           | enable EXMC NOR/PSRAM bank                          |
| exmc_norsram_disable          | disable EXMC NOR/PSRAM bank                         |
| exmc_norsram_page_size_config | configure CRAM page size                            |

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

**Table 3-162. Structure exmc\_norsram\_timing\_parameter\_struct**

| Member name             | Function description   |
|-------------------------|--|
| asyn_access_mode        | asynchronous access mode                                       |
| syn_data_latency        | configure the data latency                                     |
| syn_clk_division        | configure the clock divide ratio                               |
| bus_latency             | configure the bus latency                                      |
| asyn_data_setup_time    | configure the data setup time,asynchronous access mode valid   |
| asyn_address_hold_time  | configure the address hold time,asynchronous access mode valid |
| asyn_address_setup_time | configure the data setup time,asynchronous access mode valid   |

## Structure `exmc_norsram_parameter_struct`

**Table 3-163. Structure `exmc_norsram_parameter_struct`**

| Member name                    | Function description   |
|--------------------------------|--|
| <code>write_mode</code>        | the write mode, synchronous mode or asynchronous mode  |
| <code>extended_mode</code>     | enable or disable the extended mode  |
| <code>asyn_wait</code>         | enable or disable the asynchronous wait function   |
| <code>nwait_signal</code>      | enable or disable the NWAIT signal while in synchronous bust mode  |
| <code>memory_write</code>      | enable or disable the write operation  |
| <code>nwait_config</code>      | NWAIT signal configuration   |
| <code>wrap_burst_mode</code>   | enable or disable the wrap burst mode  |
| <code>nwait_polarity</code>    | specifies the polarity of NWAIT signal from memory   |
| <code>burst_mode</code>        | enable or disable the burst mode   |
| <code>databus_width</code>     | specifies the databus width of external memory   |
| <code>memory_type</code>       | specifies the type of external memory  |
| <code>address_data_mux</code>  | specifies whether the data bus and address bus are multiplexed   |
| <code>read_write_timing</code> | timing parameters for read and write if the extended mode is not used or the timing parameters for read if the extended mode is used |
| <code>write_timing</code>      | timing parameters for write when the extended mode is used   |

## `exmc_norsram_deinit`

The description of `exmc_norsram_deinit` is shown as below:

**Table 3-164. Function `exmc_norsram_deinit`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | <code>exmc_norsram_deinit</code>             |
| <b>Function prototype</b>    | <code>void exmc_norsram_deinit(void);</code> |
| <b>Function descriptions</b> | deinitialize EXMC NOR/SRAM bank              |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* deinitialize EXMC NOR/SRAM bank */
exmc_norsram_deinit();
```

## `exmc_norsram_init`

The description of `exmc_norsram_init` is shown as below:

**Table 3-165. Function exmc\_norsram\_init**

|                                 |   |
|---------------------------------|---|
| <b>Function name</b>            | exmc_norsram_init   |
| <b>Function prototype</b>       | void exmc_norsram_init(exmc_norsram_parameter_struct*<br>exmc_norsram_init_struct);   |
| <b>Function descriptions</b>    | initialize EXMC NOR/SRAM bank   |
| <b>Precondition</b>             | -   |
| <b>The called functions</b>     | -   |
| <b>Input parameter{in}</b>      |   |
| <b>exmc_norsram_init_struct</b> | Structure for initialization, the structure members can refer to <a href="#">Table 3-163. Structure exmc_norsram_parameter_struct</a> |
| <b>Output parameter{out}</b>    |   |
| -                               | -   |
| <b>Return value</b>             |   |
| -                               | -   |

Example:

```

/* initialize EXMC NOR/SRAM bank */

exmc_norsram_parameter_struct lcd_init_struct;

exmc_norsram_timing_parameter_struct lcd_timing_init_struct;

/* configure timing parameter */

lcd_timing_init_struct.asyn_access_mode = EXMC_ACCESS_MODE_A;

lcd_timing_init_struct.syn_data_latency = EXMC_DATA_LAT_2_CLK;

lcd_timing_init_struct.syn_clk_division = EXMC_SYN_CLOCK_RATIO_DISABLE;

lcd_timing_init_struct.bus_latency = 1;

lcd_timing_init_struct.asyn_data_setup_time = 5;

lcd_timing_init_struct.asyn_address_hold_time = 2;

lcd_timing_init_struct.asyn_address_setup_time = 2;

/* configure EXMC bus parameters */

lcd_init_struct.write_mode = EXMC_ASYNC_WRITE;

lcd_init_struct.extended_mode = DISABLE;

lcd_init_struct.asyn_wait = DISABLE;

lcd_init_struct.nwait_signal = DISABLE;

lcd_init_struct.memory_write = ENABLE;

lcd_init_struct.nwait_config = EXMC_NWAIT_CONFIG_BEFORE;

```



```

lcd_init_struct.wrap_burst_mode = DISABLE;

lcd_init_struct.nwait_polarity = EXMC_NWAIT_POLARITY_LOW;

lcd_init_struct.burst_mode = DISABLE;

lcd_init_struct.databus_width = EXMC_NOR_DATABUS_WIDTH_16B;

lcd_init_struct.memory_type = EXMC_MEMORY_TYPE_SRAM;

lcd_init_struct.address_data_mux = DISABLE;

lcd_init_struct.read_write_timing = &lcd_timing_init_struct;

lcd_init_struct.write_timing = &lcd_timing_init_struct;

exmc_norsram_init(&lcd_init_struct);

```

## exmc\_norsram\_struct\_para\_init

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

**Table 3-166. Function exmc\_norsram\_struct\_para\_init**

| Function name            | exmc_norsram_struct_para_init   |
|--------------------------|---|
| Function prototype       | void exmc_norsram_struct_para_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);  |
| Function descriptions    | initialize the struct exmc_norsram_parameter_struct   |
| Precondition             | -   |
| The called functions     | -   |
| Input parameter{in}      |   |
| exmc_norsram_init_struct | Structure for initialization, the structure members can refer to <a href="#">Table 3-163. Structure exmc_norsram_parameter_struct</a> |
| Output parameter{out}    |   |
| -                        | -   |
| Return value             |   |
| -                        | -   |

Example:

```

/* initialize the struct nor_init_struct */

exmc_norsram_parameter_struct nor_init_struct;

exmc_norsram_struct_para_init (&nor_init_struct);

```

## exmc\_norsram\_enable

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

**Table 3-167. Function exmc\_norsram\_enable**

| Function name | exmc_norsram_enable |
|---------------|---------------------|
|---------------|---------------------|

|                              |                                 |
|------------------------------|---------------------------------|
| <b>Function prototype</b>    | void exmc_norsram_enable(void); |
| <b>Function descriptions</b> | enable EXMC NOR/PSRAM bank      |
| <b>Precondition</b>          | -                               |
| <b>The called functions</b>  | -                               |
| <b>Input parameter{in}</b>   |                                 |
| -                            | -                               |
| <b>Output parameter{out}</b> |                                 |
| -                            | -                               |
| <b>Return value</b>          |                                 |
| -                            | -                               |

Example:

```
/* enable bank */
```

```
exmc_norsram_enable();
```

## exmc\_norsram\_disable

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

**Table 3-168. Function exmc\_norsram\_disable**

|                              |                                  |
|------------------------------|----------------------------------|
| <b>Function name</b>         | exmc_norsram_disable             |
| <b>Function prototype</b>    | void exmc_norsram_disable(void); |
| <b>Function descriptions</b> | disable EXMC NOR/PSRAM bank      |
| <b>Precondition</b>          | -                                |
| <b>The called functions</b>  | -                                |
| <b>Input parameter{in}</b>   |                                  |
| -                            | -                                |
| <b>Output parameter{out}</b> |                                  |
| -                            | -                                |
| <b>Return value</b>          |                                  |
| -                            | -                                |

Example:

```
/* disable bank */
```

```
exmc_norsram_disable();
```

## exmc\_norsram\_page\_size\_config

The description of exmc\_norsram\_page\_size\_config is shown as below:

**Table 3-169. Function exmc\_norsram\_page\_size\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | exmc_norsram_page_size_config                           |
| <b>Function prototype</b>    | void exmc_norsram_page_size_config(uint32_t page_size); |
| <b>Function descriptions</b> | configure CRAM page size                                |

|                                       |   |
|---------------------------------------|---|
| <b>Precondition</b>                   | -   |
| <b>The called functions</b>           | -   |
| <b>Input parameter{in}</b>            |   |
| <b>page_size</b>                      | CRAM page size  |
| <i>EXMC_CRAM_AUTO_SPLIT</i>           | the clock is generated only during synchronous access |
| <i>EXMC_CRAM_PAGE_SIZE_128_BYTES</i>  | page size is 128 bytes                                |
| <i>EXMC_CRAM_PAGE_SIZE_256_BYTES</i>  | page size is 256 bytes                                |
| <i>EXMC_CRAM_PAGE_SIZE_512_BYTES</i>  | page size is 512 bytes                                |
| <i>EXMC_CRAM_PAGE_SIZE_1024_BYTES</i> | page size is 1024 bytes                               |
| <b>Output parameter{out}</b>          |   |
| -                                     | -   |
| <b>Return value</b>                   |   |
| -                                     | -   |

Example:

```
/* configure CRAM page size */
```

```
exmc_norsram_page_size_config (EXMC_CRAM_PAGE_SIZE_128_BYTES);
```

## 3.10. EXTI

EXTI is the interrupt / event controller in the MCU. It contains up to 19 independent edge detectors and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter [3.10.1](#), 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 3-170. EXTI Registers**

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

## 3.10.2. Descriptions of Peripheral functions

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

**Table 3-171. EXTI firmware function**

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

### Enum exti\_line\_enum

**Table 3-172. Enum exti\_line\_enum**

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

## Enum exti\_mode\_enum

**Table 3-173. Enum exti\_mode\_enum**

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

## Enum exti\_trig\_type\_enum

**Table 3-174. Enum exti\_trig\_type\_enum**

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

## exti\_deinit

The description of exti\_deinit is shown as below:

**Table 3-175. Function exti\_deinit**

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

Example:

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

## exti\_init

The description of exti\_init is shown as below:

**Table 3-176. 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 line x   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>linex</b>                 | EXTI line x, refer to <a href="#">Table 3-172. Enum exti_line_enum</a>       |
| <b>Input parameter{in}</b>   |  |
| <b>mode</b>                  | EXTI mode, refer to <a href="#">Table 3-173. Enum exti_mode_enum</a>         |
| <b>Input parameter{in}</b>   |  |
| <b>trig_type</b>             | trigger type, refer to <a href="#">Table 3-174. Enum exti_trig_type_enum</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

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

## exti\_interrupt\_enable

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

**Table 3-177. Function exti\_interrupt\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | exti_interrupt_enable  |
| <b>Function prototype</b>    | void exti_interrupt_enable(exti_line_enum linex);                      |
| <b>Function descriptions</b> | enable the interrupts from EXTI line x                                 |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>linex</b>                 | EXTI line x, refer to <a href="#">Table 3-172. Enum exti_line_enum</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

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

## exti\_interrupt\_disable

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

**Table 3-178. Function exti\_interrupt\_disable**

|                       |  |
|-----------------------|--|
| Function name         | exti_interrupt_disable   |
| Function prototype    | void exti_interrupt_disable(exti_line_enum linex);                     |
| Function descriptions | disable the interrupts from EXTI line x                                |
| Precondition          | -  |
| The called functions  | -  |
| Input parameter{in}   |  |
| linex                 | EXTI line x, refer to <a href="#">Table 3-172. Enum exti_line_enum</a> |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

Example:

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

```
exti_interrupt_disable(EXTI_0);
```

## exti\_event\_enable

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

**Table 3-179. Function exti\_event\_enable**

|                       |  |
|-----------------------|--|
| Function name         | exti_event_enable  |
| Function prototype    | void exti_event_enable(exti_line_enum linex);                          |
| Function descriptions | enable the events from EXTI line x                                     |
| Precondition          | -  |
| The called functions  | -  |
| Input parameter{in}   |  |
| linex                 | EXTI line x, refer to <a href="#">Table 3-172. Enum exti_line_enum</a> |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

Example:

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

```
exti_event_enable(EXTI_0);
```

## exti\_event\_disable

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

**Table 3-180. Function exti\_event\_disable**

|               |                    |
|---------------|--------------------|
| Function name | exti_event_disable |
|---------------|--------------------|

|                              |  |
|------------------------------|--|
| <b>Function prototype</b>    | void exti_event_disable(exti_line_enum linex);                         |
| <b>Function descriptions</b> | disable the events from EXTI line x                                    |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>linex</b>                 | EXTI line x, refer to <a href="#">Table 3-172. Enum exti_line_enum</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

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

## exti\_software\_interrupt\_enable

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

**Table 3-181. Function exti\_software\_interrupt\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | exti_software_interrupt_enable   |
| <b>Function prototype</b>    | void exti_software_interrupt_enable(exti_line_enum linex);             |
| <b>Function descriptions</b> | enable the software interrupt event from EXTI line x                   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>linex</b>                 | EXTI line x, refer to <a href="#">Table 3-172. Enum exti_line_enum</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

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

## exti\_software\_interrupt\_disable

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

**Table 3-182. Function exti\_software\_interrupt\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | exti_software_interrupt_disable                             |
| <b>Function prototype</b>    | void exti_software_interrupt_disable(exti_line_enum linex); |
| <b>Function descriptions</b> | disable the software interrupt event from EXTI line x       |



|                              |  |
|------------------------------|--|
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>linex</b>                 | EXTI line x, refer to <a href="#">Table 3-172. Enum exti_line_enum</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

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

```
exti_software_interrupt_disable(EXTI_0);
```

## exti\_flag\_get

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

**Table 3-183. Function exti\_flag\_get**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | exti_flag_get  |
| <b>Function prototype</b>    | FlagStatus exti_flag_get(exti_line_enum linex);                        |
| <b>Function descriptions</b> | get EXTI line x interrupt pending flag                                 |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>linex</b>                 | EXTI line x, refer to <a href="#">Table 3-172. Enum exti_line_enum</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>FlagStatus</b>            | SET or RESET   |

Example:

```
/* get EXTI line 0 flag status */
```

```
FlagStatus state = exti_flag_get(EXTI_0);
```

## exti\_flag\_clear

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

**Table 3-184. Function exti\_flag\_clear**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | exti_flag_clear                             |
| <b>Function prototype</b>    | void exti_flag_clear(exti_line_enum linex); |
| <b>Function descriptions</b> | clear EXTI line x interrupt pending flag    |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |

| Input parameter{in}   |  |
|-----------------------|--|
| linex                 | EXTI line x, refer to <a href="#">Table 3-172. Enum exti_line_enum</a> |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

Example:

```
/* clear EXTI line 0 flag status */
```

```
exti_flag_clear(EXTI_0);
```

### exti\_interrupt\_flag\_get

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

**Table 3-185. Function exti\_interrupt\_flag\_get**

| Function name         | exti_interrupt_flag_get  |
|-----------------------|--|
| Function prototype    | FlagStatus exti_interrupt_flag_get(exti_line_enum linex);              |
| Function descriptions | get EXTI line x interrupt pending flag                                 |
| Precondition          | -  |
| The called functions  | -  |
| Input parameter{in}   |  |
| linex                 | EXTI line x, refer to <a href="#">Table 3-172. Enum exti_line_enum</a> |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| FlagStatus            | SET or RESET   |

Example:

```
/* get EXTI line 0 interrupt flag status */
```

```
FlagStatus state = exti_interrupt_flag_get(EXTI_0);
```

### exti\_interrupt\_flag\_clear

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

**Table 3-186. Function exti\_interrupt\_flag\_clear**

| Function name         | exti_interrupt_flag_clear  |
|-----------------------|--|
| Function prototype    | void exti_interrupt_flag_clear(exti_line_enum linex);                  |
| Function descriptions | clear EXTI line x interrupt pending flag                               |
| Precondition          | -  |
| The called functions  | -  |
| Input parameter{in}   |  |
| linex                 | EXTI line x, refer to <a href="#">Table 3-172. Enum exti_line_enum</a> |

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

Example:

```
/* clear EXTI line 0 interrupt flag status */
```

```
exti_interrupt_flag_clear(EXTI_0);
```

## 3.11. FMC

There is flash controller and option byte for GD32E10x 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 3-187. FMC Registers**

| Registers  | Descriptions                      |
|------------|-----------------------------------|
| FMC_WS     | Wait state register               |
| FMC_KEY    | Unlock key register               |
| FMC_OBKEY  | Option byte unlock key register   |
| FMC_STAT   | Status register                   |
| FMC_CTL    | Control register                  |
| FMC_ADDR   | Address register                  |
| FMC_OBSTAT | Option byte status register       |
| FMC_WP     | Erase/Program Protection register |
| FMC_PID    | Product ID register               |

### 3.11.2. Descriptions of Peripheral functions

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

**Table 3-188. FMC firmware function**

| Function name        | Function description           |
|----------------------|--------------------------------|
| fmc_wsnt_set         | set the FMC wait state counter |
| fmc_prefetch_enable  | enable pre-fetch               |
| fmc_prefetch_disable | disable pre-fetch              |
| fmc_ibus_enable      | enable IBUS cache              |
| fmc_ibus_disable     | disable IBUS cache             |

| Function name                   | Function description                                   |
|---------------------------------|--|
| fmc_dbus_enable                 | enable DBUS cache                                      |
| fmc_dbus_disable                | disable DBUS cache                                     |
| fmc_ibus_reset                  | reset IBUS cache                                       |
| fmc_dbus_reset                  | reset DBUS cache                                       |
| fmc_program_width_set           | set program width to flash memory                      |
| fmc_unlock                      | unlock the main FMC operation                          |
| fmc_lock                        | lock the main FMC operation                            |
| fmc_page_erase                  | FMC erase page   |
| fmc_mass_erase                  | FMC erase whole chip                                   |
| fmc_doubleword_program          | FMC program a double word at the corresponding address |
| fmc_word_program                | FMC program a word at the corresponding address        |
| ob_unlock                       | unlock the option byte operation                       |
| ob_lock                         | lock the option byte operation                         |
| ob_erase                        | erase the option byte                                  |
| ob_write_protection_enable      | enable write protection                                |
| ob_security_protection_config   | configure the option byte security protection          |
| ob_user_write                   | write the FMC user option byte                         |
| ob_data_program                 | program option bytes data                              |
| ob_user_get                     | get the FMC user option byte                           |
| ob_data_get                     | get OB_DATA in register FMC_OBSTAT                     |
| ob_write_protection_get         | get the FMC option byte write protection               |
| ob_security_protection_flag_get | get option byte security protection state              |
| fmc_interrupt_enable            | enable FMC interrupt                                   |
| fmc_interrupt_disable           | disable FMC interrupt                                  |
| fmc_flag_get                    | check flag is set or not                               |
| fmc_flag_clear                  | clear the FMC flag                                     |
| fmc_interrupt_flag_get          | get FMC interrupt flag state                           |
| fmc_interrupt_flag_clear        | clear FMC interrupt flag state                         |
| fmc_state_get                   | get FMC state  |
| fmc_ready_wait                  | check FMC ready or not                                 |

## Enum fmc\_state\_enum

**Table 3-189. Enum fmc\_state\_enum**

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

## Enum fmc\_int\_enum

**Table 3-190. Enum fmc\_int\_enum**

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

## Enum fmc\_flag\_enum

**Table 3-191. Enum fmc\_flag\_enum**

| enum name           | enum description                        |
|---------------------|---|
| FMC_FLAG_BUSY       | FMC busy flag                           |
| FMC_FLAG_PGER<br>R  | FMC operation error flag                |
| FMC_FLAG_PGAE<br>RR | FMC program alignment error flag        |
| FMC_FLAG_WPER<br>R  | FMC erase/program protection error flag |
| FMC_FLAG_END        | FMC end of operation flag               |
| FMC_FLAG_OBER<br>R  | FMC option bytes read error flag        |

## Enum fmc\_interrupt\_flag\_enum

**Table 3-192. Enum fmc\_interrupt\_flag\_enum**

| enum name               | enum description                                  |
|-------------------------|---|
| FMC_INT_FLAG_P<br>GERR  | FMC operation error interrupt flag                |
| FMC_INT_FLAG_P<br>GAERR | FMC program alignment error interrupt flag bit    |
| FMC_INT_FLAG_W<br>PERR  | FMC erase/program protection error interrupt flag |
| FMC_INT_FLAG_E<br>ND    | FMC end of operation interrupt flag               |

## fmc\_wscnt\_set

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

**Table 3-193. Function fmc\_wscnt\_set**

| Function name         | fmc_wscnt_set                                      |
|-----------------------|--|
| Function prototype    | void fmc_wscnt_set(uint32_t wscnt);                |
| Function descriptions | set the wait state counter value                   |
| Precondition          | -  |
| The called functions  | rcu_periph_reset_enable / rcu_periph_reset_disable |

| Input parameter{in}     |                          |
|-------------------------|--------------------------|
| <b>wscnt</b>            | wait state counter value |
| <i>FMC_WAIT_STATE_0</i> | FMC 0 wait               |
| <i>FMC_WAIT_STATE_1</i> | FMC 1 wait               |
| <i>FMC_WAIT_STATE_2</i> | FMC 2 wait               |
| <i>FMC_WAIT_STATE_3</i> | FMC 3 wait               |
| Output parameter{out}   |                          |
| -                       | -                        |
| Return value            |                          |
| -                       | -                        |

Example:

```
/* set the wait state counter value */
fmc_wscnt_set (FMC_WAIT_STATE_1);
```

## fmc\_prefetch\_enable

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

**Table 3-194. Function fmc\_prefetch\_enable**

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

Example:

```
/* enable pre-fetch */
fmc_prefetch_enable( );
```

## fmc\_prefetch\_disable

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

**Table 3-195. Function fmc\_prefetch\_disable**

|                              |                                   |
|------------------------------|-----------------------------------|
| <b>Function name</b>         | fmc_prefetch_disable              |
| <b>Function prototype</b>    | void fmc_prefetch_disable (void); |
| <b>Function descriptions</b> | disable pre-fetch                 |

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

Example:

```
/* disable pre-fetch */
fmc_prefetch_disable( );
```

## fmc\_ibus\_enable

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

**Table 3-196. Function fmc\_ibus\_enable**

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

Example:

```
/* enable IBUS cache */
fmc_ibus_enable( );
```

## fmc\_ibus\_disable

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

**Table 3-197. Function fmc\_ibus\_disable**

|                              |                              |
|------------------------------|------------------------------|
| <b>Function name</b>         | fmc_ibus_disable             |
| <b>Function prototype</b>    | void fmc_ibus_disable(void); |
| <b>Function descriptions</b> | disable IBUS cache           |
| <b>Precondition</b>          | -                            |
| <b>The called functions</b>  | -                            |

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

Example:

```
/* disable IBUS cache */
```

```
fmc_ibus_disable( );
```

## fmc\_dbus\_enable

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

**Table 3-198. Function fmc\_dbus\_enable**

| Function name         | fmc_dbus_enable             |
|-----------------------|-----------------------------|
| Function prototype    | void fmc_dbus_enable(void); |
| Function descriptions | enable DBUS cache           |
| Precondition          | -                           |
| The called functions  | -                           |
| Input parameter{in}   |                             |
| -                     | -                           |
| Output parameter{out} |                             |
| -                     | -                           |
| Return value          |                             |
| -                     | -                           |

Example:

```
/* enable DBUS cache */
```

```
fmc_dbus_enable( );
```

## fmc\_dbus\_disable

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

**Table 3-199. Function fmc\_dbus\_disable**

| Function name         | fmc_dbus_disable             |
|-----------------------|------------------------------|
| Function prototype    | void fmc_dbus_disable(void); |
| Function descriptions | disable DBUS cache           |
| Precondition          | -                            |
| The called functions  | -                            |
| Input parameter{in}   |                              |
| -                     | -                            |



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

Example:

```
/* disable DBUS cache */
```

```
fmc_dbus_disable( );
```

## fmc\_ibus\_reset

The description of fmc\_ibus\_reset is shown as below:

**Table 3-200. Function fmc\_ibus\_reset**

| Function name         | fmc_ibus_reset              |
|-----------------------|-----------------------------|
| Function prototype    | void fmc_ibus_reset (void); |
| Function descriptions | reset IBUS cache            |
| Precondition          | -                           |
| The called functions  | -                           |
| Input parameter{in}   |                             |
| -                     | -                           |
| Output parameter{out} |                             |
| -                     | -                           |
| Return value          |                             |
| -                     | -                           |

Example:

```
/* reset IBUS cache */
```

```
fmc_ibus_reset( );
```

## fmc\_dbus\_reset

The description of fmc\_dbus\_reset is shown as below:

**Table 3-201. Function fmc\_dbus\_reset**

| Function name         | fmc_dbus_reset             |
|-----------------------|----------------------------|
| Function prototype    | void fmc_dbus_reset(void); |
| Function descriptions | reset DBUS cache           |
| Precondition          | -                          |
| The called functions  | -                          |
| Input parameter{in}   |                            |
| -                     | -                          |
| Output parameter{out} |                            |
| -                     | -                          |

| Return value |   |
|--------------|---|
| -            | - |

Example:

```
/* reset DBUS cache */
```

```
fmc_dbus_reset( );
```

## fmc\_program\_width\_set

The description of fmc\_program\_width\_set is shown as below:

**Table 3-202. Function fmc\_program\_width\_set**

| Function name         | fmc_program_width_set                     |
|-----------------------|---|
| Function prototype    | void fmc_program_width_set(uint32_t pgw); |
| Function descriptions | set program width to flash memory         |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |
| pgw                   | program width                             |
| FMC_PROG_W_32B        | 32-bit program width to flash memory      |
| FMC_PROG_W_64B        | 64-bit program width to flash memory      |
| Output parameter{out} |   |
| -                     | -   |
| Return value          |   |
| -                     | -   |

Example:

```
/* set program width to flash memory */
```

```
fmc_program_width_set(FMC_PROG_W_32B);
```

## fmc\_unlock

The description of fmc\_unlock is shown as below:

**Table 3-203. Function fmc\_unlock**

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

| Return value |   |
|--------------|---|
| -            | - |

Example:

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

```
fmc_unlock ( );
```

## fmc\_lock

The description of fmc\_lock is shown as below:

**Table 3-204. Function fmc\_lock**

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

Example:

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

```
fmc_lock( );
```

## fmc\_page\_erase

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

**Table 3-205. Function fmc\_page\_erase**

|                       |   |
|-----------------------|---|
| Function name         | fmc_page_erase  |
| Function prototype    | fmc_state_enum fmc_page_erase(uint32_t page_address); |
| Function descriptions | erase page  |
| Precondition          | fmc_unlock  |
| The called functions  | fmc_ready_wait  |
| Input parameter{in}   |   |
| page_address          | the page address to be erased                         |
| Output parameter{out} |   |
| -                     | -   |
| Return value          |   |
| fmc_state_enum        | <a href="#"><u>state of FMC</u></a>                   |

Example:

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

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

## fmc\_mass\_erase

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

**Table 3-206. 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               | <a href="#">state of FMC</a>          |

Example:

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

```
fmc_mass_erase ( );
```

## fmc\_doubleword\_program

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

**Table 3-207. Function fmc\_doubleword\_program**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | fmc_doubleword_program  |
| <b>Function prototype</b>    | fmc_state_enum fmc_doubleword_program(uint32_t address, uint64_t data); |
| <b>Function descriptions</b> | program a double 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               | <a href="#">state of FMC</a>  |

Example:

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

```
fmc_doubleword_program( 0x08004000,0xaabbccddeeffgghh);
```

## fmc\_word\_program

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

**Table 3-208. Function fmc\_word\_program**

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

Example:

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

```
fmc_word_program ( 0x08004000,0xaabbccdd);
```

## ob\_unlock

The description of ob\_unlock is shown as below:

**Table 3-209. 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 3-210. 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\_erase

The description of ob\_erase is shown as below:

**Table 3-211. Function ob\_erase**

|                              |                           |
|------------------------------|---------------------------|
| <b>Function name</b>         | ob_erase                  |
| <b>Function prototype</b>    | void ob_erase(void);      |
| <b>Function descriptions</b> | erase the FMC option byte |
| <b>Precondition</b>          | ob_unlock                 |
| <b>The called functions</b>  | fmc_ready_wait            |
| <b>Input parameter{in}</b>   |                           |
| -                            | -                         |
| <b>Output parameter{out}</b> |                           |
| -                            | -                         |
| <b>Return value</b>          |                           |
| -                            | -                         |

Example:

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

```
ob_erase ( );
```

## ob\_write\_protection\_enable

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

**Table 3-212. Function ob\_write\_protection\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | ob_write_protection_enable                                 |
| <b>Function prototype</b>    | fmc_state_enum ob_write_protection_enable(uint32_t ob_wp); |
| <b>Function descriptions</b> | enable write protection                                    |
| <b>Precondition</b>          | ob_unlock  |
| <b>The called functions</b>  | fmc_ready_wait   |
| <b>Input parameter{in}</b>   |  |
| <b>ob_wp</b>                 | enable write protection                                    |
| <i>OB_WP_x</i>               | write protect specify sector x                             |
| <i>OB_WP_ALL</i>             | write protect all sector                                   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>fmc_state_enum</b>        | <a href="#"><i>state of FMC</i></a>                        |

Example:

```
/* enable write protection */
```

```
ob_write_protection_enable (OB_WP_7);
```

## ob\_security\_protection\_config

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

**Table 3-213. Function ob\_security\_protection\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | ob_security_protection_config                                  |
| <b>Function prototype</b>    | fmc_state_enum ob_security_protection_config (uint8_t ob_spc); |
| <b>Function descriptions</b> | configure security protection                                  |
| <b>Precondition</b>          | ob_unlock  |
| <b>The called functions</b>  | fmc_ready_wait   |
| <b>Input parameter{in}</b>   |  |
| <b>ob_spc</b>                | specify security protection                                    |
| <i>FMC_NSPC</i>              | no security protection   |
| <i>FMC_USPC</i>              | under security protection                                      |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>fmc_state_enum</b>        | <a href="#"><i>state of FMC</i></a>                            |

Example:

```
/* enable security protection */
```

```
ob_security_protection_config (FMC_USPC);
```

## ob\_user\_write

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

**Table 3-214. Function ob\_user\_write**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | ob_user_write   |
| <b>Function prototype</b>    | fmc_state_enum ob_user_write(uint8_t ob_fwdgt, uint8_t ob_deepsleep, uint8_t ob_stdby); |
| <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_fwdgt</b>              | option byte watchdog value  |
| OB_FWDGT_SW                  | software free watchdog  |
| OB_FWDGT_HW                  | hardware free watchdog  |
| <b>Input parameter{in}</b>   |   |
| <b>ob_deepsleep</b>          | option byte deepsleep reset value   |
| OB_DEEPSLEEP_NRS<br>T        | no reset when entering deepsleep mode   |
| OB_DEEPSLEEP_RST             | generate a reset instead of entering deepsleep mode                                     |
| <b>Input parameter{in}</b>   |   |
| <b>ob_stdby</b>              | option byte standby reset value   |
| OB_STDBY_NRS                 | no reset when entering standby mode   |
| OB_STDBY_RST                 | generate a reset instead of entering standby mode                                       |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| fmc_state_enum               | <a href="#"><u>state of FMC</u></a>   |

Example:

```
/* configure user option byte */
```

```
ob_user_write(OB_FWDGT_HW, OB_DEEPSLEEP_RST, OB_STDBY_RST);
```

## ob\_data\_program

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

**Table 3-215. Function ob\_data\_program**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | ob_data_program   |
| <b>Function prototype</b>    | fmc_state_enum ob_data_program(uint32_t address, uint8_t data); |
| <b>Function descriptions</b> | program option bytes data                                       |



|                              |   |
|------------------------------|---|
| <b>Precondition</b>          | ob_unlock                                 |
| <b>The called functions</b>  | fmc_ready_wait                            |
| <b>Input parameter{in}</b>   |   |
| <b>address</b>               | the option bytes address to be programmed |
| <b>data</b>                  | the byte to be programmed                 |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| fmc_state_enum               | <a href="#"><u>state of FMC</u></a>       |

Example:

```
/* program option bytes data */
ob_data_program (0x1fff804, 0x56);
```

## ob\_user\_get

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

**Table 3-216. Function ob\_user\_get**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | ob_user_get                                  |
| <b>Function prototype</b>    | uint8_t ob_user_get(void);                   |
| <b>Function descriptions</b> | get the FMC user option byte                 |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| uint8_t                      | the FMC user option byte values(0xF0 – 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 3-217. Function ob\_data\_program**

|                              |                              |
|------------------------------|------------------------------|
| <b>Function name</b>         | ob_data_get                  |
| <b>Function prototype</b>    | Uin16_t ob_data_get(void);   |
| <b>Function descriptions</b> | get the FMC data option byte |
| <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 3-218. Function ob\_write\_protection\_get**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | ob_write_protection_get                                      |
| <b>Function prototype</b>    | uint32_t ob_write_protection_get(void);                      |
| <b>Function descriptions</b> | get the FMC option byte write protection                     |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>uint32_t</b>              | the FMC write protection option byte value(0x0 – 0xFFFFFFFF) |

Example:

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

```
uint32_t wp = ob_write_protection_get ( );
```

## ob\_security\_protection\_flag\_get

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

**Table 3-219. Function ob\_security\_protection\_flag\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | ob_security_protection_flag_get                   |
| <b>Function prototype</b>    | FlagStatus ob_security_protection_flag_get(void); |
| <b>Function descriptions</b> | get the FMC option byte security protection       |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |

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

Example:

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

```
FlagStatus spc = ob_security_protection_flag_get( );
```

## fmc\_interrupt\_enable

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

**Table 3-220. Function fmc\_interrupt\_enable**

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

Example:

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

```
fmc_interrupt_enable(FMC_INT_END);
```

## fmc\_interrupt\_disable

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

**Table 3-221. Function fmc\_interrupt\_disable**

|                       |   |
|-----------------------|---|
| Function name         | fmc_interrupt_disable                           |
| Function prototype    | void fmc_interrupt_disable(uint32_t interrupt); |
| Function descriptions | disable FMC interrupt                           |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |

|                              |                              |
|------------------------------|------------------------------|
| <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 3-222. 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 flag is set or not                    |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>flag</b>                  | check FMC flag                              |
| <i>FMC_FLAG_BUSY</i>         | FMC busy flag bit                           |
| <i>FMC_FLAG_PGERR</i>        | FMC operation error flag bit                |
| <i>FMC_FLAG_PGAERR</i>       | FMC program alignment error flag bit        |
| <i>FMC_FLAG_WPERR</i>        | FMC erase/program protection error flag bit |
| <i>FMC_FLAG_END</i>          | FMC end of operation flag bit               |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>FlagStatus</b>            | 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 3-223. 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                          |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>flag</b>                  | clear FMC flag                              |
| <i>FMC_FLAG_PGERR</i>        | FMC operation error flag bit                |
| <i>FMC_FLAG_PGAERR</i>       | FMC program alignment error flag bit        |
| <i>FMC_FLAG_WPERR</i>        | FMC erase/program protection error flag bit |
| <i>FMC_FLAG_END</i>          | FMC end of operation flag bit               |
| <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 3-224. Function fmc\_interrupt\_flag\_get**

|                                 |  |
|---------------------------------|--|
| <b>Function name</b>            | fmc_interrupt_flag_get   |
| <b>Function prototype</b>       | FlagStatus fmc_interrupt_flag_get(fmc_interrupt_flag_enum flag); |
| <b>Function descriptions</b>    | get FMC interrupt flag state                                     |
| <b>Precondition</b>             | -  |
| <b>The called functions</b>     | -  |
| <b>Input parameter{in}</b>      |  |
| <b>flag</b>                     | FMC flag   |
| <i>FMC_INT_FLAG_PGE<br/>RR</i>  | FMC operation error flag bit                                     |
| <i>FMC_INT_FLAG_PGA<br/>ERR</i> | FMC program alignment error flag bit                             |
| <i>FMC_INT_FLAG_WPE<br/>RR</i>  | FMC erase/program protection error flag bit                      |
| <i>FMC_INT_FLAG_END</i>         | FMC end of operation flag bit                                    |
| <b>Output parameter{out}</b>    |  |
| -                               | -  |
| <b>Return value</b>             |  |
| <b>FlagStatus</b>               | SET or RESET   |

Example:

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

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

## fmc\_interrupt\_flag\_clear

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

**Table 3-225. Function fmc\_interrupt\_flag\_clear**

|                                       |   |
|---------------------------------------|---|
| <b>Function name</b>                  | fmc_interrupt_flag_clear  |
| <b>Function prototype</b>             | FlagStatus fmc_interrupt_flag_clear (fmc_interrupt_flag_enum flag); |
| <b>Function descriptions</b>          | clear FMC interrupt flag state                                      |
| <b>Precondition</b>                   | -   |
| <b>The called functions</b>           | -   |
| <b>Input parameter{in}</b>            |   |
| <b>flag</b>                           | clear FMC flag  |
| <i>FMC_INT_FLAG_PGE</i><br><i>RR</i>  | FMC operation error flag bit  |
| <i>FMC_INT_FLAG_PGA</i><br><i>ERR</i> | FMC program alignment error flag bit                                |
| <i>FMC_INT_FLAG_WPE</i><br><i>RR</i>  | FMC erase/program protection error flag bit                         |
| <i>FMC_INT_FLAG_END</i>               | FMC end of operation flag bit                                       |
| <b>Output parameter{out}</b>          |   |
| -                                     | -   |
| <b>Return value</b>                   |   |
| -                                     | -   |

Example:

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

```
fmc_interrupt_flag_get (FMC_INT_FLAG_PGERR);
```

## fmc\_state\_get

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

**Table 3-226. 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> |                                     |

|                |                              |
|----------------|------------------------------|
| -              | -                            |
| Return value   |                              |
| fmc_state_enum | <a href="#">state of FMC</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 3-227. Function fmc\_ready\_wait**

|                       |  |
|-----------------------|--|
| Function name         | fmc_ready_wait                                   |
| Function prototype    | fmc_state_enum fmc_ready_wait(uint32_t timeout); |
| Function descriptions | check whether FMC is ready or not                |
| Precondition          | -  |
| The called functions  | fmc_state_get()                                  |
| Input parameter{in}   |  |
| timeout               | count of loop                                    |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| fmc_state_enum        | <a href="#">state of FMC</a>                     |

Example:

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

```
fmc_ready_wait (0x00001000 );
```

## 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 3-228. FWDGT Registers**

| Registers | Descriptions     |
|-----------|------------------|
| FWDGT_CTL | Control register |

| Registers  | Descriptions       |
|------------|--------------------|
| FWDGT_PSC  | Prescaler register |
| FWDGT_RLD  | Reload register    |
| FWDGT_STAT | Status register    |

## 3.12.2. Descriptions of Peripheral functions

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

**Table 3-229. FWDGT firmware function**

| Function name                             | Function description  |
|---|---|
| <code>fwdgt_write_enable</code>           | enable write access to FWDGT_PSC and FWDGT_RLD              |
| <code>fwdgt_write_disable</code>          | disable write access to FWDGT_PSC and FWDGT_RLD             |
| <code>fwdgt_enable</code>                 | start the free watchdog timer counter                       |
| <code>fwdgt_prescaler_value_config</code> | configure the free watchdog timer counter prescaler value   |
| <code>fwdgt_reload_value_config</code>    | configure the free watchdog timer counter reload value      |
| <code>fwdgt_counter_reload</code>         | reload the counter of FWDGT                                 |
| <code>fwdgt_config</code>                 | configure counter reload value, and prescaler divider value |
| <code>fwdgt_flag_get</code>               | get flag state of FWDGT                                     |

### `fwdgt_write_enable`

The description of `fwdgt_write_enable` is shown as below:

**Table 3-230. Function `fwdgt_write_enable`**

|                       |  |
|-----------------------|--|
| Function name         | <code>fwdgt_write_enable</code>                |
| Function prototype    | <code>void fwdgt_write_enable(void);</code>    |
| Function descriptions | enable write access to FWDGT_PSC and FWDGT_RLD |
| Precondition          | -  |
| The called functions  | -  |
| Input parameter{in}   |  |
| -                     | -  |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

Example:

```
/* enable write access to FWDGT_PSC and FWDGT_RLD */
fwdgt_write_enable ( );
```

### `fwdgt_write_disable`

The description of `fwdgt_write_disable` is shown as below:



**Table 3-231. Function fwdgt\_write\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | fwdgt_write_disable                             |
| <b>Function prototype</b>    | void fwdgt_write_disable(void);                 |
| <b>Function descriptions</b> | disable write access to FWDGT_PSC and FWDGT_RLD |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <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 3-232. Function fwdgt\_enable**

|                              |                                       |
|------------------------------|---------------------------------------|
| <b>Function name</b>         | fwdgt_enable                          |
| <b>Function prototype</b>    | void fwdgt_enable(void);              |
| <b>Function descriptions</b> | start the free watchdog timer counter |
| <b>Precondition</b>          | -                                     |
| <b>The called functions</b>  | -                                     |
| <b>Input parameter{in}</b>   |                                       |
| -                            | -                                     |
| <b>Output parameter{out}</b> |                                       |
| -                            | -                                     |
| <b>Return value</b>          |                                       |
| -                            | -                                     |

Example:

```
/* start the free watchdog timer counter */
fwdgt_enable ( );
```

## fwdgt\_prescaler\_value\_config

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

**Table 3-233. Function fwdgt\_prescaler\_value\_config**

|                      |                              |
|----------------------|------------------------------|
| <b>Function name</b> | fwdgt_prescaler_value_config |
|----------------------|------------------------------|

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

Example:

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

```
ErrStatus flag;
```

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

## **fwdgt\_reload\_value\_config**

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

**Table 3-234. Function fwdgt\_reload\_value\_config**

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

Example:

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

```
ErrStatus flag;
```

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

## fwdgt\_counter\_reload

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

**Table 3-235. Function fwdgt\_counter\_reload**

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

Example:

```
/* reload FWDGT counter */
```

```
fwdgt_counter_reload ( );
```

## fwdgt\_config

The description of fwdgt\_config is shown as below:

**Table 3-236. Function fwdgt\_config**

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

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

Example:

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

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

## fwdgt\_flag\_get

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

**Table 3-237. Function fwdgt\_flag\_get fwdgt\_write\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | fwdgt_flag_get                                      |
| <b>Function prototype</b>    | FlagStatus fwdgt_flag_get(uint16_t flag);           |
| <b>Function descriptions</b> | get flag state of FWDGT                             |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>flag</b>                  | flag to get   |
| <i>FWDGT_FLAG_PUD</i>        | a write operation to FWDGT_PSC register is on going |
| <i>FWDGT_FLAG_RUD</i>        | a write operation to FWDGT_RLD register is on going |
| <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);
```

```
if(status == RESET)
```

```
{
```

```
...
```

```
}else
```

```
{
```

```
...
```

```
}
```

## 3.13. GPIO

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 3-238. GPIO Registers**

| Registers    | Descriptions                       |
|--------------|------------------------------------|
| GPIOx_CTL0   | Port control register 0            |
| GPIOx_CTL1   | Port control register 1            |
| GPIOx_ISTAT  | Port input status register         |
| GPIOx_OCTL   | Port output control register       |
| GPIOx_BOP    | Port bit operate register          |
| GPIOx_BC     | Port bit clear register            |
| GPIOx_LOCK   | Port configuration lock register   |
| GPIOx_SPD    | Port bit speed register            |
| AFIO_EC      | Event control register             |
| AFIO_PCF0    | AFIO port configuration register 0 |
| AFIO_EXTISS0 | EXTI sources selection register 0  |
| AFIO_EXTISS1 | EXTI sources selection register 1  |
| AFIO_EXTISS2 | EXTI sources selection register 2  |
| AFIO_EXTISS3 | EXTI sources selection register 3  |
| AFIO_PCF1    | AFIO port configuration register 1 |
| AFIO_CPSCTL  | IO compensation control register   |

### 3.13.2. Descriptions of Peripheral functions

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

**Table 3-239. GPIO firmware function**

| Function name       | Function description                  |
|---------------------|---------------------------------------|
| gpio_deinit         | reset GPIO port                       |
| gpio_afio_deinit    | reset alternate function I/O(AFIO)    |
| gpio_init           | GPIO parameter initialization         |
| gpio_bit_set        | set GPIO pin                          |
| gpio_bit_reset      | reset GPIO pin                        |
| 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            |

| Function name              | Function description                            |
|----------------------------|---|
| gpio_output_bit_get        | get GPIO pin output status                      |
| gpio_output_port_get       | get GPIO port output status                     |
| gpio_pin_remap_config      | configure GPIO pin remap                        |
| gpio_exti_source_select    | select GPIO pin exti sources                    |
| gpio_event_output_config   | configure GPIO pin event output                 |
| gpio_event_output_enable   | enable GPIO pin event output                    |
| gpio_event_output_disable  | disable GPIO pin event output                   |
| gpio_pin_lock              | lock GPIO pin                                   |
| gpio_compensation_config   | configure the I/O compensation cell             |
| gpio_compensation_flag_get | check the I/O compensation cell is ready or not |

## gpio\_deinit

The description of gpio\_deinit is shown as below:

**Table 3-240. 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,E)                               |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

Example:

```
/* reset GPIOA */
gpio_deinit (GPIOA);
```

## gpio\_afio\_deinit

The description of gpio\_afio\_deinit is shown as below:

**Table 3-241. Function gpio\_afio\_deinit**

|                       |  |
|-----------------------|--|
| Function name         | gpio_afio_deinit                                   |
| Function prototype    | void gpio_afio_deinit(void);                       |
| Function descriptions | reset alternate function I/O(AFIO)                 |
| Precondition          | -  |
| The called functions  | rcu_periph_reset_enable / rcu_periph_reset_disable |

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

Example:

```
/* reset alternate function */
```

```
gpio_afio_deinit();
```

## gpio\_init

The description of gpio\_init is shown as below:

**Table 3-242. Function gpio\_init**

| Function name                | gpio_init  |
|------------------------------|--|
| Function prototype           | void gpio_init(uint32_t gpio_periph, uint32_t mode, uint32_t speed, uint32_t pin); |
| Function descriptions        | GPIO parameter initialization  |
| Precondition                 | -  |
| The called functions         | -  |
| Input parameter{in}          |  |
| <b>gpio_periph</b>           | GPIO port  |
| <i>GPIOx</i>                 | GPIOx(x = A,B,C,D,E)   |
| Input parameter{in}          |  |
| <b>gpio_mode</b>             | gpio pin mode  |
| <i>GPIO_MODE_AIN</i>         | analog input mode  |
| <i>GPIO_MODE_IN_FLOATING</i> | floating input mode  |
| <i>GPIO_MODE_IPD</i>         | pull-down input mode   |
| <i>GPIO_MODE_IPU</i>         | pull-up input mode   |
| <i>GPIO_MODE_OUT_OD</i>      | GPIO output with open-drain  |
| <i>GPIO_MODE_OUT_PP</i>      | GPIO output with push-pull   |
| <i>GPIO_MODE_AF_OD</i>       | AFIO output with open-drain  |
| <i>GPIO_MODE_AF_PP</i>       | AFIO output with push-pull   |
| Input parameter{in}          |  |
| <b>speed</b>                 | gpio output max speed value  |
| <i>GPIO_OSPEED_10MHZ</i>     | output max speed 10MHz   |
| <i>GPIO_OSPEED_2MHZ</i>      | output max speed 2MHz  |
| <i>GPIO_OSPEED_50MHZ</i>     | output max speed 50MHz   |

|                              |                                  |
|------------------------------|----------------------------------|
| <i>Z</i>                     |                                  |
| <i>GPIO_OSPEED_MAX</i>       | output max speed more than 50MHz |
| <b>Input parameter{in}</b>   |                                  |
| <b>pin</b>                   | GPIO pin                         |
| <i>GPIO_PIN_x</i>            | GPIO_PIN_x(x=0..15)              |
| <i>GPIO_PIN_ALL</i>          | All pins                         |
| <b>Output parameter{out}</b> |                                  |
| -                            | -                                |
| <b>Return value</b>          |                                  |
| -                            | -                                |

Example:

```
/* config PA0 as analog input mode */
```

```
gpio_init (GPIOA, GPIO_MODE_AIN, GPIO_OSPEED_50MHZ, GPIO_PIN_0);
```

## gpio\_bit\_set

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

**Table 3-243. Function gpio\_bit\_set**

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

Example:

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

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



## gpio\_bit\_reset

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

**Table 3-244. Function gpio\_bit\_reset**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | gpio_bit_reset  |
| <b>Function prototype</b>    | void gpio_bit_reset(uint32_t gpio_periph,uint32_t pin); |
| <b>Function descriptions</b> | reset GPIO pin  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>gpio_periph</b>           | GPIO port   |
| <i>GPIOx</i>                 | GPIOx(x = A,B,C,D,E)                                    |
| <b>Input parameter{in}</b>   |   |
| <b>pin</b>                   | GPIO pin  |
| <i>GPIO_PIN_x</i>            | GPIO_PIN_x(x=0..15)                                     |
| <i>GPIO_PIN_ALL</i>          | All pins  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* reset PA0 */
```

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

## gpio\_bit\_write

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

**Table 3-245. Function gpio\_bit\_write**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | gpio_bit_write   |
| <b>Function prototype</b>    | void gpio_bit_write(uint32_t gpio_periph,uint32_t pin,bit_status bit_value); |
| <b>Function descriptions</b> | write data to the specified GPIO pin   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>gpio_periph</b>           | GPIO port  |
| <i>GPIOx</i>                 | GPIOx(x = A,B,C,D,E)   |
| <b>Input parameter{in}</b>   |  |
| <b>pin</b>                   | GPIO pin   |
| <i>GPIO_PIN_x</i>            | GPIO_PIN_x(x=0..15)  |
| <i>GPIO_PIN_ALL</i>          | All pins   |
| <b>Input parameter{in}</b>   |  |

|                              |                    |
|------------------------------|--------------------|
| <b>bit_value</b>             | SET or RESET       |
| <i>RESET</i>                 | clear the port pin |
| <i>SET</i>                   | set the port pin   |
| <b>Output parameter{out}</b> |                    |
| -                            | -                  |
| <b>Return value</b>          |                    |
| -                            | -                  |

Example:

```
/* write 1 to PA0 */
```

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

## gpio\_port\_write

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

**Table 3-246. Function gpio\_port\_write**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | gpio_port_write  |
| <b>Function prototype</b>    | void gpio_port_write(uint32_t gpio_periph,uint16_t data);        |
| <b>Function descriptions</b> | write data to the specified GPIO port                            |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>gpio_periph</b>           | GPIO port  |
| <i>GPIOx</i>                 | GPIOx(x = A,B,C,D,E)   |
| <b>Input parameter{in}</b>   |  |
| <b>data</b>                  | specify the value to be written to the port output data register |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

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

```
gpio_port_write (GPIOA, 0xA5);
```

## gpio\_input\_bit\_get

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

**Table 3-247. Function gpio\_input\_bit\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | gpio_input_bit_get  |
| <b>Function prototype</b>    | FlagStatus gpio_input_bit_get(uint32_t gpio_periph,uint32_t pin); |
| <b>Function descriptions</b> | get GPIO pin input status   |

|                              |                      |
|------------------------------|----------------------|
| <b>Precondition</b>          | -                    |
| <b>The called functions</b>  | -                    |
| <b>Input parameter{in}</b>   |                      |
| <b>gpio_periph</b>           | GPIO port            |
| <i>GPIOx</i>                 | GPIOx(x = A,B,C,D,E) |
| <b>Input parameter{in}</b>   |                      |
| <b>pin</b>                   | GPIO pin             |
| <i>GPIO_PIN_x</i>            | GPIO_PIN_x(x=0..15)  |
| <i>GPIO_PIN_ALL</i>          | All pins             |
| <b>Output parameter{out}</b> |                      |
| -                            | -                    |
| <b>Return value</b>          |                      |
| <b>FlagStatus</b>            | SET / RESET          |

Example:

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

```
FlagStatus bit_state;
```

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

## gpio\_input\_port\_get

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

**Table 3-248. Function gpio\_input\_port\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | gpio_input_port_get                                 |
| <b>Function prototype</b>    | uint16_t gpio_input_port_get(uint32_t gpio_periph); |
| <b>Function descriptions</b> | get GPIO port input status                          |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>gpio_periph</b>           | GPIO port   |
| <i>GPIOx</i>                 | GPIOx(x = A,B,C,D,E)                                |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>uint16_t</b>              | 0x00-0xFF   |

Example:

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

```
uint16_t port_state;
```

```
port_state = gpio_input_bit_get (GPIOA);
```

## gpio\_output\_bit\_get

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

**Table 3-249. Function gpio\_output\_bit\_get**

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

Example:

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

```
FlagStatus bit_state;
```

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

## gpio\_output\_port\_get

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

**Table 3-250. 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 port output status                          |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>gpio_periph</b>           | GPIO port  |
| <i>GPIOx</i>                 | GPIOx(x = A,B,C,D,E)                                 |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |

|                 |           |
|-----------------|-----------|
| <b>Uint16_t</b> | 0x00-0xFF |
|-----------------|-----------|

Example:

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

uint16_t port_state;

port_state = gpio_output_port_get (GPIOA);
```

## gpio\_pin\_remap\_config

The description of gpio\_pin\_remap\_config is shown as below:

**Table 3-251. Function gpio\_pin\_remap\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | gpio_pin_remap_config   |
| <b>Function prototype</b>    | void gpio_pin_remap_config(uint32_t remap, ControlStatus newvalue); |
| <b>Function descriptions</b> | configure GPIO pin remap  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>gpio_remap</b>            | select the pin to remap   |
| GPIO_SPI0_REMAP              | SPI0 remapping  |
| GPIO_I2C0_REMAP              | I2C0 remapping  |
| GPIO_USART0_REMAP            | USART0 remapping  |
| GPIO_USART1_REMAP            | USART1 remapping  |
| GPIO_USART2_PARTIAL_REMAP    | USART2 partial remapping  |
| GPIO_USART2_FULL_REMAP       | USART2 full remapping   |
| GPIO_TIMER0_PARTIAL_REMAP    | TIMER0 partial remapping  |
| GPIO_TIMER0_FULL_REMAP       | TIMER0 full remapping   |
| GPIO_TIMER1_PARTIAL_REMAP1   | TIMER1 partial remapping  |
| GPIO_TIMER1_PARTIAL_REMAP2   | TIMER1 partial remapping  |
| GPIO_TIMER1_FULL_REMAP       | TIMER1 full remapping   |
| GPIO_TIMER2_PARTIAL_REMAP    | TIMER2 partial remapping  |
| GPIO_TIMER2_FULL_REMAP       | TIMER2 full remapping   |

|                                  |  |
|----------------------------------|--|
| <i>GPIO_TIMER3_REMAP</i>         | TIMER3 remapping                                   |
| <i>GPIO_PD01_REMAP</i>           | PD01 remapping                                     |
| <i>GPIO_ADC0_ETRGRT_REMAP</i>    | ADC0 external trigger routine conversion remapping |
| <i>GPIO_ADC1_ETRGRT_REMAP</i>    | ADC1 external trigger routine conversion remapping |
| <i>GPIO_TIMER4CH3_IRMAP</i>      | TIMER4 channel3 internal remapping                 |
| <i>GPIO_SWJ_NONJTRST_REMAP</i>   | full SWJ(JTAG-DP + SW-DP),but without NJTRST       |
| <i>GPIO_SWJ_SWDPENABLE_REMAP</i> | JTAG-DP disabled and SW-DP enabled                 |
| <i>GPIO_SWJ_DISABLE_REMAP</i>    | JTAG-DP disabled and SW-DP disabled                |
| <i>GPIO_SPI2_REMAP</i>           | SPI2 remapping                                     |
| <i>GPIO_TIMER1ITR0_REMAP</i>     | TIMER1 internal trigger 0 remapping                |
| <i>GPIO_TIMER8_REMAP</i>         | TIMER8 remapping                                   |
| <i>GPIO_EXMC_NADV_REMAP</i>      | EXMC_NADV connect/disconnect                       |
| <i>GPIO_CTC_REMAP0</i>           | CTC remapping(PD15)                                |
| <b>Input parameter{in}</b>       |  |
| <b>newvalue</b>                  | ENABLE / DISABLE                                   |
| <i>ENABLE</i>                    |  |
| <i>DISABLE</i>                   |  |
| <b>Output parameter{out}</b>     |  |
| -                                | -  |
| <b>Return value</b>              |  |
| -                                | -  |

Example:

```
/* enable SPI0 remapping */
```

```
gpio_pin_remap_config (GPIO_SPI0_REMAP, ENABLE);
```

## gpio\_exti\_source\_select

The description of gpio\_exti\_source\_select is shown as below:

**Table 3-252. Function gpio\_exti\_source\_select**

|                           |  |
|---------------------------|--|
| <b>Function name</b>      | gpio_exti_source_select  |
| <b>Function prototype</b> | void gpio_exti_source_select(uint8_t output_port, uint8_t output_pin); |

|                               |                                   |
|-------------------------------|-----------------------------------|
| <b>Function descriptions</b>  | select GPIO pin exti sources      |
| <b>Precondition</b>           | -                                 |
| <b>The called functions</b>   | -                                 |
| <b>Input parameter{in}</b>    |                                   |
| <b>gpio_outputport</b>        | gpio event output port            |
| <b>GPIO_PORT_SOURCE_GPIOx</b> | output port source (x= A,B,C,D,E) |
| <b>Input parameter{in}</b>    |                                   |
| <b>gpio_outputpin</b>         | gpio event output pin             |
| <b>GPIO_PIN_SOURCE_x</b>      | Pin number(x=0..15)               |
| <b>Output parameter{out}</b>  |                                   |
| -                             | -                                 |
| <b>Return value</b>           |                                   |
| -                             | -                                 |

Example:

```
/* config PA0 as EXTI source */
```

```
gpio_exti_source_select (GPIO_PORT_SOURCE_GPIOA, GPIO_PIN_SOURCE_0);
```

## gpio\_event\_output\_config

The description of gpio\_event\_output\_config is shown as below:

**Table 3-253. Function gpio\_event\_output\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | gpio_event_output_config  |
| <b>Function prototype</b>    | void gpio_event_output_config(uint8_t output_port, uint8_t output_pin); |
| <b>Function descriptions</b> | configure GPIO pin event output   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>gpio_outputport</b>       | gpio event output port  |
| <b>GPIO_EVENT_PORT_GPIOx</b> | event output port x (x= A,B,C,D,E)                                      |
| <b>Input parameter{in}</b>   |   |
| <b>gpio_outputpin</b>        | gpio event output pin   |
| <b>GPIO_EVENT_PIN_x</b>      | Pin number (x=0..15)  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* Config PA0 as the output of event */
```

gpio\_event\_output\_config (GPIO\_EVENT\_PORT\_GPIOA, GPIO\_EVENT\_PIN\_0);

## gpio\_event\_output\_enable

The description of gpio\_event\_output\_enable is shown as below:

**Table 3-254. Function gpio\_event\_output\_enable**

|                              |                                      |
|------------------------------|--------------------------------------|
| <b>Function name</b>         | gpio_event_output_enable             |
| <b>Function prototype</b>    | void gpio_event_output_enable(void); |
| <b>Function descriptions</b> | enable GPIO pin event output         |
| <b>Precondition</b>          | -                                    |
| <b>The called functions</b>  | -                                    |
| <b>Input parameter{in}</b>   |                                      |
| -                            | -                                    |
| <b>Output parameter{out}</b> |                                      |
| -                            | -                                    |
| <b>Return value</b>          |                                      |
| -                            | -                                    |

Example:

```
/* enable GPIO pin event output */
gpio_event_output_enable(void);
```

## gpio\_event\_output\_disable

The description of gpio\_event\_output\_disable is shown as below:

**Table 3-255. Function gpio\_event\_output\_disable**

|                              |                                       |
|------------------------------|---------------------------------------|
| <b>Function name</b>         | gpio_event_output_disable             |
| <b>Function prototype</b>    | void gpio_event_output_disable(void); |
| <b>Function descriptions</b> | disable GPIO pin event output         |
| <b>Precondition</b>          | -                                     |
| <b>The called functions</b>  | -                                     |
| <b>Input parameter{in}</b>   |                                       |
| -                            | -                                     |
| <b>Output parameter{out}</b> |                                       |
| -                            | -                                     |
| <b>Return value</b>          |                                       |
| -                            | -                                     |

Example:

```
/* disable GPIO pin event output */
gpio_event_output_disable(void);
```



## gpio\_pin\_lock

The description of gpio\_pin\_lock is shown as below:

**Table 3-256. 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   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>gpio_periph</b>           | GPIO port   |
| <i>GPIOx</i>                 | GPIOx(x = A,B,C,D,E)                                    |
| <b>Input parameter{in}</b>   |   |
| <b>pin</b>                   | GPIO pin  |
| <i>GPIO_PIN_x</i>            | GPIO_PIN_x(x=0..15)                                     |
| <i>GPIO_PIN_ALL</i>          | All pins  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* lock PA0 */
```

```
gpio_pin_lock (GPIOA, GPIO_PIN_0);
```

## gpio\_compensation\_config

The description of gpio\_compensation\_config is shown as below:

**Table 3-257. Function gpio\_compensation\_config**

|                                  |   |
|----------------------------------|---|
| <b>Function name</b>             | gpio_compensation_config                              |
| <b>Function prototype</b>        | void gpio_compensation_config(uint32_t 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>compensation</b>              | specifies the I/O compensation cell mode              |
| <i>GPIO_COMPENSATION_ENABLE</i>  | I/O compensation cell is enabled                      |
| <i>GPIO_COMPENSATION_DISABLE</i> | I/O compensation cell is disabled                     |
| <b>Output parameter{out}</b>     |   |
| -                                | -   |

| Return value |   |
|--------------|---|
| -            | - |

Example:

```
/* enabled I/O compensation cell */
```

```
gpio_compensation_config (GPIO_COMPENSATION_ENABLE);
```

## gpio\_compensation\_flag\_get

The description of gpio\_compensation\_flag\_get is shown as below:

**Table 3-258. Function gpio\_compensation\_flag\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | gpio_compensation_flag_get                      |
| <b>Function prototype</b>    | FlagStatus gpio_compensation_flag_get(void);    |
| <b>Function descriptions</b> | check the I/O compensation cell is ready or not |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>FlagStatus</b>            | SET or RESET                                    |

Example:

```
/* check the I/O compensation cell state */
```

```
FlagStatus cell_state;
```

```
cell_state = gpio_compensation_flag_get (void);
```

## 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 3-259. I2C Registers**

| Registers  | Descriptions                      |
|------------|-----------------------------------|
| I2C_CTL0   | Control register 0                |
| I2C_CTL1   | Control register 1                |
| I2C_SADDR0 | Slave address register 0          |
| I2C_SADDR1 | Slave address register 1          |
| I2C_DATA   | Transfer buffer register          |
| I2C_STAT0  | Transfer status register 0        |
| I2C_STAT1  | Transfer status register 1        |
| I2C_CKCFG  | Clock configure register          |
| I2C_RT     | Rise time register                |
| I2C_SAMCS  | SAM control and status 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 3-260. I2C firmware function**

| Function name                      | Function description  |
|------------------------------------|---|
| i2c_deinit                         | reset I2C   |
| i2c_clock_config                   | configure I2C clock   |
| i2c_mode_addr_config               | configure I2C address   |
| i2c_smbus_type_config              | select SMBus type   |
| i2c_ack_config                     | whether or not to send an ACK                                   |
| i2c_ackpos_config                  | configure I2C POAP position                                     |
| i2c_master_addressing              | master sends slave address                                      |
| i2c_dualaddr_enable                | enable dual-address mode  |
| i2c_dualaddr_disable               | disable dual-address mode                                       |
| i2c_enable                         | enable I2C  |
| i2c_disable                        | disable I2C   |
| i2c_start_on_bus                   | generate a START condition on I2C bus                           |
| i2c_stop_on_bus                    | generate a STOP condition on I2C bus                            |
| i2c_data_transmit                  | I2C transmit data function                                      |
| i2c_data_receive                   | I2C receive data function                                       |
| i2c_dma_config                     | configure I2C DMA mode  |
| i2c_dma_last_transfer_config       | configure whether next DMA EOT is DMA last transfer or not      |
| i2c_stretch_scl_low_config         | whether to stretch SCL low when data is not ready in slave mode |
| i2c_slave_response_to_gcall_config | whether or not to response to a general call                    |
| i2c_software_reset_config          | configure software reset of I2C                                 |
| i2c_pec_config                     | configure I2C PEC calculation                                   |
| i2c_pec_transfer_config            | configure whether to transfer PEC value                         |

| Function name            | Function description                   |
|--------------------------|--|
| i2c_pec_value_get        | get packet error checking value        |
| i2c_smbus_alert_config   | configure I2C alert through SMBA pin   |
| i2c_smbus_arp_config     | configure I2C ARP protocol in SMBus    |
| i2c_sam_enable           | enable SAM_V interface                 |
| i2c_sam_disable          | disable SAM_V interface                |
| i2c_sam_timeout_enable   | enable SAM_V interface timeout detect  |
| i2c_sam_timeout_disable  | disable SAM_V interface timeout detect |
| i2c_flag_get             | get I2C flag status                    |
| i2c_flag_clear           | clear I2C flag status                  |
| i2c_interrupt_enable     | enable I2C interrupt                   |
| i2c_interrupt_disable    | disable I2C interrupt                  |
| i2c_interrupt_flag_get   | get I2C interrupt flag status          |
| i2c_interrupt_flag_clear | clear I2C interrupt flag status        |

## Enum i2c\_flag\_enum

**Table 3-261. Enum i2c\_flag\_enum**

| Member name        | Function description   |
|--------------------|--|
| I2C_FLAG_SBSEND    | start condition sent out in master mode  |
| I2C_FLAG_ADDSEND   | address is sent in master mode or received and matches in slave mode             |
| I2C_FLAG_BTC       | byte transmission finishes   |
| I2C_FLAG_ADD10SEND | header of 10-bit address is sent in master mode                                  |
| I2C_FLAG_STPDET    | stop condition detected in slave mode  |
| I2C_FLAG_RBNE      | I2C_DATA is not empty during receiving   |
| I2C_FLAG_TBNE      | 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 |

| Member name  | Function description |
|--------------|----------------------|
| I2C_FLAG_TFF | txframe fall flag    |
| I2C_FLAG_TFR | txframe rise flag    |
| I2C_FLAG_RFF | rxframe fall flag    |
| I2C_FLAG_RFR | rxframe rise flag    |

## Enum i2c\_interrupt\_flag\_enum

**Table 3-262. Enum i2c\_interrupt\_flag\_enum**

| Member name            | Function description   |
|------------------------|--|
| I2C_INT_FLAG_SBSEND    | start condition sent out in master mode interrupt flag                                       |
| I2C_INT_FLAG_ADDSEND   | address is sent in master mode or received and matches in slave mode interrupt flag          |
| I2C_INT_FLAG_BTC       | byte transmission finishes interrupt flag  |
| I2C_INT_FLAG_ADD10SEND | header of 10-bit address is sent in master mode interrupt flag                               |
| I2C_INT_FLAG_STPDET    | stop condition detected in slave mode interrupt flag   |
| I2C_INT_FLAG_RBNE      | I2C_DATA is not empty during receiving interrupt flag  |
| I2C_INT_FLAG_TBE       | I2C_DATA is empty during transmitting interrupt flag   |
| I2C_INT_FLAG_BERR      | a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag |
| 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  |
| I2C_INT_FLAG_TFF       | txframe fall interrupt flag  |
| I2C_INT_FLAG_TFR       | txframe rise interrupt flag  |
| I2C_INT_FLAG_RFF       | rxframe fall interrupt flag  |
| I2C_INT_FLAG_RFR       | rxframe rise interrupt flag  |

## Enum i2c\_interrupt\_enum

**Table 3-263. Enum i2c\_interrupt\_enum**

| Member name | Function description   |
|-------------|------------------------|
| I2C_INT_ERR | error interrupt        |
| I2C_INT_EV  | event interrupt        |
| I2C_INT_BUF | buffer interrupt       |
| I2C_INT_TFF | txframe fall interrupt |
| I2C_INT_TFR | txframe rise interrupt |
| I2C_INT_RFF | rxframe fall interrupt |
| I2C_INT_RFR | rxframe rise interrupt |

## i2c\_deinit

The description of i2c\_deinit is shown as below:

**Table 3-264. Function i2c\_deinit**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_deinit   |
| <b>Function prototype</b>    | void i2c_deinit(uint32_t i2c_periph);              |
| <b>Function descriptions</b> | reset I2C  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | rcu_periph_reset_enable / rcu_periph_reset_disable |
| <b>Input parameter{in}</b>   |  |
| <b>i2c_periph</b>            | I2C peripheral                                     |
| <i>I2Cx</i>                  | (x=0,1)  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* reset I2C0 */
```

```
i2c_deinit (I2C0);
```

## i2c\_clock\_config

The description of i2c\_clock\_config is shown as below:

**Table 3-265. Function i2c\_clock\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_clock_config   |
| <b>Function prototype</b>    | void i2c_clock_config(uint32_t i2c_periph, uint32_t clkspeed, uint32_t dutycyc); |
| <b>Function descriptions</b> | configure I2C clock  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | rcu_clock_freq_get   |
| <b>Input parameter{in}</b>   |  |
| <b>i2c_periph</b>            | I2C peripheral   |
| <i>I2Cx</i>                  | (x=0,1)  |
| <b>Input parameter{in}</b>   |  |
| <b>clkspeed</b>              | i2c clock speed  |
| <b>Input parameter{in}</b>   |  |
| <b>duty cyc</b>              | duty cycle in fast mode  |
| <i>I2C_DTCY_2</i>            | T_low/T_high=2   |
| <i>I2C_DTCY_16_9</i>         | T_low/T_high=16/9  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |

| Return value |   |
|--------------|---|
| -            | - |

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 3-266. Function i2c\_mode\_addr\_config**

| Function name               | i2c_mode_addr_config  |
|-----------------------------|---|
| Function prototype          | void i2c_mode_addr_config(uint32_t i2c_periph, uint32_t mode, uint32_t addformat, uint32_t addr); |
| Function descriptions       | configure I2C address   |
| Precondition                | -   |
| The called functions        | -   |
| Input parameter{in}         |   |
| <b>i2c_periph</b>           | I2C peripheral  |
| <i>I2Cx</i>                 | (x=0,1)   |
| Input parameter{in}         |   |
| <b>mode</b>                 | I2C mode select   |
| <i>I2C_I2CMODE_ENABLE</i>   | I2C mode  |
| <i>I2C_SMBUSMODE_ENABLE</i> | SMBus mode  |
| Input parameter{in}         |   |
| <b>addformat</b>            | 7bits or 10bits   |
| <i>I2C_ADDFORMAT_7BITS</i>  | address format is 7 bits  |
| <i>I2C_ADDFORMAT_10BITS</i> | address format is 10 bits   |
| Input parameter{in}         |   |
| <b>addr</b>                 | I2C address   |
| Output parameter{out}       |   |
| -                           | -   |
| Return value                |   |
| -                           | -   |

Example:

```
/* configure I2C0 address as 0x82, using 7 bits */
```

```
i2c_mode_addr_config(I2C0, I2C_I2CMODE_ENABLE, I2C_ADDFORMAT_7BITS, 0x82);
```

## i2c\_smbus\_type\_config

The description of i2c\_smbus\_type\_config is shown as below:

**Table 3-267. Function i2c\_smbus\_type\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | i2c_smbus_type_config   |
| <b>Function prototype</b>    | void i2c_smbus_type_config(uint32_t i2c_periph, uint32_t type); |
| <b>Function descriptions</b> | select SMBus type   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>i2c_periph</b>            | I2C peripheral  |
| <i>I2Cx</i>                  | (x=0,1)   |
| <b>Input parameter{in}</b>   |   |
| <b>type</b>                  | Device or host  |
| <i>I2C_SMBUS_DEVICE</i>      | SMBus mode device type  |
| <i>I2C_SMBUS_HOST</i>        | SMBus mode host type  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* config I2C0 as SMBUS host type*/
```

```
i2c_smbus_type_config (I2C0, I2C_SMBUS_HOST);
```

## i2c\_ack\_config

The description of i2c\_ack\_config is shown as below:

**Table 3-268. Function i2c\_ack\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | i2c_ack_config  |
| <b>Function prototype</b>    | void i2c_ack_config(uint32_t i2c_periph, uint32_t ack); |
| <b>Function descriptions</b> | whether or not to send an ACK                           |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>i2c_periph</b>            | I2C peripheral  |
| <i>I2Cx</i>                  | (x=0,1)   |
| <b>Input parameter{in}</b>   |   |
| <b>ack</b>                   | whether or not to send an ACK                           |
| <i>I2C_ACK_ENABLE</i>        | ACK will be sent  |
| <i>I2C_ACK_DISABLE</i>       | ACK will not be sent                                    |
| <b>Output parameter{out}</b> |   |



|              |   |
|--------------|---|
| -            | - |
| Return value |   |
| -            | - |

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 3-269. Function i2c\_ackpos\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_ackpos_config  |
| <b>Function prototype</b>    | void i2c_ackpos_config(uint32_t i2c_periph, uint32_t pos);               |
| <b>Function descriptions</b> | configure I2C POAP position  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>i2c_periph</b>            | I2C peripheral   |
| <i>I2Cx</i>                  | (x=0,1)  |
| <b>Input parameter{in}</b>   |  |
| <b>pos</b>                   | ACK position   |
| <i>I2C_ACKPOS_CURRENT</i>    | ACKEN bit decides whether or not to send ACK or not for the current byte |
| <i>I2C_ACKPOS_NEXT</i>       | ACKEN bit decides whether or not to send ACK for the next byte           |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* The ACK of I2C0 is send for the current frame */
```

```
i2c_ackpos_config (I2C0, I2C_ACKPOS_CURRENT);
```

## i2c\_master\_addressing

The description of i2c\_master\_addressing is shown as below:

**Table 3-270. Function i2c\_master\_addressing**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | i2c_master_addressing   |
| <b>Function prototype</b>    | void i2c_master_addressing(uint32_t i2c_periph, uint32_t addr, uint32_t trandirection); |
| <b>Function descriptions</b> | master sends slave address  |

|                              |                         |
|------------------------------|-------------------------|
| <b>Precondition</b>          | -                       |
| <b>The called functions</b>  | -                       |
| <b>Input parameter{in}</b>   |                         |
| <b>i2c_periph</b>            | I2C peripheral          |
| <i>I2Cx</i>                  | (x=0,1)                 |
| <b>Input parameter{in}</b>   |                         |
| <b>addr</b>                  | slave address           |
| <b>Input parameter{in}</b>   |                         |
| <b>trandirection</b>         | transmitter or receiver |
| <i>I2C_TRANSMITTER</i>       | transmitter             |
| <i>I2C_RECEIVER</i>          | receiver                |
| <b>Output parameter{out}</b> |                         |
| -                            | -                       |
| <b>Return value</b>          |                         |
| -                            | -                       |

Example:

```
/* send slave address to I2C bus and I2C0 act as receiver */
```

```
i2c_master_addressing(I2C0, 0x82, I2C_RECEIVER);
```

## i2c\_dualaddr\_enable

The description of i2c\_dualaddr\_enable is shown as below:

**Table 3-271. Function i2c\_dualaddr\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_dualaddr_enable  |
| <b>Function prototype</b>    | void i2c_dualaddr_enable(uint32_t i2c_periph, uint32_t addr) |
| <b>Function descriptions</b> | enable dual-address mode                                     |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>i2c_periph</b>            | I2C peripheral   |
| <i>I2Cx</i>                  | (x=0,1)  |
| <b>Input parameter{in}</b>   |  |
| <b>addr</b>                  | second address in dual-address mode                          |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable I2C0 dual-address*/
```

```
i2c_dualaddr_enable (I2C0, 0x80);
```

## i2c\_dualaddr\_disable

The description of i2c\_dualaddr\_disable is shown as below:

**Table 3-272. Function i2c\_dualaddr\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_dualaddr_disable                           |
| <b>Function prototype</b>    | void i2c_dualaddr_disable(uint32_t i2c_periph) |
| <b>Function descriptions</b> | disable dual-address mode                      |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>i2c_periph</b>            | I2C peripheral                                 |
| <i>I2Cx</i>                  | (x=0,1)  |
| <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 3-273. Function i2c\_enable**

|                              |                                       |
|------------------------------|---------------------------------------|
| <b>Function name</b>         | i2c_enable                            |
| <b>Function prototype</b>    | void i2c_enable(uint32_t i2c_periph); |
| <b>Function descriptions</b> | enable I2C                            |
| <b>Precondition</b>          | -                                     |
| <b>The called functions</b>  | -                                     |
| <b>Input parameter{in}</b>   |                                       |
| <b>i2c_periph</b>            | I2C peripheral                        |
| <i>I2Cx</i>                  | (x=0,1)                               |
| <b>Output parameter{out}</b> |                                       |
| -                            | -                                     |
| <b>Return value</b>          |                                       |
| -                            | -                                     |

Example:

```
/* enable I2C0 */
```

```
i2c_enable (I2C0);
```

## i2c\_disable

The description of i2c\_disable is shown as below:

**Table 3-274. Function i2c\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_disable                            |
| <b>Function prototype</b>    | void i2c_disable(uint32_t i2c_periph); |
| <b>Function descriptions</b> | disable I2C                            |
| <b>Precondition</b>          | -                                      |
| <b>The called functions</b>  | -                                      |
| <b>Input parameter{in}</b>   |  |
| <b>i2c_periph</b>            | I2C peripheral                         |
| <i>I2Cx</i>                  | (x=0,1)                                |
| <b>Output parameter{out}</b> |  |
| -                            | -                                      |
| <b>Return value</b>          |  |
| -                            | -                                      |

Example:

```
/* disable I2C0 */
i2c_disable (I2C0);
```

## i2c\_start\_on\_bus

The description of i2c\_start\_on\_bus is shown as below:

**Table 3-275. Function i2c\_start\_on\_bus**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | i2c_start_on_bus                            |
| <b>Function prototype</b>    | void i2c_start_on_bus(uint32_t i2c_periph); |
| <b>Function descriptions</b> | generate a START condition on I2C bus       |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>i2c_periph</b>            | I2C peripheral                              |
| <i>I2Cx</i>                  | (x=0,1)                                     |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* I2C0 send a start condition to I2C bus */
i2c_start_on_bus (I2C0);
```

## i2c\_stop\_on\_bus

The description of i2c\_stop\_on\_bus is shown as below:

**Table 3-276. Function i2c\_stop\_on\_bus**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_stop_on_bus                            |
| <b>Function prototype</b>    | void i2c_stop_on_bus(uint32_t i2c_periph); |
| <b>Function descriptions</b> | generate a STOP condition on I2C bus       |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>i2c_periph</b>            | I2C peripheral                             |
| <i>I2Cx</i>                  | (x=0,1)                                    |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* I2C0 generate a STOP condition to I2C bus */
```

```
i2c_stop_on_bus (I2C0);
```

## i2c\_data\_transmit

The description of i2c\_data\_transmit is shown as below:

**Table 3-277. Function i2c\_data\_transmit**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_data_transmit  |
| <b>Function prototype</b>    | void i2c_data_transmit(uint32_t i2c_periph, uint8_t data); |
| <b>Function descriptions</b> | I2C transmit data function                                 |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>i2c_periph</b>            | I2C peripheral   |
| <i>I2Cx</i>                  | (x=0,1)  |
| <b>Input parameter{in}</b>   |  |
| <b>data</b>                  | transmit data  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* I2C0 transmit data */
```

i2c\_data\_transmit (I2C0, 0x80);

## i2c\_data\_receive

The description of i2c\_data\_receive is shown as below:

**Table 3-278. Function i2c\_data\_receive**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_data_receive                               |
| <b>Function prototype</b>    | uint8_t i2c_data_receive(uint32_t i2c_periph); |
| <b>Function descriptions</b> | I2C receive data function                      |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>i2c_periph</b>            | I2C peripheral                                 |
| <i>I2Cx</i>                  | (x=0,1)  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>uint8_t</b>               | 0x00..0xFF                                     |

Example:

```
/* I2C0 receive data */
```

```
uint8_t i2c_receiver;
```

```
i2c_receiver = i2c_data_receive(I2C0);
```

## i2c\_dma\_config

The description of i2c\_dma\_config is shown as below:

**Table 3-279. Function i2c\_dma\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_dma_config   |
| <b>Function prototype</b>    | void i2c_dma_config(uint32_t i2c_periph, uint32_t dmastate); |
| <b>Function descriptions</b> | configure I2C DMA mode                                       |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>i2c_periph</b>            | I2C peripheral   |
| <i>I2Cx</i>                  | (x=0,1)  |
| <b>Input parameter{in}</b>   |  |
| <b>dmastate</b>              | on or off  |
| <i>I2C_DMA_ON</i>            | enable DMA mode  |
| <i>I2C_DMA_OFF</i>           | disable DMA mode   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |

| Return value |   |
|--------------|---|
| -            | - |

Example:

```
/* I2C0 DMA mode enable */
```

```
i2c_dma_config (I2C0, I2C_DMA_ON);
```

## i2c\_dma\_last\_transfer\_config

The description of i2c\_dma\_last\_transfer\_config is shown as below:

**Table 3-280. Function i2c\_dma\_last\_transfer\_config**

| Function name         | i2c_dma_last_transfer_config  |
|-----------------------|---|
| Function prototype    | void i2c_dma_last_transfer_config(uint32_t i2c_periph, uint32_t dmalast); |
| Function descriptions | configure whether next DMA EOT is DMA last transfer or not                |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |
| i2c_periph            | I2C peripheral  |
| I2Cx                  | (x=0,1)   |
| Input parameter{in}   |   |
| 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                                     |
| Output parameter{out} |   |
| -                     | -   |
| Return value          |   |
| -                     | -   |

Example:

```
/* next DMA EOT is the last transfer */
```

```
i2c_dma_last_transfer_config (I2C0, I2C_DMALST_ON);
```

## i2c\_stretch\_scl\_low\_config

The description of i2c\_stretch\_scl\_low\_config is shown as below:

**Table 3-281. Function i2c\_stretch\_scl\_low\_config**

| Function name         | i2c_stretch_scl_low_config  |
|-----------------------|---|
| Function prototype    | void i2c_stretch_scl_low_config(uint32_t i2c_periph, uint32_t stretchpara); |
| Function descriptions | whether to stretch SCL low when data is not ready in slave mode             |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |

|                               |                                  |
|-------------------------------|----------------------------------|
| <b>i2c_periph</b>             | I2C peripheral                   |
| <i>I2Cx</i>                   | (x=0,1)                          |
| <b>Input parameter{in}</b>    |                                  |
| <b>stretchpara</b>            | SCL stretching enable or disable |
| <i>I2C_SCLSTRETCH_ENABLE</i>  | enable SCL stretching            |
| <i>I2C_SCLSTRETCH_DISABLE</i> | disable SCL stretching           |
| <b>Output parameter{out}</b>  |                                  |
| -                             | -                                |
| <b>Return value</b>           |                                  |
| -                             | -                                |

Example:

```
/* stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_config (I2C0, I2C_SCLSTRETCH_ENABLE);
```

## i2c\_slave\_response\_to\_gcall\_config

The description of i2c\_slave\_response\_to\_gcall\_config is shown as below:

**Table 3-282. Function i2c\_slave\_response\_to\_gcall\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | i2c_slave_response_to_gcall_config  |
| <b>Function prototype</b>    | void i2c_slave_response_to_gcall_config(uint32_t i2c_periph, uint32_t gcallpara); |
| <b>Function descriptions</b> | whether or not to response to a general call                                      |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>i2c_periph</b>            | I2C peripheral  |
| <i>I2Cx</i>                  | (x=0,1)   |
| <b>Input parameter{in}</b>   |   |
| <b>gcallpara</b>             | response to a general call or not   |
| <i>I2C_GCEN_ENABLE</i>       | slave will response to a general call   |
| <i>I2C_GCEN_DISABLE</i>      | slave will not response to a general call   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* I2C0 will response to a general call */
```

```
i2c_slave_response_to_gcall_config (I2C0, I2C_GCEN_ENABLE);
```



## i2c\_software\_reset\_config

The description of i2c\_software\_reset\_config is shown as below:

**Table 3-283. Function i2c\_software\_reset\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | i2c_software_reset_config   |
| <b>Function prototype</b>    | void i2c_software_reset_config(uint32_t i2c_periph, uint32_t sreset); |
| <b>Function descriptions</b> | configure software reset of I2C                                       |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>i2c_periph</b>            | I2C peripheral  |
| <i>I2Cx</i>                  | (x=0,1)   |
| <b>Input parameter{in}</b>   |   |
| <b>sreset</b>                | reset or not  |
| <i>I2C_SRESET_SET</i>        | I2C is under reset  |
| <i>I2C_SRESET_RESET</i>      | I2C is not under reset  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* software reset I2C0 */
```

```
i2c_software_reset_config (I2C0, I2C_SRESET_SET);
```

## i2c\_pec\_config

The description of i2c\_pec\_config is shown as below:

**Table 3-284. Function i2c\_pec\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_pec_config   |
| <b>Function prototype</b>    | void i2c_pec_config(uint32_t i2c_periph, uint32_t pecstate); |
| <b>Function descriptions</b> | configure I2C PEC calculation or not                         |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>i2c_periph</b>            | I2C peripheral   |
| <i>I2Cx</i>                  | (x=0,1)  |
| <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> |  |

|              |   |
|--------------|---|
| -            | - |
| Return value |   |
| -            | - |

Example:

```
/* Enable I2C PEC calculation */
```

```
i2c_pec_config (I2C0, I2C_PEC_ENABLE);
```

## i2c\_pec\_transfer\_config

The description of i2c\_pec\_transfer\_enable is shown as below:

**Table 3-285. Function i2c\_pec\_transfer\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_pec_transfer_config  |
| <b>Function prototype</b>    | void i2c_pec_transfer_config(uint32_t i2c_periph, uint32_t pecpara); |
| <b>Function descriptions</b> | configure whether to transfer PEC value                              |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>i2c_periph</b>            | I2C peripheral   |
| <i>I2Cx</i>                  | (x=0,1)  |
| <b>Input parameter{in}</b>   |  |
| <b>pecpara</b>               | Transfer PEC or not  |
| <i>I2C_PECTRANS_ENA BLE</i>  | transfer PEC   |
| <i>I2C_PECTRANS_DISA BLE</i> | not transfer PEC   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* I2C0 transfer PEC */
```

```
i2c_pec_transfer_config (I2C0, I2C_PECTRANS_ENABLE);
```

## i2c\_pec\_value\_get

The description of i2c\_pec\_value\_get is shown as below:

**Table 3-286. Function i2c\_pec\_value\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | i2c_pec_value_get                               |
| <b>Function prototype</b>    | uint8_t i2c_pec_value_get(uint32_t i2c_periph); |
| <b>Function descriptions</b> | get packet error checking value                 |

|                              |                |
|------------------------------|----------------|
| <b>Precondition</b>          | -              |
| <b>The called functions</b>  | -              |
| <b>Input parameter{in}</b>   |                |
| <b>i2c_periph</b>            | I2C peripheral |
| <i>I2Cx</i>                  | (x=0,1)        |
| <b>Output parameter{out}</b> |                |
| -                            | -              |
| <b>Return value</b>          |                |
| <b>uint8_t</b>               | PEC value      |

Example:

```
/* I2C0 get packet error checking value */
```

```
uint8_t pec_value;
```

```
pec_value = i2c_pec_value_get (I2C0);
```

## i2c\_smbus\_alert\_config

The description of i2c\_smbus\_alert\_config is shown as below:

**Table 3-287. Function i2c\_smbus\_alert\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_smbus_alert_config   |
| <b>Function prototype</b>    | void i2c_smbus_alert_config (uint32_t i2c_periph, uint32_t smbuspara); |
| <b>Function descriptions</b> | configure I2C alert through SMBA pin                                   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>i2c_periph</b>            | I2C peripheral   |
| <i>I2Cx</i>                  | (x=0,1)  |
| <b>Input parameter{in}</b>   |  |
| <b>smbuspara</b>             | issue alert through SMBA pin or not                                    |
| <i>I2C_SALTSEND_ENABLE</i>   | issue alert through SMBA pin   |
| <i>I2C_SALTSEND_DISABLE</i>  | not issue alert through SMBA pin                                       |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* I2C0 issue alert through SMBA pin enable*/
```

```
i2c_smbus_alert_config (I2C0, I2C_SALTSEND_ENABLE);
```

## i2c\_smbus\_arp\_config

The description of i2c\_smbus\_arp\_config is shown as below:

**Table 3-288. Function i2c\_smbus\_arp\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_smbus_arp_config   |
| <b>Function prototype</b>    | void i2c_smbus_arp_config(uint32_t i2c_periph, uint32_t arpstate); |
| <b>Function descriptions</b> | configure I2C ARP protocol in SMBus                                |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>i2c_periph</b>            | I2C peripheral   |
| <i>I2Cx</i>                  | (x=0,1)  |
| <b>Input parameter{in}</b>   |  |
| <b>arpstate</b>              | ARP protocol in SMBus switch                                       |
| <i>I2C_ARP_ENABLE</i>        | enable ARP   |
| <i>I2C_ARP_DISABLE</i>       | disable ARP  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable I2C0 ARP protocol in SMBus switch */
```

```
i2c_smbus_arp_config (I2C0, I2C_ARP_ENABLE);
```

## i2c\_sam\_enable

The description of i2c\_sam\_enable is shown as below:

**Table 3-289. Function i2c\_sam\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | i2c_sam_enable                            |
| <b>Function prototype</b>    | void i2c_sam_enable(uint32_t i2c_periph); |
| <b>Function descriptions</b> | enable SAM_V interface                    |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>i2c_periph</b>            | I2C peripheral                            |
| <i>I2Cx</i>                  | (x=0,1)                                   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable I2C0 SAM_V interface */

i2c_sam_enable (I2C0);
```

## i2c\_sam\_disable

The description of i2c\_sam\_disable is shown as below:

**Table 3-290. Function i2c\_sam\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_sam_disable                            |
| <b>Function prototype</b>    | void i2c_sam_disable(uint32_t i2c_periph); |
| <b>Function descriptions</b> | disable SAM_V interface                    |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>i2c_periph</b>            | I2C peripheral                             |
| <i>I2Cx</i>                  | (x=0,1)                                    |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable I2C0 SAM_V interface */

i2c_sam_disable (I2C0);
```

## i2c\_sam\_timeout\_enable

The description of i2c\_sam\_timeout\_enable is shown as below:

**Table 3-291. Function i2c\_sam\_timeout\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | i2c_sam_timeout_enable                            |
| <b>Function prototype</b>    | void i2c_sam_timeout_enable(uint32_t i2c_periph); |
| <b>Function descriptions</b> | enable SAM_V interface timeout detect             |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>i2c_periph</b>            | I2C peripheral                                    |
| <i>I2Cx</i>                  | (x=0,1)   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable I2C0 SAM_V interface timeout detect */
```

```
i2c_sam_timeout_enable (I2C0);
```

## i2c\_sam\_timeout\_disable

The description of i2c\_sam\_timeout\_disable is shown as below:

**Table 3-292. Function i2c\_sam\_timeout\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_sam_timeout_disable                            |
| <b>Function prototype</b>    | void i2c_sam_timeout_disable(uint32_t i2c_periph); |
| <b>Function descriptions</b> | disable SAM_V interface timeout detect             |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>i2c_periph</b>            | I2C peripheral                                     |
| <i>I2Cx</i>                  | (x=0,1)  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable I2C0 SAM_V interface timeout detect */
```

```
i2c_sam_timeout_disable (I2C0);
```

## i2c\_flag\_get

The description of i2c\_flag\_get is shown as below:

**Table 3-293. Function i2c\_flag\_get**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_flag_get   |
| <b>Function prototype</b>    | FlagStatus i2c_flag_get(uint32_t i2c_periph, i2c_flag_enum flag)     |
| <b>Function descriptions</b> | check I2C flag is set or not   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>i2c_periph</b>            | I2C peripheral   |
| <i>I2Cx</i>                  | (x=0,1)  |
| <b>Input parameter{in}</b>   |  |
| <b>flag</b>                  | specify get which flag   |
| <i>I2C_FLAG_SBSEND</i>       | start condition sent out in master mode                              |
| <i>I2C_FLAG_ADDSEND</i>      | address is sent in master mode or received and matches in slave mode |

|                              |  |
|------------------------------|--|
| <i>I2C_FLAG_BTC</i>          | byte transmission finishes   |
| <i>I2C_FLAG_ADD10SEND</i>    | header of 10-bit address is sent in master mode                                  |
| <i>I2C_FLAG_STPDET</i>       | stop condition detected in slave mode  |
| <i>I2C_FLAG_RBNE</i>         | I2C_DATA is not empty during receiving   |
| <i>I2C_FLAG_TBE</i>          | I2C_DATA is empty during transmitting  |
| <i>I2C_FLAG_BERR</i>         | a bus error occurs indication a unexpected start or stop condition on I2C bus    |
| <i>I2C_FLAG_LOSTARB</i>      | arbitration lost in master mode  |
| <i>I2C_FLAG_AERR</i>         | acknowledge error  |
| <i>I2C_FLAG_OUERR</i>        | over-run or under-run situation occurs in slave mode                             |
| <i>I2C_FLAG_PECERR</i>       | PEC error when receiving data  |
| <i>I2C_FLAG_SMBTO</i>        | timeout signal in SMBus mode   |
| <i>I2C_FLAG_SMBALT</i>       | SMBus alert status   |
| <i>I2C_FLAG_MASTER</i>       | a flag indicating whether I2C block is in master or slave mode                   |
| <i>I2C_FLAG_I2CBSY</i>       | busy flag  |
| <i>I2C_FLAG_TR</i>           | whether the I2C is a transmitter or a receiver                                   |
| <i>I2C_FLAG_RXGC</i>         | general call address (00h) received  |
| <i>I2C_FLAG_DEFSMB</i>       | default address of SMBus device  |
| <i>I2C_FLAG_HSTSMB</i>       | SMBus host header detected in slave mode   |
| <i>I2C_FLAG_DUMOD</i>        | dual flag in slave mode indicating which address is matched in dual-address mode |
| <i>I2C_FLAG_TFF</i>          | txframe fall flag  |
| <i>I2C_FLAG_TFR</i>          | txframe rise flag  |
| <i>I2C_FLAG_RFF</i>          | rxframe fall flag  |
| <i>I2C_FLAG_RFR</i>          | rxframe rise flag  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>FlagStatus</b>            | SET / RESET  |

Example:

```
/* check whether start condition send out */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_flag_get (I2C0, I2C_FLAG_SBSEND);
```

### i2c\_flag\_clear

The description of i2c\_flag\_clear is shown as below:

**Table 3-294. Function i2c\_flag\_clear**

|                           |  |
|---------------------------|--|
| <b>Function name</b>      | i2c_flag_clear   |
| <b>Function prototype</b> | void i2c_flag_clear(uint32_t i2c_periph, i2c_flag_enum flag) |

|                              |   |
|------------------------------|---|
| <b>Function descriptions</b> | clear I2C flag status   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>i2c_periph</b>            | I2C peripheral  |
| <i>I2Cx</i>                  | (x=0,1)   |
| <b>Input parameter{in}</b>   |   |
| <b>flag</b>                  | flag type   |
| <i>I2C_FLAG_SMBALT</i>       | SMBus Alert status  |
| <i>I2C_FLAG_SMBTO</i>        | timeout signal in SMBus mode  |
| <i>I2C_FLAG_PECERR</i>       | PEC error when receiving data   |
| <i>I2C_FLAG_OUERR</i>        | over-run or under-run situation occurs in slave mode                          |
| <i>I2C_FLAG_AERR</i>         | acknowledge error   |
| <i>I2C_FLAG_LOSTARB</i>      | arbitration lost in master mode   |
| <i>I2C_FLAG_BERR</i>         | a bus error occurs indication a unexpected start or stop condition on I2C bus |
| <i>I2C_FLAG_ADDSEND</i>      | address is sent in master mode or received and matches in slave mode          |
| <i>I2C_FLAG_TFF</i>          | txframe fall flag   |
| <i>I2C_FLAG_TFR</i>          | txframe rise flag   |
| <i>I2C_FLAG_RFF</i>          | rxframe fall flag   |
| <i>I2C_FLAG_RFR</i>          | rxframe rise flag   |
| <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 3-295. Function i2c\_interrupt\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | i2c_interrupt_enable  |
| <b>Function prototype</b>    | void i2c_interrupt_enable(uint32_t i2c_periph, i2c_interrupt_enum interrupt); |
| <b>Function descriptions</b> | enable I2C interrupt  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>i2c_periph</b>            | I2C peripheral  |
| <i>I2Cx</i>                  | (x=0,1)   |



| Input parameter{in}   |                        |
|-----------------------|------------------------|
| <b>interrupt</b>      | interrupt type         |
| <i>I2C_INT_ERR</i>    | error interrupt        |
| <i>I2C_INT_EV</i>     | event interrupt        |
| <i>I2C_INT_BUF</i>    | buffer interrupt       |
| <i>I2C_INT_TFF</i>    | txframe fall interrupt |
| <i>I2C_INT_TFR</i>    | txframe rise interrupt |
| <i>I2C_INT_RFF</i>    | rxframe fall interrupt |
| <i>I2C_INT_RFR</i>    | rxframe rise interrupt |
| Output parameter{out} |                        |
| -                     | -                      |
| Return value          |                        |
| -                     | -                      |

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 3-296. 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>  | -  |
| Input parameter{in}          |  |
| <b>i2c_periph</b>            | I2C peripheral   |
| <i>I2Cx</i>                  | (x=0,1)  |
| Input parameter{in}          |  |
| <b>inttype</b>               | interrupt type   |
| <i>I2C_INT_ERR</i>           | error interrupt  |
| <i>I2C_INT_EV</i>            | event interrupt  |
| <i>I2C_INT_BUF</i>           | buffer interrupt   |
| <i>I2C_INT_TFF</i>           | txframe fall interrupt   |
| <i>I2C_INT_TFR</i>           | txframe rise interrupt   |
| <i>I2C_INT_RFF</i>           | rxframe fall interrupt   |
| <i>I2C_INT_RFR</i>           | rxframe rise interrupt   |
| Output parameter{out}        |  |
| -                            | -  |

| Return value |   |
|--------------|---|
| -            | - |

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 3-297. Function i2c\_interrupt\_flag\_get**

| Function name                      | i2c_interrupt_flag_get  |
|------------------------------------|---|
| Function prototype                 | FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph,<br>i2c_interrupt_flag_enum int_flag)     |
| Function descriptions              | get I2C interrupt flag  |
| Precondition                       | -   |
| The called functions               | -   |
| Input parameter{in}                |   |
| <b>i2c_periph</b>                  | I2C peripheral  |
| <i>I2Cx</i>                        | (x=0,1)   |
| Input parameter{in}                |   |
| <b>int_flag</b>                    | interrupt flag  |
| <i>I2C_INT_FLAG_SBSE<br/>ND</i>    | start condition sent out in master mode interrupt flag  |
| <i>I2C_INT_FLAG_ADDS<br/>END</i>   | address is sent in master mode or received and matches in slave mode<br>interrupt flag          |
| <i>I2C_INT_FLAG_BTC</i>            | byte transmission finishes  |
| <i>I2C_INT_FLAG_ADD10<br/>SEND</i> | header of 10-bit address is sent in master mode interrupt flag                                  |
| <i>I2C_INT_FLAG_STPD<br/>ET</i>    | stop condition detected in slave mode interrupt flag  |
| <i>I2C_INT_FLAG_RBNE</i>           | I2C_DATA is not Empty during receiving interrupt flag   |
| <i>I2C_INT_FLAG_TBE</i>            | I2C_DATA is empty during transmitting interrupt flag  |
| <i>I2C_INT_FLAG_BERR</i>           | a bus error occurs indication a unexpected start or stop condition on I2C<br>bus interrupt flag |
| <i>I2C_INT_FLAG_LOSTA<br/>RB</i>   | arbitration lost in master mode interrupt flag  |
| <i>I2C_INT_FLAG_AERR</i>           | acknowledge error interrupt flag  |
| <i>I2C_INT_FLAG_OUER<br/>R</i>     | over-run or under-run situation occurs in slave mode interrupt flag                             |
| <i>I2C_INT_FLAG_PECE<br/>RR</i>    | PEC error when receiving data interrupt flag  |

|                                |   |
|--------------------------------|---|
| <i>I2C_INT_FLAG_SMBT</i><br>O  | timeout signal in SMBus mode interrupt flag |
| <i>I2C_INT_FLAG_SMBA</i><br>LT | 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 3-298. Function i2c\_interrupt\_flag\_clear**

|  |  |
|--|--|
| <b>Function name</b>                   | i2c_interrupt_flag_clear   |
| <b>Function prototype</b>              | void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);        |
| <b>Function descriptions</b>           | clear I2C interrupt flag status  |
| <b>Precondition</b>                    | -  |
| <b>The called functions</b>            | -  |
| <b>Input parameter{in}</b>             |  |
| <b>i2c_periph</b>                      | I2C peripheral   |
| <i>I2Cx</i>                            | (x=0,1)  |
| <b>Input parameter{in}</b>             |  |
| <b>int_flag</b>                        | interrupt flag   |
| <i>I2C_INT_FLAG_ADDS</i><br><i>END</i> | address is sent in master mode or received and matches in slave mode interrupt flag          |
| <i>I2C_INT_FLAG_BERR</i>               | a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag |
| <i>I2C_INT_FLAG_LOSTA</i><br><i>RB</i> | arbitration lost in master mode interrupt flag   |
| <i>I2C_INT_FLAG_AERR</i>               | acknowledge error interrupt flag   |
| <i>I2C_INT_FLAG_OUER</i><br><i>R</i>   | over-run or under-run situation occurs in slave mode interrupt flag                          |
| <i>I2C_INT_FLAG_PECE</i><br><i>RR</i>  | PEC error when receiving data interrupt flag   |
| <i>I2C_INT_FLAG_SMBT</i><br>O          | timeout signal in SMBus mode interrupt flag  |

|   |                                   |
|---|-----------------------------------|
| <code>I2C_INT_FLAG_SMBA</code><br><code>LT</code> | SMBus Alert status interrupt flag |
| <code>I2C_INT_FLAG_TFF</code>                     | txframe fall interrupt flag       |
| <code>I2C_INT_FLAG_TFR</code>                     | txframe rise interrupt flag       |
| <code>I2C_INT_FLAG_RFF</code>                     | rxframe fall interrupt flag       |
| <code>I2C_INT_FLAG_RFR</code>                     | rxframe rise interrupt flag       |
| <b>Output parameter{out}</b>                      |                                   |
| -   | -                                 |
| <b>Return value</b>                               |                                   |
| -   | -                                 |

Example:

```
/* clear the acknowledge error interrupt flag */
```

```
i2c_interrupt_flag_clear(I2C0, I2C_INT_FLAG_AERR);
```

## 3.15. 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 3-299. NVIC Registers**

| Registers                        | Descriptions                                     |
|----------------------------------|--|
| <code>ISER<sup>(1)</sup></code>  | Interrupt Set Enable Register                    |
| <code>ICER<sup>(1)</sup></code>  | Interrupt Clear Enable Register                  |
| <code>ISPR<sup>(1)</sup></code>  | Interrupt Set Pending Register                   |
| <code>ICPR<sup>(1)</sup></code>  | Interrupt Clear Pending Register                 |
| <code>IABR<sup>(1)</sup></code>  | Interrupt Active bit Register                    |
| <code>IP<sup>(1)</sup></code>    | Interrupt Priority Register                      |
| <code>STIR<sup>(1)</sup></code>  | Software Trigger Interrupt Register              |
| <code>CPUID<sup>(2)</sup></code> | CPUID Base Register                              |
| <code>ICSR<sup>(2)</sup></code>  | Interrupt Control and State Register             |
| <code>VTOR<sup>(2)</sup></code>  | Vector Table Offset Register                     |
| <code>AIRCR<sup>(2)</sup></code> | Application Interrupt and Reset Control Register |
| <code>SCR<sup>(2)</sup></code>   | System Control Register                          |
| <code>CCR<sup>(2)</sup></code>   | Configuration Control Register                   |
| <code>SHP<sup>(2)</sup></code>   | System Handlers Priority Registers               |
| <code>SHCSR<sup>(2)</sup></code> | System Handler Control and State Register        |
| <code>CFSR<sup>(2)</sup></code>  | Configurable Fault Status Register               |
| <code>HFSR<sup>(2)</sup></code>  | HardFault Status Register                        |

| Registers            | Descriptions                        |
|----------------------|-------------------------------------|
| 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\_cm4h file

2. refer to the structure SCB\_Type, is defined in the core\_cm4. file

**Table 3-300. 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\_cm4h file

## 3.15.2. Descriptions of Peripheral functions

### Enum IRQn\_Type

**Table 3-301. IRQn\_Type**

| Member name        | Function description           |
|--------------------|--------------------------------|
| WWDGT_IRQn         | WWDGT interrupt                |
| LVD_IRQn           | LVD from EXTI interrupt        |
| TAMPER_IRQn        | Tamper interrupt               |
| RTC_IRQn           | RTC global interrupt           |
| FMC_IRQn           | FMC global interrupt           |
| RCU_CTC_IRQn       | RCU and CTC interrupts         |
| EXTI0_IRQn         | EXTI line0 interrupt           |
| EXTI1_IRQn         | EXTI line1 interrupt           |
| EXTI2_IRQn         | EXTI line2 interrupt           |
| EXTI3_IRQn         | EXTI line3 interrupt           |
| EXTI4_IRQn         | EXTI line4 interrupt           |
| DMA0_Channel0_IRQn | DMA0 channel0 global interrupt |
| DMA0_Channel1_IRQn | DMA0 channel1 global interrupt |
| DMA0_Channel2_IRQn | DMA0 channel2 global interrupt |
| DMA0_Channel3_IRQn | DMA0 channel3 global interrupt |

| Member name                 | Function description   |
|-----------------------------|--|
| DMA0_Channel4_IRQn          | DMA0 channel4 global interrupt                                       |
| DMA0_Channel5_IRQn          | DMA0 channel5 global interrupt                                       |
| DMA0_Channel6_IRQn          | DMA0 channel6 global interrupt                                       |
| ADC0_1_IRQn                 | ADC0 and ADC1 interrupts   |
| EXTI5_9_IRQn                | EXTI line[9:5] interrupts  |
| TIMER0_BRK_TIMER8_IRQn      | TIMER0 break and TIMER8 global interrupts                            |
| TIMER0_UP_TIMER9_IRQn       | TIMER0 update and TIMER9 global interrupts                           |
| TIMER0_TRG_CMT_TIMER10_IRQn | TIMER0 trigger and channel commutation and TIMER10 interrupts        |
| TIMER0_Channel_IRQn         | TIMER0 channel capture compare interrupt                             |
| TIMER1_IRQn                 | TIMER1 global interrupt  |
| TIMER2_IRQn                 | TIMER2 global interrupt  |
| TIMER3_IRQn                 | TIMER3 global interrupt  |
| I2C0_EV_IRQn                | I2C0 event interrupt   |
| I2C0_ER_IRQn                | I2C0 error interrupt   |
| I2C1_EV_IRQn                | I2C1 event interrupt   |
| I2C1_ER_IRQn                | I2C1 error interrupt   |
| SPI0_IRQn                   | SPI0 global interrupt  |
| SPI1_IRQn                   | SPI1 global interrupt  |
| USART0_IRQn                 | USART0 global interrupt  |
| USART1_IRQn                 | USART1 global interrupt  |
| USART2_IRQn                 | USART2 global interrupt  |
| EXTI10_15_IRQn              | EXTI line[15:10] interrupts  |
| RTC_Alarm_IRQn              | RTC alarm from EXTI interrupt  |
| USBFS_WKUP_IRQn             | USBFS wakeup from EXTI interrupt                                     |
| TIMER7_BRK_TIMER11_IRQn     | TIMER7 break and TIMER11 global interrupts                           |
| TIMER7_UP_TIMER12_IRQn      | TIMER7 update and TIMER12 global interrupts                          |
| TIMER7_TRG_CMT_TIMER13_IRQn | TIMER7 trigger and Channel commutation and TIMER13 global interrupts |
| TIMER7_Channel_IRQn         | TIMER7 channel capture compare interrupt                             |
| EXMC_IRQn                   | EXMC global interrupt  |
| TIMER4_IRQn                 | TIMER4 global interrupt  |
| SPI2_IRQn                   | SPI2 global interrupt  |
| UART3_IRQn                  | UART3 global interrupt   |
| UART4_IRQn                  | UART4 global interrupt   |
| TIMER5_IRQn                 | TIMER5 global interrupt  |
| TIMER6_IRQn                 | TIMER6 global interrupt  |
| DMA1_Channel0_IRQn          | DMA1 channel0 global interrupt                                       |
| DMA1_Channel1_IRQn          | DMA1 channel1 global interrupt                                       |
| DMA1_Channel2_IRQn          | DMA1 channel2 global interrupt                                       |
| DMA1_Channel3_IRQn          | DMA1 channel3 global interrupt                                       |

| Member name        | Function description           |
|--------------------|--------------------------------|
| DMA1_Channel4_IRQn | DMA1 channel4 global interrupt |

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

**Table 3-302. MISC firmware function**

| Function name                        | Function description                   |
|--------------------------------------|--|
| <code>nvic_priority_group_set</code> | set the priority group                 |
| <code>nvic_irq_enable</code>         | enable NVIC interrupt request          |
| <code>nvic_irq_disable</code>        | disable NVIC interrupt request         |
| <code>nvic_vector_table_set</code>   | set the NVIC vector table base address |
| <code>system_lowpower_set</code>     | set the state of the low power mode    |
| <code>system_lowpower_reset</code>   | reset the state of the low power mode  |
| <code>systick_clksource_set</code>   | set the systick clock source           |

## **`nvic_priority_group_set`**

The description of `nvic_priority_group_set` is shown as below:

**Table 3-303. Function `nvic_priority_group_set`**

| Function name                         | <code>nvic_priority_group_set</code>                               |
|---------------------------------------|--|
| Function prototype                    | <code>void nvic_priority_group_set(uint32_t nvic_prigroup);</code> |
| Function descriptions                 | set the priority group   |
| Precondition                          | -  |
| The called functions                  | -  |
| Input parameter{in}                   |  |
| <code>nvic_prigroup</code>            | priority group   |
| <code>NVIC_PRIGROUP_PR E0_SUB4</code> | 0 bits for pre-emption priority, 4 bits for subpriority            |
| <code>NVIC_PRIGROUP_PR E1_SUB3</code> | 1 bits for pre-emption priority, 3 bits for subpriority            |
| <code>NVIC_PRIGROUP_PR E2_SUB2</code> | 2 bits for pre-emption priority, 2 bits for subpriority            |
| <code>NVIC_PRIGROUP_PR E3_SUB1</code> | 3 bits for pre-emption priority, 1 bits for subpriority            |
| <code>NVIC_PRIGROUP_PR E4_SUB0</code> | 4 bits for pre-emption priority, 0 bits for subpriority            |
| Output parameter{out}                 |  |
| -                                     | -  |
| Return value                          |  |
| -                                     | -  |

Example:

```
/* priority group configuration , 0 bits for pre-emption priority, 4 bits for subpriority */
```

```
nvic_priority_group_set(NVIC_PRIGROUP_PRE0_SUB4);
```

## nvic\_irq\_enable

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

**Table 3-304. 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, uint8_t nvic_irq_sub_priority); |
| <b>Function descriptions</b> | enable NVIC interrupt request   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | nvic_priority_group_set   |
| <b>Input parameter{in}</b>   |   |
| nvic_irq                     | NVIC interrupt, refer to enum <a href="#">Table 3-301. IRQn_Type</a>                                  |
| <b>Input parameter{in}</b>   |   |
| nvic_irq_pre_priority        | the pre-emption priority needed to set  |
| <b>Input parameter{in}</b>   |   |
| nvic_irq_sub_priority        | the subpriority needed to set   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable window watchDog timer interrupt, pre-emption priority is 1, subpriority is 1 */
nvic_irq_enable(WWDGT_IRQn, 1, 1);
```

## nvic\_irq\_disable

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

**Table 3-305. Function nvic\_irq\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | nvic_irq_disable   |
| <b>Function prototype</b>    | void nvic_irq_disable(uint8_t nvic_irq);                             |
| <b>Function descriptions</b> | disable NVIC interrupt request                                       |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| nvic_irq                     | NVIC interrupt, refer to enum <a href="#">Table 3-301. IRQn_Type</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |



Example:

```
/* disable window watchDog timer interrupt */
nvic_irq_disable(WWDGT_IRQn);
```

## nvic\_vector\_table\_set

The description of nvic\_vector\_table\_set is shown as below:

**Table 3-306. Function nvic\_vector\_table\_set**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | nvic_vector_table_set   |
| <b>Function prototype</b>    | void nvic_vector_table_set(uint32_t nvic_vect_tab, uint32_t offset);  |
| <b>Function descriptions</b> | set the NVIC vector table base address                                |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>nvic_vect_tab</b>         | the RAM or FLASH base address   |
| <i>NVIC_VECTTAB_RAM</i>      | RAM base address  |
| <i>NVIC_VECTTAB_FLASH</i>    | Flash base address  |
| <i>H</i>                     |   |
| <b>Input parameter{in}</b>   |   |
| <b>offset</b>                | vector table offset (vector table start address= base address+offset) |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* set vector table address = NVIC_VECTTAB_FLASH +0x200 */
nvic_vector_table_set(NVIC_VECTTAB_FLASH,0x200);
```

## system\_lowpower\_set

The description of system\_lowpower\_set is shown as below:

**Table 3-307. Function system\_lowpower\_set**

|                               |  |
|-------------------------------|--|
| <b>Function name</b>          | system_lowpower_set  |
| <b>Function prototype</b>     | void system_lowpower_set(uint8_t lowpower_mode);                               |
| <b>Function descriptions</b>  | set the state of the low power mode  |
| <b>Precondition</b>           | -  |
| <b>The called functions</b>   | -  |
| <b>Input parameter{in}</b>    |  |
| <b>lowpower_mode</b>          | the low power mode state   |
| <i>SCB_LPM_SLEEP_EXIT_ISR</i> | if chose this para, the system always enter low power mode by exiting from ISR |

|   |   |
|---|---|
| <i>SCB_LPM_DEEPSLEEP</i><br><i>P</i>      | if chose this para, the system will enter the DEEPSLEEP mode                                  |
| <i>SCB_LPM_WAKE_BY_</i><br><i>ALL_INT</i> | if chose this para, the lowpower mode can be woke up by all the enable and disable interrupts |
| <b>Output parameter{out}</b>              |   |
| -   | -   |
| <b>Return value</b>                       |   |
| -   | -   |

Example:

```
/* the system always enter low power mode by exiting from ISR */
system_lowpower_set (SCB_LPM_SLEEP_EXIT_ISR);
```

## system\_lowpower\_reset

The description of system\_lowpower\_reset is shown as below:

**Table 3-308. Function system\_lowpower\_reset**

|   |  |
|---|--|
| <b>Function name</b>                      | system_lowpower_reset  |
| <b>Function prototype</b>                 | void system_lowpower_reset(uint8_t lowpower_mode);                                 |
| <b>Function descriptions</b>              | reset the state of the low power mode  |
| <b>Precondition</b>                       | -  |
| <b>The called functions</b>               | -  |
| <b>Input parameter{in}</b>                |  |
| <b>lowpower_mode</b>                      | the low power mode state   |
| <i>SCB_LPM_SLEEP_EXIT_ISR</i>             | if chose this para, the system will exit low power mode by exiting from ISR        |
| <i>SCB_LPM_DEEPSLEEP</i><br><i>P</i>      | if chose this para, the system will enter the SLEEP mode                           |
| <i>SCB_LPM_WAKE_BY_</i><br><i>ALL_INT</i> | if chose this para, the lowpower mode only can be woke up by the enable interrupts |
| <b>Output parameter{out}</b>              |  |
| -   | -  |
| <b>Return value</b>                       |  |
| -   | -  |

Example:

```
/* the system will exit low power mode by exiting from ISR */
system_lowpower_reset (SCB_LPM_SLEEP_EXIT_ISR);
```

## systick\_clksource\_set

The description of systick\_clksource\_set is shown as below:

**Table 3-309. Function systick\_clksource\_set**

|                                 |   |
|---------------------------------|---|
| <b>Function name</b>            | systick_clksource_set                                   |
| <b>Function prototype</b>       | void systick_clksource_set(uint32_t systick_clksource); |
| <b>Function descriptions</b>    | set the systick clock source                            |
| <b>Precondition</b>             | -   |
| <b>The called functions</b>     | -   |
| <b>Input parameter{in}</b>      |   |
| <b>systick_clksource</b>        | the systick clock source needed to choose               |
| SYSTICK_CLKSOURC<br>E_HCLK      | systick clock source is from HCLK                       |
| SYSTICK_CLKSOURC<br>E_HCLK_DIV8 | systick clock source is from HCLK/8                     |
| <b>Output parameter{out}</b>    |   |
| -                               | -   |
| <b>Return value</b>             |   |
| -                               | -   |

Example:

```
/* systick clock source is HCLK/8 */
```

```
systick_clksource_set (SYSTICK_CLKSOURCE_HCLK_DIV8);
```

## 3.16. 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 3-310. PMU Registers**

| Registers | Descriptions                |
|-----------|-----------------------------|
| PMU_CTL   | Control register            |
| PMU_CS    | Control and status register |

### 3.16.2. Descriptions of Peripheral functions

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

**Table 3-311. PMU firmware function**

| Function name            | Function description                  |
|--------------------------|---------------------------------------|
| pmu_deinit               | deinitialize the PMU                  |
| pmu_lvd_select           | select low voltage detector threshold |
| pmu_ldo_output_select    | select LDO output voltage             |
| pmu_lvd_disable          | disable PMU lvd                       |
| pmu_to_sleepmode         | PMU work at sleep mode                |
| pmu_to_deepsleepmode     | PMU work at deepsleep mode            |
| pmu_to_standbymode       | pmu work at standby mode              |
| pmu_wakeup_pin_enable    | enable wakeup pin                     |
| pmu_wakeup_pin_disable   | disable wakeup pin                    |
| pmu_backup_write_enable  | enable backup domain write            |
| pmu_backup_write_disable | disable backup domain write           |
| pmu_flag_get             | get flag state                        |
| pmu_flag_clear           | clear flag bit                        |

## pmu\_deinit

The description of pmu\_deinit is shown as below:

**Table 3-312. Function pmu\_deinit**

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

Example:

```
/* reset PMU */
pmu_deinit ();
```

## pmu\_lvd\_select

The description of pmu\_lvd\_select is shown as below:

**Table 3-313. Function pmu\_lvd\_select**

|                    |  |
|--------------------|--|
| Function name      | pmu_lvd_select                         |
| Function prototype | void pmu_lvd_select(uint32_t lvd_t_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 2.2V             |
| <i>PMU_LVDT_1</i>            | voltage threshold is 2.3V             |
| <i>PMU_LVDT_2</i>            | voltage threshold is 2.4V             |
| <i>PMU_LVDT_3</i>            | voltage threshold is 2.5V             |
| <i>PMU_LVDT_4</i>            | voltage threshold is 2.6V             |
| <i>PMU_LVDT_5</i>            | voltage threshold is 2.7V             |
| <i>PMU_LVDT_6</i>            | voltage threshold is 2.8V             |
| <i>PMU_LVDT_7</i>            | voltage threshold is 2.9V             |
| <b>Output parameter{out}</b> |                                       |
| -                            | -                                     |
| <b>Return value</b>          |                                       |
| -                            | -                                     |

Example:

```
/* select low voltage detector threshold as 2.9V */
```

```
pmu_lvd_select (PMU_LVDT_7);
```

## pmu\_ldo\_output\_select

The description of pmu\_ldo\_output\_select is shown as below:

**Table 3-314. 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> | internal voltage regulator (LDO) output voltage select |
| <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>         | output low voltage mode                                |
| <i>PMU_LDOVS_NORMAL</i>      | output normal voltage 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 3-315. Function pmu\_lvd\_disable**

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

Example:

```
/* disable PMU lvd */
```

```
pmu_lvd_disable ();
```

## pmu\_to\_sleepmode

The description of pmu\_to\_sleepmode is shown as below:

**Table 3-316. 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 3-317. 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 normal work 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 3-318. Function pmu\_to\_standbymode**

|                              |                                |
|------------------------------|--------------------------------|
| <b>Function name</b>         | pmu_to_standbymode             |
| <b>Function prototype</b>    | void pmu_to_standbymode(void); |
| <b>Function descriptions</b> | pmu work at standby mode       |
| <b>Precondition</b>          | -                              |
| <b>The called functions</b>  | -                              |
| <b>Input parameter{in}</b>   |                                |
| -                            | -                              |
| <b>Output parameter{out}</b> |                                |

|              |   |
|--------------|---|
| -            | - |
| Return value |   |
| -            | - |

Example:

```
/* PMU work at standby mode */
```

```
pmu_to_standbymode();
```

## pmu\_wakeup\_pin\_enable

The description of pmu\_wakeup\_pin\_enable is shown as below:

**Table 3-319. Function pmu\_wakeup\_pin\_enable**

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

Example:

```
/* enable wakeup pin */
```

```
pmu_wakeup_pin_enable ();
```

## pmu\_wakeup\_pin\_disable

The description of pmu\_wakeup\_pin\_disable is shown as below:

**Table 3-320. Function pmu\_wakeup\_pin\_disable**

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



|   |   |
|---|---|
| - | - |
|---|---|

Example:

```
/* disable wakeup pin */

pmu_wakeup_pin_disable ();
```

## pmu\_backup\_write\_enable

The description of pmu\_backup\_write\_enable is shown as below:

**Table 3-321. 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 3-322. 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\_get

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

**Table 3-323. Function pmu\_flag\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | pmu_flag_get                            |
| <b>Function prototype</b>    | FlagStatus pmu_flag_get(uint32_t flag); |
| <b>Function descriptions</b> | get flag state                          |
| <b>Precondition</b>          | -                                       |
| <b>The called functions</b>  | -                                       |
| <b>Input parameter{in}</b>   |   |
| <b>flag</b>                  | flag                                    |
| PMU_FLAG_WAKEUP              | wakeup flag                             |
| PMU_FLAG_STANDBY             | standby flag                            |
| PMU_FLAG_LVD                 | low voltage detector status flag        |
| <b>Output parameter{out}</b> |   |
| -                            | -                                       |
| <b>Return value</b>          |   |
| <b>FlagStatus</b>            | SET or RESET                            |

Example:

```
/* get flag state */  
  
FlagStatus status;  
  
status = pmu_flag_get (PMU_FLAG_WAKEUP);
```

## pmu\_flag\_clear

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

**Table 3-324. Function pmu\_flag\_clear**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | pmu_flag_clear                            |
| <b>Function prototype</b>    | void pmu_flag_clear(uint32_t flag_reset); |
| <b>Function descriptions</b> | clear flag bit                            |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>flag_reset</b>            | flag                                      |
| PMU_FLAG_RESET_WAKEUP        | reset wakeup flag                         |

|                                    |                    |
|------------------------------------|--------------------|
| <i>PMU_FLAG_RESET_S<br/>TANDBY</i> | reset standby flag |
| <b>Output parameter{out}</b>       |                    |
| -                                  | -                  |
| <b>Return value</b>                |                    |
| -                                  | -                  |

Example:

```
/* clear flag bit */
```

```
pmu_flag_clear (PMU_FLAG_RESET_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 3-325. RCU Registers**

| Registers      | Descriptions                        |
|----------------|-------------------------------------|
| RCU_CTL        | Control register                    |
| RCU_CFG0       | Clock configuration register 0      |
| RCU_INT        | Clock interrupt register            |
| RCU_APB2RST    | APB2 reset register                 |
| RCU_APB1RST    | APB1 reset register                 |
| RCU_AHBEN      | AHB enable register                 |
| RCU_APB2EN     | APB2 enable register                |
| RCU_APB1EN     | APB1 enable register                |
| RCU_BDCTL      | Backup domain control register      |
| RCU_RSTSCK     | Reset source/clock register         |
| RCU_AHBRST     | AHB reset register                  |
| RCU_CFG1       | Clock configuration register 1      |
| RCU_DSV        | Deep-sleep mode voltage register    |
| RCU_ADDCTL     | Additional clock control register   |
| RCU_ADDINT     | Additional clock interrupt register |
| RCU_ADDAPB1RST | APB1 additional reset register      |
| RCU_ADDAPB1EN  | APB1 additional enable register     |

## 3.17.2. Descriptions of Peripheral functions

RCU firmware function are listed in the table shown as below:

**Table 3-326. RCU firmware function**

| Function name                     | Function description  |
|-----------------------------------|---|
| rcu_deinit                        | deinitialize the RCU  |
| rcu_periph_clock_enable           | enable the peripherals clock  |
| rcu_periph_clock_disable          | disable the peripherals clock   |
| rcu_periph_clock_sleep_enable     | enable the peripherals clock when in sleep mode                                 |
| rcu_periph_clock_sleep_disable    | disable the peripherals clock when in sleep mode                                |
| rcu_periph_reset_enable           | enable the peripherals reset  |
| rcu_periph_reset_disable          | disable the peripheral reset  |
| rcu_bkp_reset_enable              | enable the BKP domain reset   |
| rcu_bkp_reset_disable             | disable the BKP domain reset  |
| rcu_system_clock_source_config    | configure the system clock source   |
| rcu_system_clock_source_get       | get the system clock source   |
| rcu_ahb_clock_config              | configure the AHB clock prescaler selection                                     |
| rcu_apb1_clock_config             | configure the APB1 clock prescaler selection                                    |
| rcu_apb2_clock_config             | configure the APB2 clock prescaler selection                                    |
| rcu_ckout0_config                 | configure the CK_OUT0 clock source  |
| rcu_pll_config                    | configure the main PLL clock  |
| rcu_pllpreselect_config           | configure the PLL clock source preselection                                     |
| rcu_predv0_config                 | configure the PREDV0 division factor  |
| rcu_predv1_config                 | configure the PREDV1 division factor  |
| rcu_pll1_config                   | configure the PLL1 clock  |
| rcu_pll2_config                   | configure the PLL2 clock  |
| rcu_adc_clock_config              | configure the ADC prescaler factor  |
| rcu_usb_clock_config              | configure the USB prescaler factor  |
| rcu_rtc_clock_config              | configure the RTC clock source selection  |
| rcu_i2s1_clock_config             | configure the I2S1 clock source selection                                       |
| rcu_i2s2_clock_config             | configure the I2S2 clock source selection                                       |
| rcu_ck48m_clock_config            | configure the CK48M clock selection   |
| rcu_flag_get                      | get the clock stabilization and peripheral reset flags                          |
| rcu_all_reset_flag_clear          | clear all the reset flag  |
| rcu_interrupt_flag_get            | get the clock stabilization interrupt and ckm flags                             |
| rcu_interrupt_flag_clear          | clear the interrupt flags   |
| rcu_interrupt_enable              | enable the stabilization interrupt  |
| rcu_interrupt_disable             | disable the stabilization interrupt   |
| rcu_lxtal_drive_capability_config | configure the LXTAL drive capability  |
| rcu_osci_stab_wait                | wait for oscillator stabilization flags is SET or oscillator startup is timeout |
| rcu_osci_on                       | turn on the oscillator  |

| Function name                   | Function description                      |
|---------------------------------|---|
| 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_deepsleep_voltage_set       | set the deep-sleep mode voltage value     |
| rcu_clock_freq_get              | get the system clock, bus clock frequency |

## Enum rcu\_periph\_enum

Table 3-327. rcu\_periph\_enum

| enum name   | Function description     |
|-------------|--------------------------|
| RCU_DMA0    | DMA0 clock               |
| RCU_DMA1    | DMA1 clock               |
| RCU_CRC     | CRC clock                |
| RCU_EXMC    | EXMC clock               |
| RCU_USBFS   | USBFS clock              |
| RCU_TIMER1  | TIMER1 clock             |
| RCU_TIMER2  | TIMER2 clock             |
| RCU_TIMER3  | TIMER3 clock             |
| RCU_TIMER4  | TIMER4 clock             |
| RCU_TIMER5  | TIMER5 clock             |
| RCU_TIMER6  | TIMER6 clock             |
| RCU_TIMER11 | TIMER11 clock            |
| RCU_TIMER12 | TIMER12 clock            |
| RCU_TIMER13 | TIMER13 clock            |
| RCU_WWDGT   | WWDGT clock              |
| RCU_SPI1    | SPI1 clock               |
| RCU_SPI2    | SPI2 clock               |
| RCU_USART1  | USART1 clock             |
| RCU_USART2  | USART2 clock             |
| RCU_UART3   | UART3 clock              |
| RCU_UART4   | UART4 clock              |
| RCU_I2C0    | I2C0 clock               |
| RCU_I2C1    | I2C1 clock               |
| RCU_BKPI    | BKPI clock               |
| RCU_PMU     | PMU clock                |
| RCU_DAC     | DAC clock                |
| RCU_RTC     | RTC clock                |
| RCU_CTC     | CTC clock                |
| RCU_AF      | alternate function clock |

| enum name   | Function description |
|-------------|----------------------|
| RCU_GPIOA   | GPIOA clock          |
| RCU_GPIOB   | GPIOB clock          |
| RCU_GPIOC   | GPIOC clock          |
| RCU_GPIOD   | GPIOD clock          |
| RCU_GPIOE   | GPIOE clock          |
| RCU_ADC0    | ADC0 clock           |
| RCU_ADC1    | ADC1 clock           |
| RCU_TIMER0  | TIMER0 clock         |
| RCU_SPI0    | SPI0 clock           |
| RCU_TIMER7  | TIMER7 clock         |
| RCU_USART0  | USART0 clock         |
| RCU_ADC2    | ADC2 clock           |
| RCU_TIMER8  | TIMER8 clock         |
| RCU_TIMER9  | TIMER9 clock         |
| RCU_TIMER10 | TIMER10 clock        |

## Enum rcu\_periph\_sleep\_enum

Table 3-328. rcu\_periph\_sleep\_enum

| enum name    | Function description       |
|--------------|----------------------------|
| RCU_SRAM_SLP | SRAM clock when sleep mode |
| RCU_FMC_SLP  | FMC clock when sleep mode  |

## Enum rcu\_periph\_reset\_enum

Table 3-329. rcu\_periph\_reset\_enum

| enum name      | Function description |
|----------------|----------------------|
| RCU_USBFIRST   | USBFS clock reset    |
| RCU_TIMER1RST  | TIMER1 clock reset   |
| RCU_TIMER2RST  | TIMER2 clock reset   |
| RCU_TIMER3RST  | TIMER3 clock reset   |
| RCU_TIMER4RST  | TIMER4 clock reset   |
| RCU_TIMER5RST  | TIMER5 clock reset   |
| RCU_TIMER6RST  | TIMER6 clock reset   |
| RCU_TIMER11RST | TIMER11 clock reset  |
| RCU_TIMER12RST | TIMER12 clock reset  |
| RCU_TIMER13RST | TIMER13 clock reset  |
| RCU_WWDGTRST   | WWDGT clock reset    |
| RCU_SPI1RST    | SPI1 clock reset     |
| RCU_SPI2RST    | SPI2 clock reset     |
| RCU_USART1RST  | USART1 clock reset   |
| RCU_USART2RST  | USART2 clock reset   |

| enum name      | Function description           |
|----------------|--------------------------------|
| RCU_UART3RST   | UART3 clock reset              |
| RCU_UART4RST   | UART4 clock reset              |
| RCU_I2C0RST    | I2C0 clock reset               |
| RCU_I2C1RST    | I2C1 clock reset               |
| RCU_USBD RST   | USB D clock reset              |
| RCU_I2C2RST    | I2C2 clock reset               |
| RCU_BKPIRST    | BKPI clock reset               |
| RCU_PMURST     | PMU clock reset                |
| RCU_DACRST     | DAC clock reset                |
| RCU_CTCRST     | CTC clock reset                |
| RCU_AFRST      | alternate function clock reset |
| RCU_GPIOARST   | GPIOA clock reset              |
| RCU_GPIOBRST   | GPIOB clock reset              |
| RCU_GPIOCRST   | GPIOC clock reset              |
| RCU_GPIODRST   | GPIO D clock reset             |
| RCU_GPIOERST   | GPIOE clock reset              |
| RCU_ADC0RST    | ADC0 clock reset               |
| RCU_ADC1RST    | ADC1 clock reset               |
| RCU_TIMER0RST  | TIMER0 clock reset             |
| RCU_SPI0RST    | SPI0 clock reset               |
| RCU_TIMER7RST  | TIMER7 clock reset             |
| RCU_USART0RST  | USART0 clock reset             |
| RCU_TIMER8RST  | TIMER8 clock reset             |
| RCU_TIMER9RST  | TIMER9 clock reset             |
| RCU_TIMER10RST | TIMER10 clock reset            |

### Enum rcu\_flag\_enum

Table 3-330. rcu\_flag\_enum

| enum name              | Function description       |
|------------------------|----------------------------|
| RCU_FLAG_IRC8MST<br>B  | IRC8M stabilization flags  |
| RCU_FLAG_HXTALST<br>B  | HXTAL stabilization flags  |
| RCU_FLAG_PLLSTB        | PLL stabilization flags    |
| RCU_FLAG_PLL1STB       | PLL1 stabilization flags   |
| RCU_FLAG_PLL2STB       | PLL2 stabilization flags   |
| RCU_FLAG_LXTALST<br>B  | LXTAL stabilization flags  |
| RCU_FLAG_IRC40KST<br>B | IRC40K stabilization flags |
| RCU_FLAG_IRC48MS       | IRC48M stabilization flags |

| enum name             | Function description     |
|-----------------------|--------------------------|
| TB                    |                          |
| RCU_FLAG_EPRST        | external PIN reset flags |
| RCU_FLAG_PORRST       | power reset flags        |
| RCU_FLAG_SWRST        | software reset flags     |
| RCU_FLAG_FWDGTR<br>ST | FWDGT reset flags        |
| RCU_FLAG_WWDGTR<br>ST | WWDGT reset flags        |
| RCU_FLAG_LPRST        | low-power reset flags    |

### Enum rcu\_int\_flag\_enum

Table 3-331. rcu\_int\_flag\_enum

| enum name                  | Function description                |
|----------------------------|-------------------------------------|
| RCU_INT_FLAG_IRC4<br>0KSTB | IRC40K stabilization interrupt flag |
| RCU_INT_FLAG_LXTA<br>LSTB  | LXTAL stabilization interrupt flag  |
| RCU_INT_FLAG_IRC8<br>MSTB  | IRC8M stabilization interrupt flag  |
| RCU_INT_FLAG_HXTA<br>LSTB  | HXTAL stabilization interrupt flag  |
| RCU_INT_FLAG_PLLS<br>TB    | PLL stabilization interrupt flag    |
| RCU_INT_FLAG_PLL1<br>STB   | PLL1 stabilization interrupt flag   |
| RCU_INT_FLAG_PLL2<br>STB   | PLL2 stabilization interrupt flag   |
| RCU_INT_FLAG_CKM           | HXTAL clock stuck interrupt flag    |
| RCU_INT_FLAG_IRC4<br>8MSTB | IRC48M stabilization interrupt flag |

### Enum rcu\_int\_flag\_clear\_enum

Table 3-332. rcu\_int\_flag\_clear\_enum

| enum name                      | Function description                       |
|--------------------------------|--|
| RCU_INT_FLAG_IRC4<br>0KSTB_CLR | IRC40K stabilization interrupt flags clear |
| RCU_INT_FLAG_LXTA<br>LSTB_CLR  | LXTAL stabilization interrupt flags clear  |
| RCU_INT_FLAG_IRC8<br>MSTB_CLR  | IRC8M stabilization interrupt flags clear  |
| RCU_INT_FLAG_HXTA              | HXTAL stabilization interrupt flags clear  |



| enum name                      | Function description  |
|--------------------------------|---|
| LSTB_CLR                       |   |
| RCU_INT_FLAG_PLLS<br>TB_CLR    | PLL stabilization interrupt flags clear                     |
| RCU_INT_FLAG_PLL1<br>STB_CLR   | PLL1 stabilization interrupt flags clear                    |
| RCU_INT_FLAG_PLL2<br>STB_CLR   | PLL2 stabilization interrupt flags clear                    |
| RCU_INT_FLAG_CKM<br>_CLR       | CKM interrupt flags clear                                   |
| RCU_INT_FLAG_IRC4<br>8MSTB_CLR | internal 48 MHz RC oscillator stabilization interrupt clear |

## Enum rcu\_int\_enum

**Table 3-333. rcu\_int\_enum**

| enum name         | Function description                                  |
|-------------------|---|
| RCU_INT_IRC40KSTB | IRC40K stabilization interrupt                        |
| RCU_INT_LXTALSTB  | LXTAL stabilization interrupt                         |
| RCU_INT_IRC8MSTB  | IRC8M stabilization interrupt                         |
| RCU_INT_HXTALSTB  | HXTAL stabilization interrupt                         |
| RCU_INT_PLLSTB    | PLL stabilization interrupt                           |
| RCU_INT_PLL1STB   | PLL1 stabilization interrupt                          |
| RCU_INT_PLL2STB   | PLL2 stabilization interrupt                          |
| RCU_INT_IRC48MSTB | internal 48 MHz RC oscillator stabilization interrupt |

## Enum rcu\_osci\_type\_enum

**Table 3-334. rcu\_osci\_type\_enum**

| enum name   | Function description |
|-------------|----------------------|
| RCU_HXTAL   | HXTAL                |
| RCU_LXTAL   | LXTAL                |
| RCU_IRC8M   | IRC8M                |
| RCU_IRC48M  | IRC48M               |
| RCU_IRC40K  | IRC40K               |
| RCU_PLL_CK  | PLL                  |
| RCU_PLL1_CK | PLL1                 |
| RCU_PLL2_CK | PLL2                 |

## Enum rcu\_clock\_freq\_enum

**Table 3-335. rcu\_clock\_freq\_enum**

| enum name | Function description |
|-----------|----------------------|
| CK_SYS    | system clock         |

| enum name | Function description |
|-----------|----------------------|
| CK_AHB    | AHB clock            |
| CK_APB1   | APB1 clock           |
| CK_APB2   | APB2 clock           |

## rcu\_deinit

The description of rcu\_deinit is shown as below:

**Table 3-336. Function rcu\_deinit**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_deinit   |
| <b>Function prototype</b>    | void rcu_deinit(void);   |
| <b>Function descriptions</b> | deinitialize the RCU, reset the value of all RCU registers into initial values |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | rcu_osc_stab_wait  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* reset RCU */
rcu_deinit();
```

## rcu\_periph\_clock\_enable

The description of rcu\_periph\_clock\_enable is shown as below:

**Table 3-337. 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>   |  |
| periph                       | RCU peripherals, refer to <a href="#">Table 3-327. rcu_periph_enum</a> |
| <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 3-338. Function rcu\_periph\_clock\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_periph_clock_disable   |
| <b>Function prototype</b>    | void rcu_periph_clock_disable(rcu_periph_enum periph);                 |
| <b>Function descriptions</b> | disable the peripherals clock  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>periph</b>                | RCU peripherals, refer to <a href="#">Table 3-327. rcu_periph_enum</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

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 3-339. Function rcu\_periph\_clock\_sleep\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_periph_clock_sleep_enable  |
| <b>Function prototype</b>    | void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);            |
| <b>Function descriptions</b> | enable the peripherals clock when in sleep mode                              |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>periph</b>                | RCU peripherals, refer to <a href="#">Table 3-328. rcu_periph_sleep_enum</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable the FMC clock when in sleep mode */
```

```
rcu_periph_clock_sleep_enable(RCU_FMC_SLP);
```

## rcu\_periph\_clock\_sleep\_disable

The description of rcu\_periph\_clock\_sleep\_disable is shown as below:

**Table 3-340. Function rcu\_periph\_clock\_sleep\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_periph_clock_sleep_disable   |
| <b>Function prototype</b>    | void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph);           |
| <b>Function descriptions</b> | disable the peripherals clock when in sleep mode                             |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>periph</b>                | RCU peripherals, refer to <a href="#">Table 3-328. rcu_periph_sleep_enum</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable the FMC clock when in sleep mode */
```

```
rcu_periph_clock_sleep_disable(RCU_FMC_SLP);
```

## rcu\_periph\_reset\_enable

The description of rcu\_periph\_reset\_enable is shown as below:

**Table 3-341. Function rcu\_periph\_reset\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_periph_reset_enable  |
| <b>Function prototype</b>    | void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);                  |
| <b>Function descriptions</b> | enable the peripherals reset   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>periph_reset</b>          | RCU peripherals reset, refer to <a href="#">Table 3-329. rcu_periph_reset_enum</a> |
| <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 3-342. Function rcu\_periph\_reset\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_periph_reset_disable   |
| <b>Function prototype</b>    | void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);                 |
| <b>Function descriptions</b> | disable the peripheral reset   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>periph_reset</b>          | RCU peripherals reset, refer to <a href="#">Table 3-329. rcu_periph_reset_enum</a> |
| <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 3-343. 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 3-344. 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 3-345. Function rcu\_system\_clock\_source\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rcu_system_clock_source_config                        |
| <b>Function prototype</b>    | void rcu_system_clock_source_config(uint32_t ck_sys); |
| <b>Function descriptions</b> | configure the system clock source                     |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>ck_sys</b>                | system clock source select                            |
| <i>RCU_CKSYSSRC_IRC8M</i>    | select CK_IRC8M as the CK_SYS source                  |
| <i>RCU_CKSYSSRC_HXTAL</i>    | select CK_HXTAL as the CK_SYS source                  |
| <i>RCU_CKSYSSRC_PLL</i>      | select CK_PLL as the CK_SYS source                    |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* configure the CK_HXTAL as the CK_SYS source */
```

```
rcu_system_clock_source_config(RCU_CKSYSSRC_HXTAL);
```

## rcu\_system\_clock\_source\_get

The description of rcu\_system\_clock\_source\_get is shown as below:

**Table 3-346. 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>          |   |
| uint32_t                     | 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 3-347. 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>   |  |
| ck_ahb                       | AHB clock prescaler selection                            |
| RCU_AHB_CKSYS_DIVx           | 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 3-348. Function rcu\_apb1\_clock\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rcu_apb1_clock_config                         |
| <b>Function prototype</b>    | void rcu_apb1_clock_config(uint32_t ck_apb1); |
| <b>Function descriptions</b> | configure the APB1 clock prescaler selection  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>ck_apb1</b>               | APB1 clock prescaler selection                |
| <i>RCU_APB1_CKAHB_DIV1</i>   | select CK_AHB as CK_APB1                      |
| <i>RCU_APB1_CKAHB_DIV2</i>   | select CK_AHB/2 as CK_APB1                    |
| <i>RCU_APB1_CKAHB_DIV4</i>   | select CK_AHB/4 as CK_APB1                    |
| <i>RCU_APB1_CKAHB_DIV8</i>   | select CK_AHB/8 as CK_APB1                    |
| <i>RCU_APB1_CKAHB_DIV16</i>  | select CK_AHB/16 as CK_APB1                   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <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 3-349. Function rcu\_apb2\_clock\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rcu_apb2_clock_config                         |
| <b>Function prototype</b>    | void rcu_apb2_clock_config(uint32_t ck_apb2); |
| <b>Function descriptions</b> | configure the APB2 clock prescaler selection  |



|  |                                |
|--|--------------------------------|
| <b>Precondition</b>                    | -                              |
| <b>The called functions</b>            | -                              |
| <b>Input parameter{in}</b>             |                                |
| <b>ck_apb2</b>                         | APB2 clock prescaler selection |
| <i>RCU_APB2_CKAHB_D</i><br><i>IV1</i>  | select CK_AHB as CK_APB2       |
| <i>RCU_APB2_CKAHB_D</i><br><i>IV2</i>  | select CK_AHB/2 as CK_APB2     |
| <i>RCU_APB2_CKAHB_D</i><br><i>IV4</i>  | select CK_AHB/4 as CK_APB2     |
| <i>RCU_APB2_CKAHB_D</i><br><i>IV8</i>  | select CK_AHB/8 as CK_APB2     |
| <i>RCU_APB2_CKAHB_D</i><br><i>IV16</i> | select CK_AHB/16 as CK_APB2    |
| <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\_ckout0\_config

The description of rcu\_ckout0\_config is shown as below:

**Table 3-350. Function rcu\_ckout0\_config**

|                                       |  |
|---------------------------------------|--|
| <b>Function name</b>                  | rcu_ckout0_config                              |
| <b>Function prototype</b>             | void rcu_ckout0_config(uint32_t ckout0_src);   |
| <b>Function descriptions</b>          | configure the CK_OUT0 clock source             |
| <b>Precondition</b>                   | -  |
| <b>The called functions</b>           | -  |
| <b>Input parameter{in}</b>            |  |
| <b>ckout0_src</b>                     | CK_OUT0 clock source selection                 |
| <i>RCU_CKOUT0SRC_N</i><br><i>ONE</i>  | no clock selected                              |
| <i>RCU_CKOUT0SRC_C</i><br><i>KSYS</i> | select system clock CK_SYS                     |
| <i>RCU_CKOUT0SRC_IR</i><br><i>C8M</i> | select high speed 8M internal oscillator clock |
| <i>RCU_CKOUT0SRC_H</i><br><i>XTAL</i> | select HXTAL                                   |

|   |                            |
|---|----------------------------|
| <i>RCU_CKOUT0SRC_C</i><br><i>KPLL_DIV2</i>  | select (CK_PLL / 2) clock  |
| <i>RCU_CKOUT0SRC_C</i><br><i>KPLL1</i>      | select CK_PLL1 clock       |
| <i>RCU_CKOUT0SRC_C</i><br><i>KPLL2_DIV2</i> | select (CK_PLL2 / 2) clock |
| <i>RCU_CKOUT0SRC_C</i><br><i>KPLL2</i>      | select CK_PLL2 clock       |
| <i>RCU_CKOUT0SRC_IR</i><br><i>C48M</i>      | select IRC48M clock        |
| <i>RCU_CKOUT0SRC_IR</i><br><i>C8M_DIV8</i>  | select (IRC48M / 8) clock  |
| <b>Output parameter{out}</b>                |                            |
| -   | -                          |
| <b>Return value</b>                         |                            |
| -   | -                          |

Example:

```
/* configure the HXTAL as CK_OUT0 clock source */
```

```
rcu_ckout0_config(RCU_CKOUT0SRC_HXTAL);
```

### rcu\_pll\_config

The description of rcu\_pll\_config is shown as below:

**Table 3-351. Function rcu\_pll\_config**

|   |  |
|---|--|
| <b>Function name</b>                      | rcu_pll_config   |
| <b>Function prototype</b>                 | void rcu_pll_config(uint32_t pll_src, uint32_t pll_mul); |
| <b>Function descriptions</b>              | configure the main PLL clock                             |
| <b>Precondition</b>                       | -  |
| <b>The called functions</b>               | -  |
| <b>Input parameter{in}</b>                |  |
| <b>pll_src</b>                            | PLL clock source selection                               |
| <i>RCU_PLLSRC_IRC8M</i><br><i>_DIV2</i>   | IRC8M/2 clock is selected as source clock of PLL         |
| <i>RCU_PLLSRC_HXTAL</i><br><i>_IRC48M</i> | HXTAL or IRC48M is selected as source clock of PLL       |
| <b>Input parameter{in}</b>                |  |
| <b>pll_mul</b>                            | PLL clock multiplication factor                          |
| <i>RCU_PLL_MULx</i>                       | PLL clock * x (x = 2..14, 6.5, 16..31)                   |
| <b>Output parameter{out}</b>              |  |
| -   | -  |
| <b>Return value</b>                       |  |

|   |   |
|---|---|
| - | - |
|---|---|

Example:

```
/* configure the PLL */
```

```
rcu_pll_config(RCU_PLLSRC_HXTAL, RCU_PLL_MUL10);
```

## rcu\_pllpresel\_config

The description of rcu\_pllpresel\_config is shown as below:

**Table 3-352. Function rcu\_pllpresel\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rcu_pllpresel_config                            |
| <b>Function prototype</b>    | void rcu_pllpresel_config(uint32_t pll_presel); |
| <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 preselection                   |
| <i>RCU_PLLPRESRC_HXTAL</i>   | HXTAL selected as PREDV0 source clock           |
| <i>RCU_PLLPRESRC_IRC48M</i>  | CK_PLL selected as PREDV0 input source clock    |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* configure the PLL clock source preselection */
```

```
rcu_pllpresel_config (RCU_PLLPRESRC_HXTAL);
```

## rcu\_predv0\_config

The description of rcu\_predv0\_config is shown as below:

**Table 3-353. Function rcu\_predv0\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_predv0_config  |
| <b>Function prototype</b>    | void rcu_predv0_config(uint32_t predv0_source, uint32_t predv0_div); |
| <b>Function descriptions</b> | configure the PREDV0 division factor                                 |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>predv0_source</b>         | PREDV0 input clock source selection                                  |
| <i>RCU_PREDV0SRC_HXTAL</i>   | select HXTAL or IRC48M as PREDV0 input source clock                  |

|                              |  |
|------------------------------|--|
| <i>XTAL_IRC48M</i>           |  |
| <i>RCU_PREDV0SRC_CKPLL1</i>  | select CK_PLL1 as PREDV0 input source clock      |
| <b>Input parameter{in}</b>   |  |
| <b>predv0_div</b>            | PREDV0 division factor                           |
| <i>RCU_PREDV0_DIVx</i>       | PREDV0 input source clock is divided x (x=1..16) |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure the PREDV0 division factor */
rcu_predv0_config(RCU_PREDV0SRC_HXTAL_IRC48M, RCU_PREDV0_DIV4);
```

## rcu\_predv1\_config

The description of rcu\_predv1\_config is shown as below:

**Table 3-354. Function rcu\_predv1\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_predv1_config                                |
| <b>Function prototype</b>    | void rcu_predv1_config(uint32_t predv1_div);     |
| <b>Function descriptions</b> | configure the PREDV1 division factor             |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>predv1_div</b>            | PREDV1 division factor                           |
| <i>RCU_PREDV1_DIVx</i>       | PREDV1 input source clock is divided x (x=1..16) |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure the PREDV1 division factor */
rcu_predv1_config(RCU_PREDV1_DIV8);
```

## rcu\_pll1\_config

The description of rcu\_pll1\_config is shown as below:

**Table 3-355. Function rcu\_pll1\_config**

|                           |   |
|---------------------------|---|
| <b>Function name</b>      | rcu_pll1_config                         |
| <b>Function prototype</b> | void rcu_pll1_config(uint32_t pll_mul); |

|                              |                                 |
|------------------------------|---------------------------------|
| <b>Function descriptions</b> | configure the PLL1 clock        |
| <b>Precondition</b>          | -                               |
| <b>The called functions</b>  | -                               |
| <b>Input parameter{in}</b>   |                                 |
| <b>pll_mul</b>               | PLL clock multiplication factor |
| <i>RCU_PLL1_MULx</i>         | PLL1 clock * x, (x = 8..16, 20) |
| <b>Output parameter{out}</b> |                                 |
| -                            | -                               |
| <b>Return value</b>          |                                 |
| -                            | -                               |

Example:

```
/* configure the PLL1 clock */
rcu_pll1_config(RCU_PLL1_MUL8);
```

## rcu\_pll2\_config

The description of rcu\_pll2\_config is shown as below:

**Table 3-356. Function rcu\_pll2\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_pll2_config                        |
| <b>Function prototype</b>    | void rcu_pll2_config(uint32_t pll_mul) |
| <b>Function descriptions</b> | configure the PLL2 clock               |
| <b>Precondition</b>          | -                                      |
| <b>The called functions</b>  | -                                      |
| <b>Input parameter{in}</b>   |  |
| <b>pll_mul</b>               | PLL clock multiplication factor        |
| <i>RCU_PLL2_MULx</i>         | PLL2 clock * x, (x = 8..16, 20)        |
| <b>Output parameter{out}</b> |  |
| -                            | -                                      |
| <b>Return value</b>          |  |
| -                            | -                                      |

Example:

```
/* configure the PLL2 clock */
rcu_pll2_config(RCU_PLL2_MUL8);
```

## rcu\_adc\_clock\_config

The description of rcu\_adc\_clock\_config is shown as below:

**Table 3-357. Function rcu\_adc\_clock\_config**

|                           |  |
|---------------------------|--|
| <b>Function name</b>      | rcu_adc_clock_config                         |
| <b>Function prototype</b> | void rcu_adc_clock_config(uint32_t adc_psc); |

|                               |                                    |
|-------------------------------|------------------------------------|
| <b>Function descriptions</b>  | configure the ADC prescaler factor |
| <b>Precondition</b>           | -                                  |
| <b>The called functions</b>   | -                                  |
| <b>Input parameter{in}</b>    |                                    |
| <b>adc_psc</b>                | ADC prescaler factor               |
| <i>RCU_CKADC_CKAPB2_DIV2</i>  | $CK\_ADC = CK\_APB2 / 2$           |
| <i>RCU_CKADC_CKAPB2_DIV4</i>  | $CK\_ADC = CK\_APB2 / 4$           |
| <i>RCU_CKADC_CKAPB2_DIV6</i>  | $CK\_ADC = CK\_APB2 / 6$           |
| <i>RCU_CKADC_CKAPB2_DIV8</i>  | $CK\_ADC = CK\_APB2 / 8$           |
| <i>RCU_CKADC_CKAPB2_DIV12</i> | $CK\_ADC = CK\_APB2 / 12$          |
| <i>RCU_CKADC_CKAPB2_DIV16</i> | $CK\_ADC = CK\_APB2 / 16$          |
| <i>RCU_CKADC_CKAHB_DIV3</i>   | $CK\_ADC = CK\_AHB / 3$            |
| <i>RCU_CKADC_CKAHB_DIV5</i>   | $CK\_ADC = CK\_AHB / 5$            |
| <i>RCU_CKADC_CKAHB_DIV7</i>   | $CK\_ADC = CK\_AHB / 7$            |
| <i>RCU_CKADC_CKAHB_DIV9</i>   | $CK\_ADC = CK\_AHB / 9$            |
| <b>Output parameter{out}</b>  |                                    |
| -                             | -                                  |
| <b>Return value</b>           |                                    |
| -                             | -                                  |

Example:

```
/* configure the ADC prescaler factor */
```

```
rcu_adc_clock_config(RCU_CKADC_CKAPB2_DIV8);
```

## rcu\_usb\_clock\_config

The description of rcu\_usb\_clock\_config is shown as below:

**Table 3-358. Function rcu\_usb\_clock\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_usb_clock_config                         |
| <b>Function prototype</b>    | void rcu_usb_clock_config(uint32_t usb_psc); |
| <b>Function descriptions</b> | configure the USB prescaler factor           |
| <b>Precondition</b>          | -  |

|                               |                             |
|-------------------------------|-----------------------------|
| <b>The called functions</b>   | -                           |
| <b>Input parameter{in}</b>    |                             |
| <b>usb_psc</b>                | USB prescaler factor        |
| <i>RCU_CKUSB_CKPLL_DIV1_5</i> | $CK\_USBFS = CK\_PLL / 1.5$ |
| <i>RCU_CKUSB_CKPLL_DIV1</i>   | $CK\_USBFS = CK\_PLL / 1$   |
| <i>RCU_CKUSB_CKPLL_DIV2_5</i> | $CK\_USBFS = CK\_PLL / 2.5$ |
| <i>RCU_CKUSB_CKPLL_DIV2</i>   | $CK\_USBFS = CK\_PLL / 2$   |
| <i>RCU_CKUSB_CKPLL_DIV3</i>   | $CK\_USBFS = CK\_PLL / 3$   |
| <i>RCU_CKUSB_CKPLL_DIV3_5</i> | $CK\_USBFS = CK\_PLL / 3.5$ |
| <i>RCU_CKUSB_CKPLL_DIV4</i>   | $CK\_USBFS = CK\_PLL / 4$   |
| <b>Output parameter{out}</b>  |                             |
| -                             | -                           |
| <b>Return value</b>           |                             |
| -                             | -                           |

Example:

```
/* configure the USB prescaler factor */
```

```
rcu_usb_clock_config(RCU_CKUSB_CKPLL_DIV2_5);
```

## rcu\_rtc\_clock\_config

The description of rcu\_rtc\_clock\_config is shown as below:

**Table 3-359. 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</i>      | select CK_HXTAL/128 as RTC source clock               |

|                              |   |
|------------------------------|---|
| <code>_DIV_128</code>        |   |
| <b>Output parameter{out}</b> |   |
| -                            | - |
| <b>Return value</b>          |   |
| -                            | - |

Example:

```
/* configure the RTC clock source selection */
rcu_rtc_clock_config(RCU_RTCSRC_IRC40K);
```

## rcu\_i2s1\_clock\_config

The description of rcu\_i2s1\_clock\_config is shown as below:

**Table 3-360. Function rcu\_i2s1\_clock\_config**

|                                      |  |
|--------------------------------------|--|
| <b>Function name</b>                 | rcu_i2s1_clock_config                                  |
| <b>Function prototype</b>            | void rcu_i2s1_clock_config(uint32_t i2s_clock_source); |
| <b>Function descriptions</b>         | configure the I2S1 clock source selection              |
| <b>Precondition</b>                  | -  |
| <b>The called functions</b>          | -  |
| <b>Input parameter{in}</b>           |  |
| <b>i2s_clock_source</b>              | I2S clock source selection                             |
| <code>RCU_I2S1SRC_CKSYS</code>       | select system clock as I2S1 source clock               |
| <code>RCU_I2S1SRC_CKPLL2_MUL2</code> | select CK_PLL2 * 2 as I2S1 source clock                |
| <b>Output parameter{out}</b>         |  |
| -                                    | -  |
| <b>Return value</b>                  |  |
| -                                    | -  |

Example:

```
/* configure the I2S1 clock source selection */
rcu_i2s1_clock_config(RCU_I2S1SRC_CKPLL2_MUL2);
```

## rcu\_i2s2\_clock\_config

The description of rcu\_i2s2\_clock\_config is shown as below:

**Table 3-361. Function rcu\_i2s2\_clock\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_i2s2_clock_config                                  |
| <b>Function prototype</b>    | void rcu_i2s2_clock_config(uint32_t i2s_clock_source); |
| <b>Function descriptions</b> | configure the I2S2 clock source selection              |
| <b>Precondition</b>          | -  |



|                                |  |
|--------------------------------|--|
| <b>The called functions</b>    | -  |
| <b>Input parameter{in}</b>     |  |
| <b>i2s_clock_source</b>        | I2S clock source selection               |
| <i>RCU_I2S2SRC_CKSYS</i>       | select system clock as I2S2 source clock |
| <i>RCU_I2S2SRC_CKPLL2_MUL2</i> | select CK_PLL2 * 2 as I2S2 source clock  |
| <b>Output parameter{out}</b>   |  |
| -                              | -  |
| <b>Return value</b>            |  |
| -                              | -  |

Example:

```
/* configure the I2S2 clock source selection */
```

```
rcu_i2s2_clock_config(RCU_I2S2SRC_CKPLL2_MUL2);
```

## rcu\_ck48m\_clock\_config

The description of rcu\_ck48m\_clock\_config is shown as below:

**Table 3-362. 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_CKPLL</i>    | CK_PLL 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 CK48M clock source selection */
```

```
rcu_ck48m_clock_config(RCU_CK48MSRC_CKPLL);
```

## rcu\_flag\_get

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

**Table 3-363. Function rcu\_flag\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rcu_flag_get  |
| <b>Function prototype</b>    | FlagStatus rcu_flag_get(rcu_flag_enum flag);  |
| <b>Function descriptions</b> | get the clock stabilization and peripheral reset flags  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>flag</b>                  | the clock stabilization and peripheral reset flags, refer to <a href="#">Table 3-330. rcu_flag_enum</a> |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>FlagStatus</b>            | SET or RESET  |

Example:

```
/* get the clock stabilization flag */
if(RESET != rcu_flag_get(RCU_FLAG_LXTALSTB)){
}
```

## rcu\_all\_reset\_flag\_clear

The description of rcu\_all\_reset\_flag\_clear is shown as below:

**Table 3-364. 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 3-365. Function rcu\_interrupt\_flag\_get**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_interrupt_flag_get   |
| <b>Function prototype</b>    | FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag);                   |
| <b>Function descriptions</b> | get the clock stabilization interrupt and ckm flags                              |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>int_flag</b>              | interrupt and ckm flags, refer to <a href="#">Table 3-331. rcu_int_flag_enum</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>FlagStatus</b>            | SET or RESET   |

Example:

```
/* get the clock stabilization interrupt flag */
if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){
}
```

## rcu\_interrupt\_flag\_clear

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

**Table 3-366. Function rcu\_interrupt\_flag\_clear**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_interrupt_flag_clear   |
| <b>Function prototype</b>    | void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag_clear)  |
| <b>Function descriptions</b> | clear the interrupt flags  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>int_flag_clear</b>        | clock stabilization and stuck interrupt flags clear, refer to <a href="#">Table 3-332. rcu_int_flag_clear_enum</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* clear the interrupt HXTAL stabilization interrupt flag */
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

## rcu\_interrupt\_enable

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

**Table 3-367. Function rcu\_interrupt\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rcu_interrupt_enable  |
| <b>Function prototype</b>    | void rcu_interrupt_enable(rcu_int_enum stab_int);                                 |
| <b>Function descriptions</b> | enable the stabilization interrupt  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>stab_int</b>              | clock stabilization interrupt, refer to <a href="#">Table 3-333. rcu_int_enum</a> |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable the HXTAL stabilization interrupt */
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

## rcu\_interrupt\_disable

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

**Table 3-368. Function rcu\_interrupt\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rcu_interrupt_disable   |
| <b>Function prototype</b>    | void rcu_interrupt_disable(rcu_int_enum stab_int);                                |
| <b>Function descriptions</b> | disable the stabilization interrupt   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>stab_int</b>              | clock stabilization interrupt, refer to <a href="#">Table 3-333. rcu_int_enum</a> |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable the HXTAL stabilization interrupt */
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

## rcu\_lxtal\_drive\_capability\_config

The description of rcu\_lxtal\_drive\_capability\_config is shown as below:

**Table 3-369. Function rcu\_lxtal\_drive\_capability\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_lxtal_drive_capability_config                              |
| <b>Function prototype</b>    | void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap); |
| <b>Function descriptions</b> | configure the LXTAL drive capability                           |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>lxtal_dricap</b>          | drive capability of LXTAL                                      |
| <i>RCU_LXTAL_LOWDRI</i>      | lower driving capability                                       |
| <i>RCU_LXTAL_MED_LOWDRI</i>  | medium low driving capability                                  |
| <i>RCU_LXTAL_MED_HIGHDRI</i> | medium high driving capability                                 |
| <i>RCU_LXTAL_HIGHDRI</i>     | higher driving capability                                      |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure the LXTAL drive capability */
```

```
rcu_lxtal_drive_capability_config (RCU_LXTAL_LOWDRI);
```

## rcu\_osci\_stab\_wait

The description of rcu\_osci\_stab\_wait is shown as below:

**Table 3-370. Function rcu\_osci\_stab\_wait**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rcu_osci_stab_wait  |
| <b>Function prototype</b>    | ErrStatus rcu_osci_stab_wait(rcu_osci_type_enum osci);                          |
| <b>Function descriptions</b> | wait 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 3-334. rcu_osci_type_enum</a>      |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>ErrStatus</b>             | SUCCESS or ERROR  |

Example:

```
/* wait for oscillator stabilization flag */

if(SUCCESS == rcu_osc_stab_wait(RCU_HXTAL)){

}
```

## rcu\_osc\_on

The description of rcu\_osc\_on is shown as below:

**Table 3-371. Function rcu\_osc\_on**

|                       |   |
|-----------------------|---|
| Function name         | rcu_osc_on  |
| Function prototype    | void rcu_osc_on(rcu_osc_type_enum osci);                                  |
| Function descriptions | turn on the oscillator  |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |
| osci                  | oscillator types, refer to <a href="#">Table 3-334. rcu_osc_type_enum</a> |
| Output parameter{out} |   |
| -                     | -   |
| 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 3-372. Function rcu\_osc\_off**

|                       |   |
|-----------------------|---|
| Function name         | rcu_osc_off   |
| Function prototype    | void rcu_osc_off(rcu_osc_type_enum osci);                                 |
| Function descriptions | turn off the oscillator   |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |
| osci                  | oscillator types, refer to <a href="#">Table 3-334. rcu_osc_type_enum</a> |
| Output parameter{out} |   |
| -                     | -   |
| Return value          |   |
| -                     | -   |

Example:

```
/* turn off the high speed crystal oscillator */
```

```
rcu_osci_off(RCU_HXTAL);
```

## rcu\_osci\_bypass\_mode\_enable

The description of rcu\_osci\_bypass\_mode\_enable is shown as below:

**Table 3-373. Function rcu\_osci\_bypass\_mode\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_osci_bypass_mode_enable  |
| <b>Function prototype</b>    | void rcu_osci_bypass_mode_enable(rcu_osci_type_enum osci);                 |
| <b>Function descriptions</b> | enable the oscillator bypass mode  |
| <b>Precondition</b>          | HXTALEN or LXTALEN must be reset before it                                 |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>osci</b>                  | oscillator types, refer to <a href="#">Table 3-334. rcu_osci_type_enum</a> |
| RCU_HXTAL                    | high speed crystal oscillator(HXTAL)                                       |
| RCU_LXTAL                    | low speed crystal oscillator(LXTAL)  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_enable(RCU_HXTAL);
```

## rcu\_osci\_bypass\_mode\_disable

The description of rcu\_osci\_bypass\_mode\_disable is shown as below:

**Table 3-374. Function rcu\_osci\_bypass\_mode\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_osci_bypass_mode_disable   |
| <b>Function prototype</b>    | void rcu_osci_bypass_mode_disable(rcu_osci_type_enum osci);                |
| <b>Function descriptions</b> | disable the oscillator bypass mode   |
| <b>Precondition</b>          | HXTALEN or LXTALEN must be reset before it                                 |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>osci</b>                  | oscillator types, refer to <a href="#">Table 3-334. rcu_osci_type_enum</a> |
| RCU_HXTAL                    | high speed crystal oscillator(HXTAL)                                       |
| RCU_LXTAL                    | low speed crystal oscillator(LXTAL)  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |

| Return value |   |
|--------------|---|
| -            | - |

Example:

```
/* disable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_disable(RCU_HXTAL);
```

## rcu\_hxtal\_clock\_monitor\_enable

The description of rcu\_hxtal\_clock\_monitor\_enable is shown as below:

**Table 3-375. Function rcu\_hxtal\_clock\_monitor\_enable**

| Function name         | rcu_hxtal_clock_monitor_enable             |
|-----------------------|--|
| Function prototype    | void rcu_hxtal_clock_monitor_enable(void); |
| Function descriptions | enable the HXTAL clock monitor             |
| Precondition          | -  |
| The called functions  | -  |
| Input parameter{in}   |  |
| -                     | -  |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

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 3-376. Function rcu\_hxtal\_clock\_monitor\_disable**

| Function name         | rcu_hxtal_clock_monitor_disable             |
|-----------------------|---|
| Function prototype    | void rcu_hxtal_clock_monitor_disable(void); |
| Function descriptions | disable the HXTAL clock monitor             |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |
| -                     | -   |
| Output parameter{out} |   |
| -                     | -   |
| Return value          |   |
| -                     | -   |



Example:

```
/* disable the HXTAL clock monitor */  
  
rcu_hxtal_clock_monitor_disable();
```

## rcu\_irc8m\_adjust\_value\_set

The description of rcu\_irc8m\_adjust\_value\_set is shown as below:

**Table 3-377. 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\_deepsleep\_voltage\_set

The description of rcu\_deepsleep\_voltage\_set is shown as below:

**Table 3-378. Function rcu\_deepsleep\_voltage\_set**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rcu_deepsleep_voltage_set                       |
| <b>Function prototype</b>    | void rcu_deepsleep_voltage_set(uint32_t dsvol); |
| <b>Function descriptions</b> | set the deep-sleep mode voltage value           |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>dsvol</b>                 | deep sleep mode voltage                         |
| <i>RCU_DEEPSLEEP_V_1_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_1_2</i>   | the core voltage is 1.2V 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 3-379. Function rcu\_clock\_freq\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rcu_clock_freq_get                                      |
| <b>Function prototype</b>    | uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock); |
| <b>Function descriptions</b> | get the system clock, bus clock frequency               |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>clock</b>                 | the clock frequency which to get                        |
| <i>CK_SYS</i>                | system clock frequency                                  |
| <i>CK_AHB</i>                | AHB clock frequency                                     |
| <i>CK_APB1</i>               | APB1 clock frequency                                    |
| <i>CK_APB2</i>               | APB2 clock frequency                                    |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>ck_freq</b>               | clock frequency of system, AHB, APB1, APB2              |

Example:

```
uint32_t temp_freq;

/* get the system clock frequency */
temp_freq = rcu_clock_freq_get(CK_SYS);
```

## 3.18. RTC

The Real-time Clock (RTC) is usually used as a clock-calendar. The ones in the Backup Domain consist of a 32-bit up-counter, an alarm, a prescaler, a divider and the RTC clock configuration register. The RTC registers are listed in chapter [3.18.1](#), the FWDGT firmware

functions are introduced in chapter [3.18.2](#).

## 3.18.1. Descriptions of Peripheral registers

RTC registers are listed in the table shown as below:

**Table 3-380. RTC Registers**

| Registers | Descriptions              |
|-----------|---------------------------|
| RTC_INTEN | Interrupt enable register |
| RTC_CTL   | Control register          |
| RTC_PSCH  | Prescaler high register   |
| RTC_PSCL  | Prescaler low register    |
| RTC_DIVH  | Divider high register     |
| RTC_DIVL  | Divider low register      |
| RTC_CNTH  | counter high register     |
| RTC_CNTL  | counter low register      |
| RTC_ALRMH | Alarm high register       |
| RTC_ALRML | Alarm low register        |

## 3.18.2. Descriptions of Peripheral functions

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

**Table 3-381. RTC firmware function**

| Function name                | Function description                            |
|------------------------------|---|
| rtc_configuration_mode_enter | enter RTC configuration mode                    |
| rtc_configuration_mode_exit  | exit RTC configuration mode                     |
| rtc_counter_set              | set RTC counter value                           |
| rtc_prescaler_set            | set RTC prescaler value                         |
| rtc_lwoff_wait               | wait RTC last write operation finished flag set |
| rtc_register_sync_wait       | wait RTC registers synchronized flag set        |
| rtc_alarm_config             | set RTC alarm value                             |
| rtc_counter_get              | get RTC counter value                           |
| rtc_divider_get              | get RTC divider value                           |
| rtc_flag_get                 | get RTC flag status                             |
| rtc_flag_clear               | clear RTC flag status                           |
| rtc_interrupt_flag_get       | get RTC interrupt flag status                   |
| rtc_interrupt_flag_clear     | clear RTC interrupt flag status                 |
| rtc_interrupt_enable         | enable RTC interrupt                            |
| rtc_interrupt_disable        | disable RTC interrupt                           |

### rtc\_configuration\_mode\_enter

The description of rtc\_configuration\_mode\_enter is shown as below:

**Table 3-382. Function rtc\_configuration\_mode\_enter**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rtc_configuration_mode_enter             |
| <b>Function prototype</b>    | void rtc_configuration_mode_enter(void); |
| <b>Function descriptions</b> | enter RTC configuration mode             |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enter RTC configuration mode */
rtc_configuration_mode_enter( );
```

## rtc\_configuration\_mode\_exit

The description of rtc\_configuration\_mode\_exit is shown as below:

**Table 3-383. Function rtc\_configuration\_mode\_exit**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rtc_configuration_mode_exit             |
| <b>Function prototype</b>    | void rtc_configuration_mode_exit(void); |
| <b>Function descriptions</b> | exit RTC configuration mode             |
| <b>Precondition</b>          | -                                       |
| <b>The called functions</b>  | -                                       |
| <b>Input parameter{in}</b>   |   |
| -                            | -                                       |
| <b>Output parameter{out}</b> |   |
| -                            | -                                       |
| <b>Return value</b>          |   |
| -                            | -                                       |

Example:

```
/* exit RTC configuration mode */
rtc_configuration_mode_exit ( );
```

## rtc\_counter\_set

The description of rtc\_counter\_set is shown as below:

**Table 3-384. Function rtc\_counter\_set**

|                      |                 |
|----------------------|-----------------|
| <b>Function name</b> | rtc_counter_set |
|----------------------|-----------------|

|                              |                                     |
|------------------------------|-------------------------------------|
| <b>Function prototype</b>    | void rtc_counter_set(uint32_t cnt); |
| <b>Function descriptions</b> | set RTC counter value               |
| <b>Precondition</b>          | -                                   |
| <b>The called functions</b>  | -                                   |
| <b>Input parameter{in}</b>   |                                     |
| <b>cnt</b>                   | RTC counter value (0-0xFFFF FFFF)   |
| <b>Output parameter{out}</b> |                                     |
| -                            | -                                   |
| <b>Return value</b>          |                                     |
| -                            | -                                   |

Example:

```
/* wait until last write operation on RTC registers has finished */
rtc_lwoff_wait( );

/* set counter value to 0xFFFF */
rtc_counter_set (0xFFFF);
```

## rtc\_prescaler\_set

The description of rtc\_prescaler\_set is shown as below:

**Table 3-385. Function rtc\_prescaler\_set**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rtc_interrupt_rtc_prescaler_set   |
| <b>Function prototype</b>    | void rtc_prescaler_set(uint32_t psc);   |
| <b>Function descriptions</b> | set RTC prescaler value   |
| <b>Precondition</b>          | before using this function, you must call rtc_lwoff_wait ( ) function (wait until LWOFF flag is set). |
| <b>The called functions</b>  | rtc_configuration_mode_enter / rtc_configuration_mode_exit  |
| <b>Input parameter{in}</b>   |   |
| <b>psc</b>                   | RTC prescaler value (0-0x000F FFFF)   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* wait until last write operation on RTC registers has finished */
rtc_lwoff_wait( );

/* set RTC prescaler value to 0x7FFFF */
rtc_prescaler_set (0x7FFFF);
```

## rtc\_lwoff\_wait

The description of rtc\_lwoff\_wait is shown as below:

**Table 3-386. Function rtc\_lwoff\_wait**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rtc_lwoff_wait                                  |
| <b>Function prototype</b>    | void rtc_lwoff_wait(void);                      |
| <b>Function descriptions</b> | wait RTC last write operation finished flag set |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* wait until last write operation on RTC registers has finished */
rtc_lwoff_wait( );

/* enable the RTC second interrupt */
rtc_interrupt_enable(RTC_INT_SECOND);
```

## rtc\_register\_sync\_wait

The description of rtc\_register\_sync\_wait is shown as below:

**Table 3-387. Function rtc\_register\_sync\_wait**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rtc_register_sync_wait                   |
| <b>Function prototype</b>    | void rtc_register_sync_wait(void);       |
| <b>Function descriptions</b> | wait RTC registers synchronized flag set |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* wait for RTC registers synchronization */
rtc_register_sync_wait( );
```

## rtc\_alarm\_config

The description of rtc\_alarm\_config is shown as below:

**Table 3-388. Function rtc\_alarm\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rtc_alarm_config  |
| <b>Function prototype</b>    | void rtc_alarm_config(uint32_t alarm);  |
| <b>Function descriptions</b> | set RTC alarm value   |
| <b>Precondition</b>          | before using this function, you must call rtc_lwoff_wait ( ) function (wait until LWOFF flag is set). - |
| <b>The called functions</b>  | rtc_configuration_mode_enter / rtc_configuration_mode_exit  |
| <b>Input parameter{in}</b>   |   |
| <b>alarm</b>                 | RTC alarm value (0-0xFFFF FFFF)   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* wait until last write operation on RTC registers has finished */
rtc_lwoff_wait( );

/* set alarm value to 0xFFFF */
rtc_alarm_config (0xFFFF);
```

## rtc\_counter\_get

The description of rtc\_counter\_get is shown as below:

**Table 3-389. Function rtc\_counter\_get**

|                              |                                 |
|------------------------------|---------------------------------|
| <b>Function name</b>         | rtc_counter_get                 |
| <b>Function prototype</b>    | uint32_t rtc_counter_get(void); |
| <b>Function descriptions</b> | get RTC counter value           |
| <b>Precondition</b>          | -                               |
| <b>The called functions</b>  | -                               |
| <b>Input parameter{in}</b>   |                                 |
| -                            | -                               |
| <b>Output parameter{out}</b> |                                 |
| -                            | -                               |
| <b>Return value</b>          |                                 |
| <b>uint32_t</b>              | the value of RTC counter        |

Example:

```
/* get the counter value */
```

```
uint32_t rtc_counter_value;
```

```
rtc_counter_value = rtc_counter_get ( );
```

## rtc\_divider\_get

The description of rtc\_divider\_get is shown as below:

**Table 3-390. Function rtc\_divider\_get**

|                              |                                 |
|------------------------------|---------------------------------|
| <b>Function name</b>         | rtc_divider_get                 |
| <b>Function prototype</b>    | uint32_t rtc_divider_get(void); |
| <b>Function descriptions</b> | get RTC divider value           |
| <b>Precondition</b>          | -                               |
| <b>The called functions</b>  | -                               |
| <b>Input parameter{in}</b>   |                                 |
| -                            | -                               |
| <b>Output parameter{out}</b> |                                 |
| -                            | -                               |
| <b>Return value</b>          |                                 |
| uint32_t                     | the value of RTC divider        |

Example:

```
/* get the current RTC divider value */
```

```
uint32_t rtc_divider_value;
```

```
rtc_divider_value = rtc_divider_get ( );
```

## rtc\_flag\_get

The description of rtc\_flag\_getrtc\_interrupt\_enable is shown as below:

**Table 3-391. Function rtc\_flag\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rtc_flag_get                            |
| <b>Function prototype</b>    | FlagStatus rtc_flag_get(uint32_t flag); |
| <b>Function descriptions</b> | get RTC flag status                     |
| <b>Precondition</b>          | -                                       |
| <b>The called functions</b>  | -                                       |
| <b>Input parameter{in}</b>   |   |
| <b>flag</b>                  | specify which RTC flag status to get    |
| <i>RTC_FLAG_SECOND</i>       | second interrupt flag                   |
| <i>RTC_FLAG_ALARM</i>        | alarm interrupt flag                    |
| <i>RTC_FLAG_OVERFLOW</i>     | overflow interrupt flag                 |
| <i>RTC_FLAG_RSYN</i>         | registers synchronized flag             |
| <i>RTC_FLAG_LWOF</i>         | last write operation finished flag      |



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

Example:

```
/* get the RTC overflow interrupt status */
```

```
FlagStatus alarm_status;
```

```
alarm_status = rtc_flag_get (RTC_FLAG_ALARM);
```

## rtc\_flag\_clear

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

**Table 3-392. Function rtc\_flag\_clear**

| Function name          | rtc_flag_clear                         |
|------------------------|--|
| Function prototype     | void rtc_flag_clear(uint32_t flag);    |
| Function descriptions  | clear RTC flag status                  |
| Precondition           | -                                      |
| The called functions   | -                                      |
| Input parameter{in}    |  |
| flag                   | specify which RTC flag status to clear |
| RTC_FLAG_SECOND        | second interrupt flag                  |
| RTC_FLAG_ALARM         | alarm interrupt flag                   |
| RTC_FLAG_OVERFLOW<br>W | overflow interrupt flag                |
| RTC_FLAG_RSYN          | registers synchronized flag            |
| Output parameter{out}  |  |
| -                      | -                                      |
| Return value           |  |
| -                      | -                                      |

Example:

```
/* clear the RTC alarm flag */
```

```
rtc_flag_clear (RTC_FLAG_ALARM);
```

## rtc\_interrupt\_flag\_get

The description of rtc\_interrupt\_flag\_getrtc\_interrupt\_enable is shown as below:

**Table 3-393. Function rtc\_interrupt\_flag\_get**

| Function name      | rtc_interrupt_flag_get                            |
|--------------------|---|
| Function prototype | FlagStatus rtc_interrupt_flag_get(uint32_t flag); |

|                                   |  |
|-----------------------------------|--|
| <b>Function descriptions</b>      | get RTC interrupt flag status                  |
| <b>Precondition</b>               | -  |
| <b>The called functions</b>       | -  |
| <b>Input parameter{in}</b>        |  |
| <b>flag</b>                       | specify which RTC interrupt flag status to get |
| <i>RTC_INT_FLAG_SEC<br/>OND</i>   | second interrupt flag                          |
| <i>RTC_INT_FLAG_ALAR<br/>M</i>    | alarm interrupt flag                           |
| <i>RTC_INT_FLAG_OVE<br/>RFLOW</i> | overflow interrupt flag                        |
| <b>Output parameter{out}</b>      |  |
| -                                 | -  |
| <b>Return value</b>               |  |
| <b>FlagStatus</b>                 | SET or RESET                                   |

Example:

```
/* get the RTC alarm interrupt status */
```

```
FlagStatus alarm_status;
```

```
alarm_status = rtc_interrupt_flag_get (RTC_INT_FLAG_ALARM);
```

## rtc\_interrupt\_flag\_clear

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

**Table 3-394. Function rtc\_interrupt\_flag\_clear**

|                                   |  |
|-----------------------------------|--|
| <b>Function name</b>              | rtc_interrupt_flag_clear                         |
| <b>Function prototype</b>         | void rtc_interrupt_flag_clear(uint32_t flag);    |
| <b>Function descriptions</b>      | clear RTC interrupt flag status                  |
| <b>Precondition</b>               | -  |
| <b>The called functions</b>       | -  |
| <b>Input parameter{in}</b>        |  |
| <b>flag</b>                       | specify which RTC interrupt flag status to clear |
| <i>RTC_INT_FLAG_SEC<br/>OND</i>   | second interrupt flag                            |
| <i>RTC_INT_FLAG_ALAR<br/>M</i>    | alarm interrupt flag                             |
| <i>RTC_INT_FLAG_OVE<br/>RFLOW</i> | overflow interrupt flag                          |
| <b>Output parameter{out}</b>      |  |
| -                                 | -  |
| <b>Return value</b>               |  |
| -                                 | -  |

Example:

```
/* clear the RTC alarm interrupt flag */

rtc_interrupt_flag_clear (RTC_INT_FLAG_ALARM);
```

## rtc\_interrupt\_enable

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

**Table 3-395. Function rtc\_interrupt\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rtc_interrupt_enable  |
| <b>Function prototype</b>    | void rtc_interrupt_enable(uint32_t interrupt);  |
| <b>Function descriptions</b> | enable RTC interrupt  |
| <b>Precondition</b>          | before using this function, you must call rtc_lwoff_wait ( ) function (wait until LWOFF flag is set). |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>interrupt</b>             | specify which RTC interrupt to enable   |
| <i>RTC_INT_SECOND</i>        | second interrupt  |
| <i>RTC_INT_ALARM</i>         | alarm interrupt   |
| <i>RTC_INT_OVERFLOW</i>      | overflow interrupt  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* wait until last write operation on RTC registers has finished */

rtc_lwoff_wait( );

/* enable the RTC second interrupt */

rtc_interrupt_enable(RTC_INT_SECOND);
```

## rtc\_interrupt\_disable

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

**Table 3-396. Function rtc\_interrupt\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rtc_interrupt_disable   |
| <b>Function prototype</b>    | void rtc_interrupt_disable(uint32_t interrupt);   |
| <b>Function descriptions</b> | disable RTC interrupt   |
| <b>Precondition</b>          | before using this function, you must call rtc_lwoff_wait ( ) function (wait until LWOFF flag is set). |
| <b>The called functions</b>  | -   |

| Input parameter{in}     |  |
|-------------------------|--|
| <b>interrupt</b>        | specify which RTC interrupt to disable |
| <i>RTC_INT_SECOND</i>   | second interrupt                       |
| <i>RTC_INT_ALARM</i>    | alarm interrupt                        |
| <i>RTC_INT_OVERFLOW</i> | overflow interrupt                     |
| Output parameter{out}   |  |
| -                       | -                                      |
| Return value            |  |
| -                       | -                                      |

Example:

```
/* wait until last write operation on RTC registers has finished */
```

```
rtc_lwoff_wait( );
```

```
/* disable the RTC second interrupt */
```

```
rtc_interrupt_disable(RTC_INT_SECOND);
```

## 3.19. SPI

The SPI/I2S module can communicate with external devices using the SPI protocol or the I2S audio protocol. The SPI/I2S registers are listed in chapter [3.19.1](#), the SPI/I2S firmware functions are introduced in chapter [3.19.2](#).

### 3.19.1. Descriptions of Peripheral registers

SPI/I2S registers are listed in the table shown as below:

**Table 3-397. SPI/I2S Registers**

| Registers   | Descriptions                     |
|-------------|----------------------------------|
| SPI_CTL0    | SPI control register 0           |
| SPI_CTL1    | SPI control register 1           |
| SPI_STAT    | SPI status register              |
| SPI_DATA    | SPI data register                |
| SPI_CRCPOLY | SPI CRC polynomial register      |
| SPI_RCRC    | SPI receive CRC register         |
| SPI_TCRC    | SPI transmit CRC register        |
| SPI_I2SCTL  | SPI/I2S control register         |
| SPI_I2SPSC  | SPI/I2S clock prescaler register |
| SPI_QCTL    | Quad-SPI mode control register   |

### **3.19.2. Descriptions of Peripheral functions**

SPI/I2S firmware functions are listed in the table shown as below:

**Table 3-398. SPI/I2S firmware function**

| Function name                     | Function description  |
|-----------------------------------|---|
| spi_i2s_deinit                    | reset SPI and I2S peripheral                                    |
| spi_struct_para_init              | initialize the parameters of SPI struct with the default values |
| spi_init                          | initialize SPI peripheral parameter                             |
| spi_enable                        | enable SPI  |
| spi_disable                       | disable SPI   |
| i2s_init                          | initialize I2S peripheral parameter                             |
| i2s_psc_config                    | configure I2S peripheral prescaler                              |
| i2s_enable                        | enable I2S  |
| i2s_disable                       | disable I2S   |
| spi_nss_output_enable             | enable SPI NSS output function                                  |
| spi_nss_output_disable            | disable SPI NSS output function                                 |
| spi_nss_internal_high             | SPI NSS pin high level in software mode                         |
| spi_nss_internal_low              | SPI NSS pin low level in software mode                          |
| spi_dma_enable                    | enable SPI DMA function   |
| spi_dma_disable                   | disable SPI DMA function  |
| spi_i2s_data_frame_format_config  | configure SPI/I2S data frame format                             |
| spi_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                                      |
| spi_quad_enable                   | enable quad wire SPI  |
| spi_quad_disable                  | disable quad wire SPI   |
| spi_quad_write_enable             | enable quad wire SPI write                                      |
| spi_quad_read_enable              | enable quad wire SPI read                                       |
| spi_quad_io23_output_enable       | enable quad wire SPI_IO2 and SPI_IO3 pin output                 |
| spi_quad_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 3-399. spi\_parameter\_struct**

| Member name          | Function description   |
|----------------------|--|
| device_mode          | SPI master or slave<br>(SPI_MASTER, SPI_SLAVE)   |
| trans_mode           | SPI transtype<br>(SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)                      |
| frame_size           | SPI frame size<br>(SPI_FRAME_SIZE_16BIT, SPI_FRAME_SIZE_8BIT)  |
| nss                  | SPI NSS control by hardware or software<br>(SPI_NSS_SOFT, SPI_NSS_HARD)  |
| endian               | SPI big endian or little endian<br>(SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)  |
| clock_polarity_phase | SPI clock phase and polarity<br>(SPI_CLOCK_PL_LOW_PH_1EDGE, SPI_CLOCK_PL_HIGH_PH_1EDGE, SPI_CLOCK_PL_LOW_PH_2EDGE, SPI_CLOCK_PL_HIGH_PH_2EDGE) |
| prescale             | SPI prescale factor<br>(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 3-400. Function spi\_i2s\_deinit**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_i2s_deinit                                     |
| <b>Function prototype</b>    | void spi_i2s_deinit(uint32_t spi_periph);          |
| <b>Function descriptions</b> | reset SPI and I2S peripheral                       |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | rcu_periph_reset_enable / rcu_periph_reset_disable |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | SPI/I2S peripheral                                 |
| <i>SPIx</i>                  | x=0,1,2  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* reset SPI0 */
spi_i2s_deinit(SPI0);
```

## spi\_struct\_para\_init

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

**Table 3-401. Function spi\_struct\_para\_init**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | spi_struct_para_init  |
| <b>Function prototype</b>    | void spi_struct_para_init(spi_parameter_struct* spi_struct);    |
| <b>Function descriptions</b> | initialize the parameters of SPI struct with the default values |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>*spi_struct</b>           | a spi_parameter_struct address                                  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* initialize the parameters of SPI */
spi_parameter_struct spi_init_struct;
spi_struct_para_init(&spi_init_struct);
```

## spi\_init

The description of spi\_init is shown as below:

**Table 3-402. Function spi\_init**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_init   |
| <b>Function prototype</b>    | void spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);  |
| <b>Function descriptions</b> | initialize SPI peripheral parameter  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | SPI peripheral   |
| <i>SPIx</i>                  | x=0,1,2  |
| <b>Input parameter{in}</b>   |  |
| <b>spi_struct</b>            | SPI parameter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-399. spi_parameter_struct</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:



```
/* initialize SPI0 */
```

```
spi_parameter_struct spi_init_struct;
```

```
spi_init_struct.trans_mode      = SPI_TRANSMODE_BDTRANSMIT;
```

```
spi_init_struct.device_mode     = SPI_MASTER;
```

```
spi_init_struct.frame_size      = SPI_FRAME_SIZE_8BIT;
```

```
spi_init_struct.clock_polarity_phase = SPI_CLOCK_POLARITY_HIGH_PHASE_2EDGE;
```

```
spi_init_struct.nss             = SPI_NSS_SOFT;
```

```
spi_init_struct.prescale        = SPI_PSC_8;
```

```
spi_init_struct.endian          = SPI_ENDIAN_MSB;
```

```
spi_init(SPI0, &spi_init_struct);
```

## spi\_enable

The description of spi\_enable is shown as below:

**Table 3-403. Function spi\_enable**

|                              |                                       |
|------------------------------|---------------------------------------|
| <b>Function name</b>         | spi_enable                            |
| <b>Function prototype</b>    | void spi_enable(uint32_t spi_periph); |
| <b>Function descriptions</b> | enable SPI                            |
| <b>Precondition</b>          | -                                     |
| <b>The called functions</b>  | -                                     |
| <b>Input parameter{in}</b>   |                                       |
| <b>spi_periph</b>            | SPI peripheral                        |
| <i>SPIx</i>                  | x=0,1,2                               |
| <b>Output parameter{out}</b> |                                       |
| -                            | -                                     |
| <b>Return value</b>          |                                       |
| -                            | -                                     |

Example:

```
/* enable SPI0 */
```

```
spi_enable(SPI0);
```

## spi\_disable

The description of spi\_disable is shown as below:

**Table 3-404. 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 SPI    |
| <b>Precondition</b>          | -              |
| <b>The called functions</b>  | -              |
| <b>Input parameter{in}</b>   |                |
| <b>spi_periph</b>            | SPI peripheral |
| <b>SPIx</b>                  | x=0,1,2        |
| <b>Output parameter{out}</b> |                |
| -                            | -              |
| <b>Return value</b>          |                |
| -                            | -              |

Example:

```
/* disable SPI0 */
spi_disable(SPI0);
```

### i2s\_init

The description of i2s\_init is shown as below:

**Table 3-405. Function i2s\_init**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2s_init   |
| <b>Function prototype</b>    | void i2s_init(uint32_t spi_periph, uint32_t mode, uint32_t standard, uint32_t ckpl); |
| <b>Function descriptions</b> | initialize I2S peripheral parameter  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | I2S peripheral   |
| <b>SPIx</b>                  | x=1,2  |
| <b>Input parameter{in}</b>   |  |
| <b>mode</b>                  | I2S operation mode   |
| <b>I2S_MODE_SLAVETX</b>      | I2S slave transmit mode  |
| <b>I2S_MODE_SLAVRX</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_PHILIPS</b>       | I2S philips 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   |

|                              |                               |
|------------------------------|-------------------------------|
| <i>I2S_STD_PCMLONG</i>       | I2S PCM long standard         |
| <b>Input parameter{in}</b>   |                               |
| <b>ckpl</b>                  | I2S idle state clock polarity |
| <i>I2S_CKPL_LOW</i>          | I2S clock polarity low level  |
| <i>I2S_CKPL_HIGH</i>         | I2S clock polarity high level |
| <b>Output parameter{out}</b> |                               |
| -                            | -                             |
| <b>Return value</b>          |                               |
| -                            | -                             |

Example:

```
/* initialize I2S1 */
```

```
i2s_init(SPI1, I2S_MODE_MASTERTX, I2S_STD_PHILIPS, I2S_CKPL_LOW);
```

## i2s\_psc\_config

The description of i2s\_psc\_config is shown as below:

**Table 3-406. Function i2s\_psc\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2s_psc_config   |
| <b>Function prototype</b>    | void i2s_psc_config(uint32_t spi_periph, uint32_t audiosample, uint32_t frameformat, uint32_t mckout); |
| <b>Function descriptions</b> | configure I2S prescaler  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | rcu_clock_freq_get   |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | I2S peripheral   |
| <i>SPIx</i>                  | x=1,2  |
| <b>Input parameter{in}</b>   |  |
| <b>audiosample</b>           | I2S audio sample rate  |
| <i>I2S_AUDIOSAMPLE_8K</i>    | audio sample rate is 8KHz  |
| <i>I2S_AUDIOSAMPLE_11K</i>   | audio sample rate is 11KHz   |
| <i>I2S_AUDIOSAMPLE_16K</i>   | audio sample rate is 16KHz   |
| <i>I2S_AUDIOSAMPLE_22K</i>   | audio sample rate is 22KHz   |
| <i>I2S_AUDIOSAMPLE_32K</i>   | audio sample rate is 32KHz   |
| <i>I2S_AUDIOSAMPLE_44K</i>   | audio sample rate is 44KHz   |
| <i>I2S_AUDIOSAMPLE_48K</i>   | audio sample rate is 48KHz   |

|                                 |  |
|---------------------------------|--|
| 8K                              |  |
| I2S_AUDIOSAMPLE_9<br>6K         | audio sample rate is 96KHz                             |
| I2S_AUDIOSAMPLE_1<br>92K        | audio sample rate is 192KHz                            |
| <b>Input parameter{in}</b>      |  |
| <b>frameformat</b>              | I2S data length and channel length                     |
| I2S_FRAMEFORMAT_<br>DT16B_CH16B | I2S data length is 16 bit and channel length is 16 bit |
| I2S_FRAMEFORMAT_<br>DT16B_CH32B | I2S data length is 16 bit and channel length is 32 bit |
| I2S_FRAMEFORMAT_<br>DT24B_CH32B | I2S data length is 24 bit and channel length is 32 bit |
| I2S_FRAMEFORMAT_<br>DT32B_CH32B | I2S data length is 32 bit and channel length is 32 bit |
| <b>Input parameter{in}</b>      |  |
| <b>mckout</b>                   | I2S master clock output                                |
| I2S_MCKOUT_ENABL<br>E           | I2S master clock output enable                         |
| I2S_MCKOUT_DISABL<br>E          | I2S master clock output disable                        |
| <b>Output parameter{out}</b>    |  |
| -                               | -  |
| <b>Return value</b>             |  |
| -                               | -  |

Example:

```
/* configure I2S1 prescaler */
```

```
i2s_psc_config(SPI1, I2S_AUDIOSAMPLE_44K, I2S_FRAMEFORMAT_DT16B_CH16B,  
I2S_MCKOUT_DISABLE);
```

## i2s\_enable

The description of i2s\_enable is shown as below:

**Table 3-407. Function i2s\_enable**

|                              |                                       |
|------------------------------|---------------------------------------|
| <b>Function name</b>         | i2s_enable                            |
| <b>Function prototype</b>    | void i2s_enable(uint32_t spi_periph); |
| <b>Function descriptions</b> | enable I2S                            |
| <b>Precondition</b>          | -                                     |
| <b>The called functions</b>  | -                                     |
| <b>Input parameter{in}</b>   |                                       |
| <b>spi_periph</b>            | I2S peripheral                        |

|                       |               |
|-----------------------|---------------|
| <i>SPIx</i>           | <i>x</i> =1,2 |
| Output parameter{out} |               |
| -                     | -             |
| Return value          |               |
| -                     | -             |

Example:

```
/* enable I2S1*/
i2s_enable(SPI1);
```

## i2s\_disable

The description of i2s\_disable is shown as below:

**Table 3-408. Function i2s\_disable**

|                       |  |
|-----------------------|--|
| Function name         | i2s_disable                            |
| Function prototype    | void i2s_disable(uint32_t spi_periph); |
| Function descriptions | disable I2S                            |
| Precondition          | -                                      |
| The called functions  | -                                      |
| Input parameter{in}   |  |
| spi_periph            | I2S peripheral                         |
| <i>SPIx</i>           | <i>x</i> =1,2                          |
| Output parameter{out} |  |
| -                     | -                                      |
| Return value          |  |
| -                     | -                                      |

Example:

```
/* disable I2S1*/
i2s_disable(SPI1);
```

## spi\_nss\_output\_enable

The description of spi\_nss\_output\_enable is shown as below:

**Table 3-409. Function spi\_nss\_output\_enable**

|                       |  |
|-----------------------|--|
| Function name         | spi_nss_output_enable                            |
| Function prototype    | void spi_nss_output_enable(uint32_t spi_periph); |
| Function descriptions | enable SPI NSS output function                   |
| Precondition          | -  |
| The called functions  | -  |
| Input parameter{in}   |  |
| spi_periph            | SPI peripheral                                   |

|                              |                |
|------------------------------|----------------|
| <i>SPIx</i>                  | <i>x=0,1,2</i> |
| <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 3-410. Function spi\_nss\_output\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | spi_nss_output_disable                            |
| <b>Function prototype</b>    | void spi_nss_output_disable(uint32_t spi_periph); |
| <b>Function descriptions</b> | disable SPI NSS output function                   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>spi_periph</b>            | SPI peripheral                                    |
| <i>SPIx</i>                  | <i>x=0,1,2</i>                                    |
| <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 3-411. Function spi\_nss\_internal\_high**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_nss_internal_high                            |
| <b>Function prototype</b>    | void spi_nss_internal_high(uint32_t spi_periph); |
| <b>Function descriptions</b> | SPI NSS pin high level in software mode          |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | SPI peripheral                                   |

|                              |                |
|------------------------------|----------------|
| <i>SPIx</i>                  | <i>x=0,1,2</i> |
| <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 3-412. Function spi\_nss\_internal\_low**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | spi_nss_internal_low                            |
| <b>Function prototype</b>    | void spi_nss_internal_low(uint32_t spi_periph); |
| <b>Function descriptions</b> | SPI NSS pin low level in software mode          |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>spi_periph</b>            | SPI peripheral                                  |
| <i>SPIx</i>                  | <i>x=0,1,2</i>                                  |
| <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 3-413. Function spi\_dma\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_dma_enable   |
| <b>Function prototype</b>    | void spi_dma_enable(uint32_t spi_periph, uint8_t dma); |
| <b>Function descriptions</b> | enable SPI DMA function                                |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | SPI peripheral   |

|                              |                           |
|------------------------------|---------------------------|
| <i>SPIx</i>                  | x=0,1,2                   |
| <b>Input parameter{in}</b>   |                           |
| <b>dma</b>                   | SPI DMA mode              |
| <i>SPI_DMA_TRANSMIT</i>      | SPI transmit data use DMA |
| <i>SPI_DMA_RECEIVE</i>       | SPI receive data use DMA  |
| <b>Output parameter{out}</b> |                           |
| -                            | -                         |
| <b>Return value</b>          |                           |
| -                            | -                         |

Example:

```
/* 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 3-414. Function spi\_dma\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | spi_dma_disable   |
| <b>Function prototype</b>    | void spi_dma_disable(uint32_t spi_periph, uint8_t dma); |
| <b>Function descriptions</b> | disable SPI DMA function                                |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>spi_periph</b>            | SPI peripheral  |
| <i>SPIx</i>                  | x=0,1,2   |
| <b>Input parameter{in}</b>   |   |
| <b>dma</b>                   | SPI DMA mode  |
| <i>SPI_DMA_TRANSMIT</i>      | SPI transmit data use DMA                               |
| <i>SPI_DMA_RECEIVE</i>       | SPI receive data use DMA                                |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable SPI0 transmit data DMA function */
```

```
spi_dma_disable(SPI0, SPI_DMA_TRANSMIT);
```

### spi\_i2s\_data\_frame\_format\_config

The description of spi\_i2s\_data\_frame\_format\_config is shown as below:



**Table 3-415. Function spi\_i2s\_data\_frame\_format\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_i2s_data_frame_format_config   |
| <b>Function prototype</b>    | void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format); |
| <b>Function descriptions</b> | configure SPI/I2S data frame format  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | SPI peripheral   |
| <i>SPIx</i>                  | x=0,1,2  |
| <b>Input parameter{in}</b>   |  |
| <b>frame_format</b>          | SPI frame size   |
| <i>SPI_FRAME_SIZE_16BIT</i>  | SPI frame size is 16 bits  |
| <i>SPI_FRAME_SIZE_8BIT</i>   | SPI frame size is 8 bits   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure SPI1/I2S1 data frame format size is 16 bits */
```

```
spi_i2s_data_frame_format_config(SPI1, SPI_FRAME_SIZE_16BIT);
```

## spi\_i2s\_data\_transmit

The description of spi\_i2s\_data\_transmit is shown as below:

**Table 3-416. Function spi\_i2s\_data\_transmit**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | spi_i2s_data_transmit   |
| <b>Function prototype</b>    | void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data); |
| <b>Function descriptions</b> | SPI transmit data   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>spi_periph</b>            | SPI peripheral  |
| <i>SPIx</i>                  | x=0,1,2   |
| <b>Input parameter{in}</b>   |   |
| <b>data</b>                  | 16-bit data   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* SPI0 transmit data */
```

```
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n]);
```

## spi\_i2s\_data\_receive

The description of spi\_i2s\_data\_receive is shown as below:

**Table 3-417. Function spi\_i2s\_data\_receive**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | spi_i2s_data_receive                                |
| <b>Function prototype</b>    | uint16_t spi_i2s_data_receive(uint32_t spi_periph); |
| <b>Function descriptions</b> | SPI receive data                                    |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>spi_periph</b>            | SPI peripheral                                      |
| <i>SPIx</i>                  | x=0,1,2   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>uint16_t</b>              | 16-bit data   |

Example:

```
/* SPI0 receive data */
```

```
spi0_receive_array[receive_n] = spi_i2s_data_receive(SPI0);
```

## spi\_bidirectional\_transfer\_config

The description of spi\_bidirectional\_transfer\_config is shown as below:

**Table 3-418. 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 SPI bidirectional transfer direction  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>spi_periph</b>            | SPI peripheral  |
| <i>SPIx</i>                  | x=0,1,2   |
| <b>Input parameter{in}</b>   |   |
| <b>transfer_direction</b>    | SPI transfer direction  |
| <i>SPI_BIDIRECTIONAL_</i>    | SPI work in transmit-only mode  |

|                                  |                               |
|----------------------------------|-------------------------------|
| <i>TRANSMIT</i>                  |                               |
| <i>SPI_BIDIRECTIONAL_RECEIVE</i> | SPI work in receive-only mode |
| <b>Output parameter{out}</b>     |                               |
| -                                | -                             |
| <b>Return value</b>              |                               |
| -                                | -                             |

Example:

```
/* SPI0 works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

## spi\_crc\_polynomial\_set

The description of spi\_crc\_polynomial\_set is shown as below:

**Table 3-419. Function spi\_crc\_polynomial\_set**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_crc_polynomial_set   |
| <b>Function prototype</b>    | void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly); |
| <b>Function descriptions</b> | set SPI CRC polynomial   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | SPI peripheral   |
| <i>SPIx</i>                  | x=0,1,2  |
| <b>Input parameter{in}</b>   |  |
| <b>crc_poly</b>              | CRC polynomial value   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <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 3-420. Function spi\_crc\_polynomial\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | spi_crc_polynomial_get                                |
| <b>Function prototype</b>    | uint16_t spi_crc_polynomial_get(uint32_t spi_periph); |
| <b>Function descriptions</b> | get SPI CRC polynomial                                |

|                              |                                  |
|------------------------------|----------------------------------|
| <b>Precondition</b>          | -                                |
| <b>The called functions</b>  | -                                |
| <b>Input parameter{in}</b>   |                                  |
| <b>spi_periph</b>            | SPI peripheral                   |
| <i>SPIx</i>                  | x=0,1,2                          |
| <b>Output parameter{out}</b> |                                  |
| -                            | -                                |
| <b>Return value</b>          |                                  |
| <b>uint16_t</b>              | 16-bit CRC polynomial (0-0xFFFF) |

Example:

```
/* get SPI0 CRC polynomial */
uint16_t crc_val;
crc_val = spi_crc_polynomial_get(SPI0);
```

## spi\_crc\_on

The description of spi\_crc\_on is shown as below:

**Table 3-421. Function spi\_crc\_on**

|                              |                                       |
|------------------------------|---------------------------------------|
| <b>Function name</b>         | spi_crc_on                            |
| <b>Function prototype</b>    | void spi_crc_on(uint32_t spi_periph); |
| <b>Function descriptions</b> | turn on SPI CRC function              |
| <b>Precondition</b>          | -                                     |
| <b>The called functions</b>  | -                                     |
| <b>Input parameter{in}</b>   |                                       |
| <b>spi_periph</b>            | SPI peripheral                        |
| <i>SPIx</i>                  | x=0,1,2                               |
| <b>Output parameter{out}</b> |                                       |
| -                            | -                                     |
| <b>Return value</b>          |                                       |
| -                            | -                                     |

Example:

```
/* turn on SPI0 CRC function */
spi_crc_on(SPI0);
```

## spi\_crc\_off

The description of spi\_crc\_off is shown as below:

**Table 3-422. Function spi\_crc\_off**

|                      |             |
|----------------------|-------------|
| <b>Function name</b> | spi_crc_off |
|----------------------|-------------|

|                              |  |
|------------------------------|--|
| <b>Function prototype</b>    | void spi_crc_off(uint32_t spi_periph); |
| <b>Function descriptions</b> | turn off SPI CRC function              |
| <b>Precondition</b>          | -                                      |
| <b>The called functions</b>  | -                                      |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | SPI peripheral                         |
| <i>SPIx</i>                  | x=0,1,2                                |
| <b>Output parameter{out}</b> |  |
| -                            | -                                      |
| <b>Return value</b>          |  |
| -                            | -                                      |

Example:

```
/* turn off SPI0 CRC function */
```

```
spi_crc_off(SPI0);
```

## spi\_crc\_next

The description of spi\_crc\_next is shown as below:

**Table 3-423. Function spi\_crc\_next**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | spi_crc_next                            |
| <b>Function prototype</b>    | void spi_crc_next(uint32_t spi_periph); |
| <b>Function descriptions</b> | SPI next data is CRC value              |
| <b>Precondition</b>          | -                                       |
| <b>The called functions</b>  | -                                       |
| <b>Input parameter{in}</b>   |   |
| <b>spi_periph</b>            | SPI peripheral                          |
| <i>SPIx</i>                  | x=0,1,2                                 |
| <b>Output parameter{out}</b> |   |
| -                            | -                                       |
| <b>Return value</b>          |   |
| -                            | -                                       |

Example:

```
/* SPI0 next data is CRC value */
```

```
spi_crc_next(SPI0);
```

## spi\_crc\_get

The description of spi\_crc\_get is shown as below:

**Table 3-424. Function spi\_crc\_get**

|                      |             |
|----------------------|-------------|
| <b>Function name</b> | spi_crc_get |
|----------------------|-------------|

|                              |   |
|------------------------------|---|
| <b>Function prototype</b>    | uint16_t spi_crc_get(uint32_t spi_periph, uint8_t crc); |
| <b>Function descriptions</b> | get SPI CRC send value or receive value                 |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>spi_periph</b>            | SPI peripheral  |
| <i>SPIx</i>                  | x=0,1,2   |
| <b>Input parameter{in}</b>   |   |
| <b>crc</b>                   | SPI crc value   |
| <i>SPI_CRC_TX</i>            | get transmit crc value                                  |
| <i>SPI_CRC_RX</i>            | get receive crc value                                   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>uint16_t</b>              | 16-bit CRC value (0-0xFFFF)                             |

Example:

```
/* get SPI0 CRC send value */
uint16_t 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 3-425. Function spi\_ti\_mode\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | spi_ti_mode_enable                            |
| <b>Function prototype</b>    | void spi_ti_mode_enable(uint32_t spi_periph); |
| <b>Function descriptions</b> | enable SPI TI mode                            |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>spi_periph</b>            | SPI peripheral                                |
| <i>SPIx</i>                  | x=0,1,2                                       |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <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 3-426. Function spi\_ti\_mode\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_ti_mode_disable                            |
| <b>Function prototype</b>    | void spi_ti_mode_disable(uint32_t spi_periph); |
| <b>Function descriptions</b> | disable SPI TI mode                            |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | SPI peripheral                                 |
| <i>SPIx</i>                  | x=0,1,2  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

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 3-427. Function spi\_nssp\_mode\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | spi_nssp_mode_enable                            |
| <b>Function prototype</b>    | void spi_nssp_mode_enable(uint32_t spi_periph); |
| <b>Function descriptions</b> | enable SPI NSS pulse mode                       |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>spi_periph</b>            | SPI peripheral                                  |
| <i>SPIx</i>                  | x=0,1,2   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

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 3-428. Function spi\_nssp\_mode\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_nssp_mode_disable                            |
| <b>Function prototype</b>    | void spi_nssp_mode_disable(uint32_t spi_periph); |
| <b>Function descriptions</b> | disable SPI NSS pulse mode                       |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | SPI peripheral                                   |
| <i>SPIx</i>                  | x=0,1,2  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable SPI0 NSS pulse mode */
spi_nssp_mode_disable(SPI0);
```

## **spi\_quad\_enable**

The description of spi\_quad\_enable is shown as below:

**Table 3-429. Function spi\_quad\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_quad_enable                            |
| <b>Function prototype</b>    | void spi_quad_enable(uint32_t spi_periph); |
| <b>Function descriptions</b> | enable quad wire SPI                       |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | SPI peripheral                             |
| <i>SPIx</i>                  | x=0  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable SPI0 quad wire mode */
spi_quad_enable(SPI0);
```



## spi\_quad\_disable

The description of spi\_quad\_disable is shown as below:

**Table 3-430. Function spi\_quad\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_quad_disable                       |
| <b>Function prototype</b>    | spi_quad_disable(uint32_t spi_periph); |
| <b>Function descriptions</b> | disable quad wire SPI                  |
| <b>Precondition</b>          | -                                      |
| <b>The called functions</b>  | -                                      |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | SPI peripheral                         |
| <i>SPIx</i>                  | x=0                                    |
| <b>Output parameter{out}</b> |  |
| -                            | -                                      |
| <b>Return value</b>          |  |
| -                            | -                                      |

Example:

```
/* disable SPI0 quad wire mode */
```

```
spi_quad_disable(SPI0);
```

## spi\_quad\_write\_enable

The description of spi\_quad\_write\_enable is shown as below:

**Table 3-431. Function spi\_quad\_write\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_quad_write_enable                            |
| <b>Function prototype</b>    | void spi_quad_write_enable(uint32_t spi_periph); |
| <b>Function descriptions</b> | enable quad wire SPI write                       |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | SPI peripheral                                   |
| <i>SPIx</i>                  | x=0  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable SPI0 quad wire write */
```

```
spi_quad_write_enable(SPI0);
```

## **spi\_quad\_read\_enable**

The description of spi\_quad\_read\_enable is shown as below:

**Table 3-432. Function spi\_quad\_read\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | spi_quad_read_enable                            |
| <b>Function prototype</b>    | void spi_quad_read_enable(uint32_t spi_periph); |
| <b>Function descriptions</b> | enable quad wire SPI read                       |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>spi_periph</b>            | SPI peripheral                                  |
| <i>SPIx</i>                  | x=0   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable SPI0 quad wire read */
```

```
spi_quad_read_enable(SPI0);
```

## **spi\_quad\_io23\_output\_enable**

The description of spi\_quad\_io23\_output\_enable is shown as below:

**Table 3-433. Function spi\_quad\_io23\_output\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_quad_io23_output_enable                            |
| <b>Function prototype</b>    | void spi_quad_io23_output_enable(uint32_t spi_periph); |
| <b>Function descriptions</b> | enable 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   |
| <i>SPIx</i>                  | x=0  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable SPI0 SPI_IO2 and SPI_IO3 pin output */
```

```
spi_quad_io23_output_enable(SPI0);
```

## spi\_quad\_io23\_output\_disable

The description of spi\_quad\_io23\_output\_disable is shown as below:

**Table 3-434. Function spi\_quad\_io23\_output\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | spi_quad_io23_output_disable                            |
| <b>Function prototype</b>    | void spi_quad_io23_output_disable(uint32_t spi_periph); |
| <b>Function descriptions</b> | 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  |
| <i>SPIx</i>                  | x=0   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable SPI0 SPI_IO2 and SPI_IO3 pin output */
```

```
spi_quad_io23_output_disable(SPI0);
```

## spi\_i2s\_interrupt\_enable

The description of spi\_i2s\_interrupt\_enable is shown as below:

**Table 3-435. Function spi\_i2s\_interrupt\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | spi_i2s_interrupt_enable  |
| <b>Function prototype</b>    | void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t interrupt);  |
| <b>Function descriptions</b> | enable SPI and I2S interrupt  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>spi_periph</b>            | SPI peripheral  |
| <i>SPIx</i>                  | x=0,1,2   |
| <b>Input parameter{in}</b>   |   |
| <b>interrupt</b>             | SPI/I2S interrupt   |
| <i>SPI_I2S_INT_TBE</i>       | transmit buffer empty interrupt   |
| <i>SPI_I2S_INT_RBNE</i>      | receive buffer not empty interrupt  |
| <i>SPI_I2S_INT_ERR</i>       | CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |

Example:

```
/* enable SPI0 transmit buffer empty interrupt */  
  
spi_i2s_interrupt_enable(SPI0, SPI_I2S_INT_TBE);
```

## spi\_i2s\_interrupt\_disable

The description of spi\_i2s\_interrupt\_disable is shown as below:

**Table 3-436. Function spi\_i2s\_interrupt\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | spi_i2s_interrupt_disable   |
| <b>Function prototype</b>    | void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);                                       |
| <b>Function descriptions</b> | disable SPI and I2S interrupt   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>spi_periph</b>            | SPI peripheral  |
| <i>SPIx</i>                  | x=0,1,2   |
| <b>Input parameter{in}</b>   |   |
| <b>interrupt</b>             | SPI/I2S interrupt   |
| <i>SPI_I2S_INT_TBE</i>       | transmit buffer empty interrupt   |
| <i>SPI_I2S_INT_RBNE</i>      | receive buffer not empty interrupt  |
| <i>SPI_I2S_INT_ERR</i>       | CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable SPI0 transmit buffer empty interrupt */  
  
spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_TBE);
```

## spi\_i2s\_interrupt\_flag\_get

The description of spi\_i2s\_interrupt\_flag\_get is shown as below:

**Table 3-437. Function spi\_i2s\_interrupt\_flag\_get**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_i2s_interrupt_flag_get   |
| <b>Function prototype</b>    | FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt); |
| <b>Function descriptions</b> | get SPI and I2S interrupt status   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |

| Input parameter{in}             |                                    |
|---------------------------------|------------------------------------|
| <b>spi_periph</b>               | SPI peripheral                     |
| <i>SPIx</i>                     | x=0,1,2                            |
| Input parameter{in}             |                                    |
| <b>interrupt</b>                | SPI/I2S interrupt flag status      |
| <i>SPI_I2S_INT_FLAG_TBE</i>     | transmit buffer empty interrupt    |
| <i>SPI_I2S_INT_FLAG_RBNE</i>    | receive buffer not empty interrupt |
| <i>SPI_I2S_INT_FLAG_RXORERR</i> | overrun interrupt                  |
| <i>SPI_INT_FLAG_CONFERR</i>     | config error interrupt             |
| <i>SPI_INT_FLAG_CRCERR</i>      | CRC error interrupt                |
| <i>I2S_INT_FLAG_TXURERR</i>     | underrun error interrupt           |
| Output parameter{out}           |                                    |
| -                               | -                                  |
| Return value                    |                                    |
| <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 3-438. Function spi\_i2s\_flag\_get**

| <b>Function name</b>         | spi_i2s_flag_get   |
|------------------------------|--|
| <b>Function prototype</b>    | FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag); |
| <b>Function descriptions</b> | get SPI and I2S flag status                                      |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| Input parameter{in}          |  |
| <b>spi_periph</b>            | SPI peripheral   |
| <i>SPIx</i>                  | x=0,1,2  |

| Input parameter{in}             |                               |
|---------------------------------|-------------------------------|
| <b>flag</b>                     | SPI/I2S flag status           |
| <i>SPI_FLAG_TBE</i>             | transmit buffer empty flag    |
| <i>SPI_FLAG_RBNE</i>            | receive buffer not empty flag |
| <i>SPI_FLAG_TRANS</i>           | transmit on-going flag        |
| <i>SPI_I2S_INT_FLAG_RXORERR</i> | receive overrun error flag    |
| <i>SPI_FLAG_CONFERR</i>         | mode config error flag        |
| <i>SPI_FLAG_CRCERR</i>          | CRC error flag                |
| <i>I2S_FLAG_RXORERR</i>         | overrun error flag            |
| <i>I2S_FLAG_TXURERR</i>         | underrun error flag           |
| <i>I2S_FLAG_CH</i>              | channel side flag             |
| Output parameter{out}           |                               |
| -                               | -                             |
| Return value                    |                               |
| <b>FlagStatus</b>               | SET or RESET                  |

Example:

```
/* get SPI0 transmit buffer empty flag status */
while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
```

## spi\_crc\_error\_clear

The description of spi\_crc\_error\_clear is shown as below:

**Table 3-439. 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>  | -  |
| Input parameter{in}          |  |
| <b>spi_periph</b>            | SPI peripheral                                 |
| <i>SPIx</i>                  | x=0,1,2  |
| Output parameter{out}        |  |
| -                            | -  |
| Return value                 |  |
| -                            | -  |

Example:

```
/* clear SPI0 CRC error flag status */
```

```
spi_crc_error_clear(SPI0);
```

## 3.20. TIMER

The timers have a 16-bit counter that can be used as an unsigned counter and supports both input capture and output compare. Timers (TIMERx) are divided into five sorts: advanced timer (TIMERx, x=0, 7), general level0 timer (TIMERx, x=1, 2, 3, 4), general level1 timer (TIMERx, x=8, 11), general level2 timer (TIMERx, x=9, 10, 12, 13), Basic timer (TIMERx, x=5, 6). The specific functions of different types of timer are different. The TIMER registers are listed in chapter [3.20.1](#), the TIMER firmware functions are introduced in chapter [3.20.2](#).

### 3.20.1. Descriptions of Peripheral registers

TIMERx registers are listed in the table shown as below:

**Table 3-440. TIMERx Registers**

| Registers      | Descriptions                              |
|----------------|---|
| TIMER_CTL0     | Control register 0                        |
| TIMER_CTL1     | Control register 1                        |
| TIMER_SMCFG    | Slave mode configuration register         |
| TIMER_DMAINTEN | DMA and interrupt enable register         |
| TIMER_INTF     | Interrupt flag register                   |
| TIMER_SWEVG    | Software event generation register        |
| TIMER_CHCTL0   | Channel control register 0                |
| TIMER_CHCTL1   | Channel control register 1                |
| TIMER_CHCTL2   | Channel control register 2                |
| TIMER_CNT      | Counter register                          |
| TIMER_PSC      | Prescaler register                        |
| TIMER_CAR      | Counter auto reload register              |
| TIMER_CREP     | Counter repetition register               |
| TIMER_CH0CV    | Channel 0 capture/compare value register  |
| TIMER_CH1CV    | Channel 1 capture/compare value register  |
| TIMER_CH2CV    | Channel 2 capture/compare value register  |
| TIMER_CH3CV    | Channel 3 capture/compare value register  |
| TIMER_CCHP     | Channel complementary protection register |
| TIMER_DMACFG   | DMA configuration register                |
| TIMER_DMATB    | DMA transfer buffer register              |
| TIMER_CFG      | Configuration register                    |

### 3.20.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

**Table 3-441. TIMERx firmware function**

| Function name                              | Function description  |
|--|---|
| timer_deinit                               | deinit a timer  |
| timer_struct_para_init                     | initialize the parameters of TIMER init parameter struct with the default values  |
| timer_init                                 | initialize TIMER counter  |
| timer_enable                               | enable a timer  |
| timer_disable                              | disable a timer   |
| timer_auto_reload_shadow_enable            | enable the auto reload shadow function  |
| timer_auto_reload_shadow_disable           | disable the auto reload shadow function   |
| timer_update_event_enable                  | enable the update event   |
| timer_update_event_disable                 | disable the update event  |
| timer_counter_alignment                    | set TIMER counter alignment mode  |
| timer_counter_up_direction                 | set TIMER counter up direction  |
| timer_counter_down_direction               | set TIMER counter down direction  |
| timer_prescaler_config                     | configure TIMER prescaler   |
| timer_repetition_value_config              | configure TIMER repetition register value   |
| timer_autoreload_value_config              | configure TIMER autoreload register value   |
| timer_counter_value_config                 | configure TIMER counter register value  |
| timer_counter_read                         | read TIMER counter value  |
| timer_prescaler_read                       | read TIMER prescaler value  |
| timer_single_pulse_mode_config             | configure TIMER single pulse mode   |
| timer_update_source_config                 | configure TIMER update source   |
| timer_dma_enable                           | enable the TIMER DMA  |
| timer_dma_disable                          | disable the TIMER DMA   |
| timer_channel_dma_request_source_select    | channel DMA request source selection  |
| timer_dma_transfer_config                  | configure the TIMER DMA transfer  |
| timer_event_software_generate              | software generate events  |
| timer_break_struct_para_init               | initialize the parameters of TIMER break parameter struct with the default values |
| timer_break_config                         | configure TIMER break function  |
| timer_break_enable                         | enable TIMER break function   |
| timer_break_disable                        | disable TIMER break function  |
| timer_automatic_output_enable              | enable TIMER output automatic function  |
| timer_automatic_output_disable             | disable TIMER output automatic function   |
| timer_primary_output_config                | configure TIMER primary output function   |
| timer_channel_control_shadow_config        | channel capture/compare control shadow register enable                            |
| timer_channel_control_shadow_update_config | configure TIMER channel control shadow register update control                    |
| timer_channel_output_struct                | initialize the parameters of TIMER channel output                                 |



| Function name                                      | Function description  |
|--|---|
| _para_init   | parameter struct with the default values  |
| timer_channel_output_config                        | configure TIMER channel output function   |
| timer_channel_output_mode_config                   | configure TIMER channel output compare mode   |
| timer_channel_output_pulse_value_config            | configure TIMER channel output pulse value  |
| timer_channel_output_shadow_config                 | configure TIMER channel output shadow function  |
| timer_channel_output_fast_config                   | configure TIMER channel output fast function  |
| timer_channel_output_clear_config                  | configure TIMER channel output clear function   |
| timer_channel_output_polarity_config               | configure TIMER channel output polarity   |
| timer_channel_complementary_output_polarity_config | configure TIMER channel complementary output polarity                                     |
| timer_channel_output_state_config                  | configure TIMER channel enable state  |
| timer_channel_complementary_output_state_config    | configure TIMER channel complementary output enable state                                 |
| timer_channel_input_struct_para_init               | initialize the parameters of TIMER channel input parameter struct with the default values |
| timer_input_capture_config                         | configure TIMER input capture parameter   |
| timer_channel_input_capture_prescaler_config       | configure TIMER channel input capture prescaler value                                     |
| timer_channel_capture_value_register_read          | read TIMER channel capture compare register value   |
| timer_input_pwm_capture_config                     | configure TIMER input pwm capture function  |
| timer_hall_mode_config                             | configure TIMER hall sensor mode  |
| timer_input_trigger_source_select                  | select TIMER input trigger source   |
| timer_master_output_trigger_source_select          | select TIMER master mode output trigger source  |
| timer_slave_mode_select                            | select TIMER slave mode   |
| timer_master_slave_mode_config                     | configure TIMER master slave mode   |
| timer_external_trigger_config                      | configure TIMER external trigger input  |
| timer_quadrature_decoder_mode_config               | configure TIMER quadrature decoder mode   |
| timer_internal_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   |

| Function name                       | Function description                            |
|-------------------------------------|---|
| timer_write_chxval_register_config  | configure TIMER write CHxVAL register selection |
| timer_output_value_selection_config | configure TIMER output value selection          |
| timer_interrupt_enable              | enable the TIMER interrupt                      |
| timer_interrupt_disable             | disable the TIMER interrupt                     |
| timer_interrupt_flag_get            | get timer interrupt flag                        |
| timer_interrupt_flag_clear          | clear TIMER interrupt flag                      |
| timer_flag_get                      | get TIMER flags                                 |
| timer_flag_clear                    | clear TIMER flags                               |

## Structure timer\_parameter\_struct

**Table 3-442. Structure timer\_parameter\_struct**

| Member name       | Function description   |
|-------------------|--|
| prescaler         | prescaler value (0~65535)  |
| alignedmode       | aligned mode (TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH) |
| counterdirection  | counter direction (TIMER_COUNTER_UP, TIMER_COUNTER_DOWN)   |
| period            | period value (0~65535)   |
| clockdivision     | clock division value (TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4)                                      |
| repetitioncounter | the counter repetition value (0~255)   |

## Structure timer\_break\_parameter\_struct

**Table 3-443. 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 3-444. 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 3-445. Structure timer\_ic\_parameter\_struct**

| Member name | Function description  |
|-------------|---|
| icpolarity  | channel input polarity (TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_FALLING, TIMER_IC_POLARITY_BOTH_EDGE)         |
| icselection | channel input mode selection (TIMER_IC_SELECTION_DIRECTTI, TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS) |
| icprescaler | channel input capture prescaler (TIMER_IC_PSC_DIV1, TIMER_IC_PSC_DIV2, TIMER_IC_PSC_DIV4, TIMER_IC_PSC_DIV8)      |
| icfilter    | channel input capture filter control (0~15)   |

## timer\_deinit

The description of timer\_deinit is shown as below:

**Table 3-446. Function timer\_deinit**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | timer_deinit                                       |
| <b>Function prototype</b>    | void timer_deinit(uint32_t timer_periph);          |
| <b>Function descriptions</b> | deinit a TIMER                                     |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | rcu_periph_reset_enable / rcu_periph_reset_disable |
| <b>Input parameter{in}</b>   |  |
| <b>timer_periph</b>          | TIMER peripheral                                   |
| <b>TIMERx(x=0..13)</b>       | TIMER peripheral selection                         |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |

|   |   |
|---|---|
| - | - |
|---|---|

Example:

```
/* reset TIMER0 */
```

```
timer_deinit (TIMER0);
```

## timer\_struct\_para\_init

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

**Table 3-447. Function timer\_struct\_para\_init**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_struct_para_init  |
| <b>Function prototype</b>    | void timer_struct_para_init(timer_parameter_struct* initpara);  |
| <b>Function descriptions</b> | initialize the parameters of TIMER init parameter struct with the default values  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>initpara</b>              | TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-442. Structure timer_parameter_struct</a> . |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* initialize TIMER init parameter struct with a default value */
```

```
timer_parameter_struct timer_initpara;
```

```
timer_struct_para_init(timer_initpara);
```

## timer\_init

The description of timer\_init is shown as below:

**Table 3-448. 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..13)</i>       | TIMER peripheral selection  |

| Input parameter{in}   |  |
|-----------------------|--|
| <b>initpara</b>       | TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-442. Structure timer parameter struct.</a> |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

Example:

```

/* initialize TIMER0 */

timer_parameter_struct timer_initpara;

timer_initpara.prescaler      = 107;

timer_initpara.alignedmode    = TIMER_COUNTER_EDGE;

timer_initpara.counterdirection = TIMER_COUNTER_UP;

timer_initpara.period         = 999;

timer_initpara.clockdivision   = TIMER_CKDIV_DIV1;

timer_initpara.repetitioncounter = 1;

timer_init(TIMER0,&timer_initpara);

```

## timer\_enable

The description of timer\_enable is shown as below:

**Table 3-449. Function timer\_enable**

| Function name          | timer_enable                              |
|------------------------|---|
| Function prototype     | void timer_enable(uint32_t timer_periph); |
| Function descriptions  | enable a timer                            |
| Precondition           | -   |
| The called functions   | -   |
| Input parameter{in}    |   |
| <b>timer_periph</b>    | TIMER peripheral                          |
| <i>TIMERx(x=0..13)</i> | TIMER peripheral selection                |
| Output parameter{out}  |   |
| -                      | -   |
| Return value           |   |
| -                      | -   |

Example:

```

/* enable TIMER0 */

timer_enable (TIMER0);

```

## timer\_disable

The description of timer\_disable is shown as below:

**Table 3-450. 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..13)</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 3-451. Function timer\_auto\_reload\_shadow\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | timer_auto_reload_shadow_enable                              |
| <b>Function prototype</b>    | void timer_auto_reload_shadow_enable(uint32_t timer_periph); |
| <b>Function descriptions</b> | enable the auto reload shadow function                       |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>timer_periph</b>          | TIMER peripheral   |
| <i>TIMERx(x=0..13)</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 3-452. Function timer\_auto\_reload\_shadow\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | timer_auto_reload_shadow_disable                               |
| <b>Function prototype</b>    | void timer_auto_reload_shadow_disable (uint32_t timer_periph); |
| <b>Function descriptions</b> | disable the auto reload shadow function                        |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>timer_periph</b>          | TIMER peripheral   |
| <i>TIMERx(x=0..13)</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 3-453. 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..13)</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 3-454. 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..13)</i>       | TIMER peripheral selection                               |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable TIMER0 the update event */
timer_update_event_disable (TIMER0);
```

## timer\_counter\_alignment

The description of timer\_counter\_alignment is shown as below:

**Table 3-455. Function timer\_counter\_alignment**

|                                   |  |
|-----------------------------------|--|
| <b>Function name</b>              | timer_counter_alignment  |
| <b>Function prototype</b>         | void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);   |
| <b>Function descriptions</b>      | set TIMER counter alignment mode   |
| <b>Precondition</b>               | -  |
| <b>The called functions</b>       | -  |
| <b>Input parameter{in}</b>        |  |
| <b>timer_periph</b>               | TIMER peripheral   |
| <i>TIMERx(x=0..4,7..13)</i>       | TIMER peripheral selection   |
| <b>Input parameter{in}</b>        |  |
| <b>aligned</b>                    | alignment mode   |
| <i>TIMER_COUNTER_EDGE</i>         | No center-aligned mode (edge-aligned mode). The direction of the counter is unspecified by the DIR bit.  |
| <i>TIMER_COUNTER_COUNTER_DOWN</i> | Center-aligned and counting down assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in <i>TIMERx_CHCTL0</i> register). Only when the counter is counting down, compare interrupt flag of channels can be set. |
| <i>TIMER_COUNTER_COUNTER_UP</i>   | Center-aligned and counting up assert mode. The counter counts under   |



|   |  |
|---|--|
| <i>NTER_UP</i>                              | 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.  |
| <i>TIMER_COUNTER_CE</i><br><i>NTER_BOTH</i> | Center-aligned and counting up/down assert mode. The counter counts under center-aligned and channel is configured in output mode (CHxMS=00 in 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 3-456. 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..4,7..13)</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 3-457. 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..4,7..13)</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 3-458. 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..13)</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 3-459. 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=0,7)</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 3-460. 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..13)</i>       | TIMER peripheral selection  |
| <b>Input parameter{in}</b>   |   |
| <b>autoreload</b>            | the counter auto-reload value (0-0xFFFF)  |

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

Example:

```
/* configure TIMER autoreload register value */
timer_autoreload_value_config (TIMER0, 3000);
```

## timer\_counter\_value\_config

The description of timer\_counter\_value\_config is shown as below:

**Table 3-461. Function timer\_counter\_value\_config**

| Function name          | timer_counter_value_config  |
|------------------------|---|
| Function prototype     | void timer_counter_value_config(uint32_t timer_periph, uint16_t counter); |
| Function descriptions  | configure TIMER counter register value                                    |
| Precondition           | -   |
| The called functions   | -   |
| Input parameter{in}    |   |
| timer_periph           | TIMER peripheral  |
| <i>TIMERx(x=0..13)</i> | TIMER peripheral selection  |
| Input parameter{in}    |   |
| counter                | the counter value (0-0xFFFF)  |
| Output parameter{out}  |   |
| -                      | -   |
| Return value           |   |
| -                      | -   |

Example:

```
/* configure TIMER0 counter register value */
timer_counter_value_config (TIMER0);
```

## timer\_counter\_read

The description of timer\_counter\_read is shown as below:

**Table 3-462. Function timer\_counter\_read**

| Function name         | timer_counter_read                                  |
|-----------------------|---|
| Function prototype    | uint32_t timer_counter_read(uint32_t timer_periph); |
| Function descriptions | read TIMER counter value                            |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |

|                              |                              |
|------------------------------|------------------------------|
| <b>timer_periph</b>          | TIMER peripheral             |
| <i>TIMERx(x=0..13)</i>       | TIMER peripheral selection   |
| <b>Output parameter{out}</b> |                              |
| -                            | -                            |
| <b>Return value</b>          |                              |
| <b>uint32_t</b>              | counter value(0x0000~0xFFFF) |

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 3-463. 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..13)</i>       | TIMER peripheral selection                            |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>uint16_t</b>              | prescaler register value (0x0000~0xFFFF)              |

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 3-464. 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..11)</i>          | TIMER peripheral selection        |
| <b>Input parameter{in}</b>      |                                   |
| <b>spmode</b>                   | pulse mode                        |
| <i>TIMER_SP_MODE_SINGLE</i>     | single pulse mode                 |
| <i>TIMER_SP_MODE_REPETITIVE</i> | repetitive pulse mode             |
| <b>Output parameter{out}</b>    |                                   |
| -                               | -                                 |
| <b>Return value</b>             |                                   |
| -                               | -                                 |

Example:

```
/* configure TIMER0 single pulse mode */
```

```
timer_single_pulse_mode_config (TIMER0, TIMER_SP_MODE_SINGLE);
```

### timer\_update\_source\_config

The description of timer\_update\_source\_config is shown as below:

**Table 3-465. 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..13)</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: <ul style="list-style-type: none"> <li>- The UPG bit is set</li> <li>- The counter generates an overflow or underflow event</li> <li>- The slave mode controller generates an update event</li> </ul> |
| <i>TIMER_UPDATE_SRC_REGULAR</i> | Only counter overflow/underflow generates an update interrupt or DMA request.  |
| <b>Output parameter{out}</b>    |  |

|              |   |
|--------------|---|
| -            | - |
| Return value |   |
| -            | - |

Example:

```
/* configure TIMER update only by counter overflow/underflow */
```

```
timer_update_source_config (TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

## timer\_dma\_enable

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

**Table 3-466. Function timer\_dma\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_dma_enable  |
| <b>Function prototype</b>    | void timer_dma_enable(uint32_t timer_periph, uint16_t dma); |
| <b>Function descriptions</b> | enable the TIMER DMA  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>timer_periph</b>          | TIMER peripheral  |
| <i>TIMERx</i>                | please refer to the following parameters                    |
| <b>Input parameter{in}</b>   |   |
| <b>dma</b>                   | timer DMA source enable                                     |
| <i>TIMER_DMA_UPD</i>         | update DMA enable, TIMERx(x=0..7)                           |
| <i>TIMER_DMA_CH0D</i>        | channel 0 DMA enable, TIMERx(x=0..4,7)                      |
| <i>TIMER_DMA_CH1D</i>        | channel 1 DMA enable, TIMERx(x=0..4,7)                      |
| <i>TIMER_DMA_CH2D</i>        | channel 2 DMA enable, TIMERx(x=0..4,7)                      |
| <i>TIMER_DMA_CH3D</i>        | channel 3 DMA enable, TIMERx(x=0..4,7)                      |
| <i>TIMER_DMA_CMTD</i>        | commutation DMA request enable, TIMERx(x=0,7)               |
| <i>TIMER_DMA_TRGD</i>        | trigger DMA enable, TIMERx(x=0..4,7)                        |
| <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 3-467. Function timer\_dma\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_dma_disable   |
| <b>Function prototype</b>    | void timer_dma_disable (uint32_t timer_periph, uint16_t dma); |
| <b>Function descriptions</b> | disable the TIMER DMA   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>timer_periph</b>          | TIMER peripheral  |
| <i>TIMERx</i>                | please refer to the following parameters                      |
| <b>Input parameter{in}</b>   |   |
| <b>dma</b>                   | timer DMA source disable                                      |
| <i>TIMER_DMA_UPD</i>         | update DMA disable, TIMERx(x=0..7)                            |
| <i>TIMER_DMA_CH0D</i>        | channel 0 DMA disable, TIMERx(x=0..4,7)                       |
| <i>TIMER_DMA_CH1D</i>        | channel 1 DMA disable, TIMERx(x=0..4,7)                       |
| <i>TIMER_DMA_CH2D</i>        | channel 2 DMA disable, TIMERx(x=0..4,7)                       |
| <i>TIMER_DMA_CH3D</i>        | channel 3 DMA disable, TIMERx(x=0..4,7)                       |
| <i>TIMER_DMA_CMTD</i>        | commutation DMA request disable, TIMERx(x=0,7)                |
| <i>TIMER_DMA_TRGD</i>        | trigger DMA disable, TIMERx(x=0..4,7)                         |
| <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 3-468. Function timer\_channel\_dma\_request\_source\_select**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | timer_channel_dma_request_source_select  |
| <b>Function prototype</b>    | void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request); |
| <b>Function descriptions</b> | channel DMA request source selection   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>timer_periph</b>          | TIMER peripheral   |
| <i>TIMERx</i>                | TIMER peripheral selection   |
| <b>Input parameter{in}</b>   |  |
| <b>dma_request</b>           | channel DMA request source selection   |



|                                      |  |
|--------------------------------------|--|
| <i>TIMER_DMAREQUEST_CHANNELEVENT</i> | DMA request of channel n is sent when channel y event occurs |
| <i>TIMER_DMAREQUEST_UPDATEEVENT</i>  | DMA request of channel n is sent when update event occurs    |
| <b>Output parameter{out}</b>         |  |
| -                                    | -  |
| <b>Return value</b>                  |  |
| -                                    | -  |

Example:

```
/* TIMER0 channel DMA request of channel n is sent when channel y event occurs */
```

```
timer_channel_dma_request_source_select(TIMER0,  
TIMER_DMAREQUEST_CHANNELEVENT);
```

### timer\_dma\_transfer\_config

The description of timer\_dma\_transfer\_config is shown as below:

**Table 3-469. Function timer\_dma\_transfer\_config**

|                                     |   |
|-------------------------------------|---|
| <b>Function name</b>                | timer_dma_transfer_config   |
| <b>Function prototype</b>           | void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth); |
| <b>Function descriptions</b>        | configure the TIMER DMA transfer  |
| <b>Precondition</b>                 | -   |
| <b>The called functions</b>         | -   |
| <b>Input parameter{in}</b>          |   |
| <b>timer_periph</b>                 | TIMER peripheral  |
| <i>TIMERx</i>                       | please refer to the following parameters  |
| <b>Input parameter{in}</b>          |   |
| <b>dma_baseaddr</b>                 | DMA transfer access start address   |
| <i>TIMER_DMACFG_DMA_TA_CTL0</i>     | DMA transfer address is TIMER_CTL0, TIMERx(x=0..4,7)  |
| <i>TIMER_DMACFG_DMA_TA_CTL1</i>     | DMA transfer address is TIMER_CTL1, TIMERx(x=0..4,7)  |
| <i>TIMER_DMACFG_DMA_TA_SMCFG</i>    | DMA transfer address is TIMER_SMCFG, TIMERx(x=0..4,7)   |
| <i>TIMER_DMACFG_DMA_TA_DMAINTEN</i> | DMA transfer address is TIMER_DMAINTEN, TIMERx(x=0..4,7)  |
| <i>TIMER_DMACFG_DMA_TA_INTF</i>     | DMA transfer address is TIMER_INTF, TIMERx(x=0..4,7)  |
| <i>TIMER_DMACFG_DMA_TA_SWEVG</i>    | DMA transfer address is TIMER_SWEVG, TIMERx(x=0..4,7)   |
| <i>TIMER_DMACFG_DMA</i>             | DMA transfer address is TIMER_CHCTL0, TIMERx(x=0..4,7)  |

|  |   |
|--|---|
| <i>TA_CHCTL0</i>                         |   |
| <i>TIMER_DMACFG_DMA<br/>TA_CHCTL1</i>    | DMA transfer address is <i>TIMER_CHCTL1</i> , <i>TIMERx</i> ( <i>x</i> =0..4,7) |
| <i>TIMER_DMACFG_DMA<br/>TA_CHCTL2</i>    | DMA transfer address is <i>TIMER_CHCTL2</i> , <i>TIMERx</i> ( <i>x</i> =0..4,7) |
| <i>TIMER_DMACFG_DMA<br/>TA_CNT</i>       | DMA transfer address is <i>TIMER_CNT</i> , <i>TIMERx</i> ( <i>x</i> =0..4,7)    |
| <i>TIMER_DMACFG_DMA<br/>TA_PSC</i>       | DMA transfer address is <i>TIMER_PSC</i> , <i>TIMERx</i> ( <i>x</i> =0..4,7)    |
| <i>TIMER_DMACFG_DMA<br/>TA_CAR</i>       | MA transfer address is <i>TIMER_CAR</i> , <i>TIMERx</i> ( <i>x</i> =0..4,7)     |
| <i>TIMER_DMACFG_DMA<br/>TA_CREP</i>      | DMA transfer address is <i>TIMER_CREP</i> , <i>TIMERx</i> ( <i>x</i> =0,7)      |
| <i>TIMER_DMACFG_DMA<br/>TA_CH0CV</i>     | DMA transfer address is <i>TIMER_CH0CV</i> , <i>TIMERx</i> ( <i>x</i> =0..4,7)  |
| <i>TIMER_DMACFG_DMA<br/>TA_CH1CV</i>     | DMA transfer address is <i>TIMER_CH1CV</i> , <i>TIMERx</i> ( <i>x</i> =0..4,7)  |
| <i>TIMER_DMACFG_DMA<br/>TA_CH2CV</i>     | DMA transfer address is <i>TIMER_CH2CV</i> , <i>TIMERx</i> ( <i>x</i> =0..4,7)  |
| <i>TIMER_DMACFG_DMA<br/>TA_CH3CV</i>     | DMA transfer address is <i>TIMER_CH3CV</i> , <i>TIMERx</i> ( <i>x</i> =0..4,7)  |
| <i>TIMER_DMACFG_DMA<br/>TA_CCHP</i>      | DMA transfer address is <i>TIMER_CCHP</i> , <i>TIMERx</i> ( <i>x</i> =0,7)      |
| <i>TIMER_DMACFG_DMA<br/>TA_DMACFG</i>    | DMA transfer address is <i>TIMER_DMACFG</i> , <i>TIMERx</i> ( <i>x</i> =0..4,7) |
| <i>TIMER_DMACFG_DMA<br/>TA_DMATB</i>     | DMA transfer address is <i>TIMER_DMATB</i> , <i>TIMERx</i> ( <i>x</i> =0..4,7)  |
| <b>Input parameter{in}</b>               |   |
| <b>dma_lenth</b>                         | DMA transfer count  |
| <i>TIMER_DMACFG_DMA<br/>TC_xTRANSFER</i> | <i>x</i> =1..18, DMA transfer <i>x</i> time                                     |
| <b>Output parameter{out}</b>             |   |
| -  | -   |
| <b>Return value</b>                      |   |
| -  | -   |

Example:

```
/* configure the TIMER0 DMA transfer */
```

```
timer_dma_transfer_config(TIMER0, TIMER_DMACFG_DMATA_CTL0,  
TIMER_DMACFG_DMATC_5TRANSFER);
```

## timer\_event\_software\_generate

The description of timer\_event\_software\_generate is shown as below:

**Table 3-470. Function timer\_event\_software\_generate**

|                               |  |
|-------------------------------|--|
| <b>Function name</b>          | timer_event_software_generate  |
| <b>Function prototype</b>     | void timer_event_software_generate(uint32_t timer_periph, uint16_t event); |
| <b>Function descriptions</b>  | software generate events   |
| <b>Precondition</b>           | -  |
| <b>The called functions</b>   | -  |
| <b>Input parameter{in}</b>    |  |
| <b>timer_periph</b>           | TIMER peripheral   |
| <i>TIMERx</i>                 | please refer to the following parameters                                   |
| <b>Input parameter{in}</b>    |  |
| <b>event</b>                  | the timer software event generation sources                                |
| <i>TIMER_EVENT_SRC_UPG</i>    | update event, TIMERx(x=0..13)  |
| <i>TIMER_EVENT_SRC_CH0G</i>   | channel 0 capture or compare event generation, TIMERx(x=0..4,7..13)        |
| <i>TIMER_EVENT_SRC_CH1G</i>   | channel 1 capture or compare event generation, TIMERx(x=0..4,7,8,11)       |
| <i>TIMER_EVENT_SRC_CH2G</i>   | channel 2 capture or compare event generation, TIMERx(x=0..4,7)            |
| <i>TIMER_EVENT_SRC_CH3G</i>   | channel 3 capture or compare event generation, TIMERx(x=0..4,7)            |
| <i>TIMER_EVENT_SRC_CMTG</i>   | channel commutation event generation, TIMERx(x=0,7)                        |
| <i>TIMER_EVENT_SRC_TRIGG</i>  | trigger event generation, TIMERx(x=0..4,7,8,11)                            |
| <i>TIMER_EVENT_SRC_BREAKG</i> | break event generation, TIMERx(x=0,7)                                      |
| <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 3-471. Function timer\_break\_struct\_para\_init**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_break_struct_para_init  |
| <b>Function prototype</b>    | void timer_break_struct_para_init(timer_break_parameter_struct* breakpara);   |
| <b>Function descriptions</b> | initialize the parameters of TIMER break parameter struct with the default values   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>breakpara</b>             | TIMER break parameter struct, the structure members can refer to <a href="#">Table 3-443. 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 3-472. 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=0,7)</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 3-443. 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 3-473. Function timer\_break\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | timer_break_enable   |
| <b>Function prototype</b>    | void timer_break_enable(uint32_t timer_periph);  |
| <b>Function descriptions</b> | enable TIMER break function  |
| <b>Precondition</b>          | This function can be called only when PROT [1:0] bit-filed in<br>TIMERx_CCHP register is 00. |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>timer_periph</b>          | TIMER peripheral   |
| <i>TIMERx(x=0,7)</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 3-474. 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<br>TIMERx_CCHP register is 00. |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>timer_periph</b>          | TIMER peripheral   |
| <i>TIMERx(x=0,7)</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 3-475. 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<br>TIMERx_CCHP register is 00. |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>timer_periph</b>          | TIMER peripheral   |
| <i>TIMERx(x=0,7)</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 3-476. Function timer\_automatic\_output\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | timer_automatic_output_disable   |
| <b>Function prototype</b>    | void timer_automatic_output_disable (uint32_t timer_periph);                                 |
| <b>Function descriptions</b> | disable TIMER output automatic function  |
| <b>Precondition</b>          | This function can be called only when PROT [1:0] bit-filed in<br>TIMERx_CCHP register is 00. |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>timer_periph</b>          | TIMER peripheral   |
| <i>TIMERx(x=0,7)</i>         | TIMER peripheral selection   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable TIMERO0 output automatic function */
```

```
timer_automatic_output_disable (TIMER0);
```

## timer\_primary\_output\_config

The description of timer\_primary\_output\_config is shown as below:

**Table 3-477. 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<br>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=0,7)</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 3-478. 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=0,7)</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 3-479. 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>  | -   |



| Input parameter{in}          |  |
|------------------------------|--|
| <b>timer_periph</b>          | TIMER peripheral   |
| <i>TIMERx(x=0,7)</i>         | TIMER peripheral selection   |
| Input parameter{in}          |  |
| <b>ccuctl</b>                | channel control shadow register update control                                       |
| <i>TIMER_UPDATECTL_CCU</i>   | the shadow registers update by when CMTG bit is set                                  |
| <i>TIMER_UPDATECTL_CUTRI</i> | the shadow registers update by when CMTG bit is set or an rising edge of TRGI occurs |
| Output parameter{out}        |  |
| -                            | -  |
| Return value                 |  |
| -                            | -  |

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 3-480. Function timer\_channel\_output\_struct\_para\_init**

| <b>Function name</b>         | timer_channel_output_struct_para_init  |
|------------------------------|--|
| <b>Function prototype</b>    | void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpa);   |
| <b>Function descriptions</b> | initialize the parameters of TIMER channel output parameter struct with the default values   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| Input parameter{in}          |  |
| <b>ocpara</b>                | TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-444. 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 3-481. Function timer\_channel\_output\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_channel_output_config   |
| <b>Function prototype</b>    | void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara);                               |
| <b>Function descriptions</b> | configure TIMER channel output function   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>timer_periph</b>          | TIMER peripheral  |
| <i>TIMERx</i>                | please refer to the following parameters  |
| <b>Input parameter{in}</b>   |   |
| <b>channel</b>               | channel to be configured  |
| <i>TIMER_CH_0</i>            | TIMER channel 0 (TIMERx(x=0..4,7..13))  |
| <i>TIMER_CH_1</i>            | TIMER channel 1 (TIMERx(x=0..4,7,8,11))   |
| <i>TIMER_CH_2</i>            | TIMER channel 2 (TIMERx(x=0..4,7))  |
| <i>TIMER_CH_3</i>            | IMER channel 3 (TIMERx(x=0..4,7))   |
| <b>Input parameter{in}</b>   |   |
| <b>ocpara</b>                | TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-444. Structure timer_oc_parameter_struct.</a> |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* configure TIMER0 channel 0 output function */

timer_oc_parameter_struct timer_ocinitpara;

timer_ocinitpara.outputstate = TIMER_CCX_ENABLE;

timer_ocinitpara.outputnstate = TIMER_CCXN_ENABLE;

timer_ocinitpara.ocpolarity = TIMER_OC_POLARITY_HIGH;

timer_ocinitpara.ocnpolarity = TIMER_OCN_POLARITY_HIGH;

timer_ocinitpara.ocidlestate = TIMER_OC_IDLE_STATE_HIGH;

timer_ocinitpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;

timer_channel_output_config(TIMER0, TIMER_CH_0, &timer_ocinitpara);
```

## timer\_channel\_output\_mode\_config

The description of timer\_channel\_output\_mode\_config is shown as below:

**Table 3-482. Function timer\_channel\_output\_mode\_config**

|                               |  |
|-------------------------------|--|
| <b>Function name</b>          | timer_channel_output_mode_config   |
| <b>Function prototype</b>     | void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint16_t ocmode); |
| <b>Function descriptions</b>  | configure TIMER channel output compare mode  |
| <b>Precondition</b>           | -  |
| <b>The called functions</b>   | -  |
| <b>Input parameter{in}</b>    |  |
| <b>timer_periph</b>           | TIMER peripheral   |
| <i>TIMERx</i>                 | please refer to the following parameters   |
| <b>Input parameter{in}</b>    |  |
| <b>channel</b>                | channel to be configured   |
| <i>TIMER_CH_0</i>             | TIMER channel 0 (TIMERx(x=0..4,7..13))   |
| <i>TIMER_CH_1</i>             | TIMER channel 1 (TIMERx(x=0..4,7,8,11))  |
| <i>TIMER_CH_2</i>             | TIMER channel 2 (TIMERx(x=0..4,7))   |
| <i>TIMER_CH_3</i>             | IMER channel 3 (TIMERx(x=0..4,7))  |
| <b>Input parameter{in}</b>    |  |
| <b>ocmode</b>                 | channel output compare mode  |
| <i>TIMER_OC_MODE_TIMING</i>   | timing mode  |
| <i>TIMER_OC_MODE_ACTIVE</i>   | set the channel output   |
| <i>TIMER_OC_MODE_INACTIVE</i> | clear the channel output   |
| <i>TIMER_OC_MODE_TOGGLE</i>   | toggle on match  |
| <i>TIMER_OC_MODE_LOW</i>      | force low mode   |
| <i>TIMER_OC_MODE_HIGH</i>     | force high mode  |
| <i>TIMER_OC_MODE_PWM0</i>     | PWM mode 0   |
| <i>TIMER_OC_MODE_PWM1</i>     | PWM mode 1   |
| <b>Output parameter{out}</b>  |  |
| -                             | -  |
| <b>Return value</b>           |  |
| -                             | -  |

Example:

```
/* configure TIMER0 channel PWM 0 mode */
```

```
timer_channel_output_mode_config(TIMER0, TIMER_CH_0, TIMER_OC_MODE_PWM0);
```

## timer\_channel\_output\_pulse\_value\_config

The description of timer\_channel\_output\_pulse\_value\_config is shown as below:

**Table 3-483. 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..4,7..13))   |
| <i>TIMER_CH_1</i>            | TIMER channel 1 (TIMERx(x=0..4,7,8,11))  |
| <i>TIMER_CH_2</i>            | TIMER channel 2 (TIMERx(x=0..4,7))   |
| <i>TIMER_CH_3</i>            | IMER channel 3 (TIMERx(x=0..4,7))  |
| <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 3-484. Function timer\_channel\_output\_shadow\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | timer_channel_output_shadow_config   |
| <b>Function prototype</b>    | void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow); |
| <b>Function descriptions</b> | configure TIMER channel output shadow function   |
| <b>Precondition</b>          | -  |

|                                |  |
|--------------------------------|--|
| <b>The called functions</b>    | -  |
| <b>Input parameter{in}</b>     |  |
| <b>timer_periph</b>            | TIMER peripheral                         |
| <i>TIMERx</i>                  | please refer to the following parameters |
| <b>Input parameter{in}</b>     |  |
| <b>channel</b>                 | channel to be configured                 |
| <i>TIMER_CH_0</i>              | TIMER channel 0 (TIMERx(x=0..4,7..13))   |
| <i>TIMER_CH_1</i>              | TIMER channel 1 (TIMERx(x=0..4,7,8,11))  |
| <i>TIMER_CH_2</i>              | TIMER channel 2 (TIMERx(x=0..4,7))       |
| <i>TIMER_CH_3</i>              | IMER channel 3 (TIMERx(x=0..4,7))        |
| <b>Input parameter{in}</b>     |  |
| <b>ocshadow</b>                | channel output shadow state              |
| <i>TIMER_OC_SHADOW_ENABLE</i>  | channel output shadow state enable       |
| <i>TIMER_OC_SHADOW_DISABLE</i> | channel output shadow state disable      |
| <b>Output parameter{out}</b>   |  |
| -                              | -  |
| <b>Return value</b>            |  |
| -                              | -  |

Example:

```
/*configure TIMER0 channel 0 output shadow function */
```

```
timer_channel_output_shadow_config (TIMER0, TIMER_CH_0,  
TIMER_OC_SHADOW_ENABLE);
```

## timer\_channel\_output\_fast\_config

The description of timer\_channel\_output\_fast\_config is shown as below:

**Table 3-485. Function timer\_channel\_output\_fast\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | timer_channel_output_fast_config   |
| <b>Function prototype</b>    | void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast); |
| <b>Function descriptions</b> | configure TIMER channel output fast function   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>timer_periph</b>          | TIMER peripheral   |
| <i>TIMERx</i>                | please refer to the following parameters   |
| <b>Input parameter{in}</b>   |  |
| <b>channel</b>               | channel to be configured   |
| <i>TIMER_CH_0</i>            | TIMER channel 0 (TIMERx(x=0..4,7..13))   |

|                              |   |
|------------------------------|---|
| <i>TIMER_CH_1</i>            | TIMER channel 1 (TIMERx(x=0..4,7,8,11)) |
| <i>TIMER_CH_2</i>            | TIMER channel 2 (TIMERx(x=0..4,7))      |
| <i>TIMER_CH_3</i>            | TIMER channel 3 (TIMERx(x=0..4,7))      |
| <b>Input parameter{in}</b>   |   |
| <b>ocfast</b>                | channel output fast function            |
| <i>TIMER_OC_FAST_ENABLE</i>  | channel output fast function enable     |
| <i>TIMER_OC_FAST_DISABLE</i> | channel output fast function disable    |
| <b>Output parameter{out}</b> |   |
| -                            | -                                       |
| <b>Return value</b>          |   |
| -                            | -                                       |

Example:

```
/* configure TIMER0 channel 0 output fast function */
```

```
timer_channel_output_fast_config (TIMER0, TIMER_CH_0, TIMER_OC_FAST_ENABLE);
```

## timer\_channel\_output\_clear\_config

The description of timer\_channel\_output\_clear\_config is shown as below:

**Table 3-486. 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 periph   |
| <i>TIMERx</i>                 | please refer to the following parameters   |
| <b>Input parameter{in}</b>    |  |
| <b>channel</b>                | channel to be configured   |
| <i>TIMER_CH_0</i>             | TIMER channel 0 (TIMERx(x=0..4,7..13))   |
| <i>TIMER_CH_1</i>             | TIMER channel 1 (TIMERx(x=0..4,7,8,11))  |
| <i>TIMER_CH_2</i>             | TIMER channel 2 (TIMERx(x=0..4,7))   |
| <i>TIMER_CH_3</i>             | TIMER channel 3 (TIMERx(x=0..4,7))   |
| <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  |

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

Example:

```
/* configure TIMER0 channel 0 output clear function */
```

```
timer_channel_output_clear_config (TIMER0, TIMER_CH_0,  
TIMER_OC_CLEAR_ENABLE);
```

## timer\_channel\_output\_polarity\_config

The description of timer\_channel\_output\_polarity\_config is shown as below:

**Table 3-487. Function timer\_channel\_output\_polarity\_config**

|                               |  |
|-------------------------------|--|
| <b>Function name</b>          | timer_channel_output_polarity_config   |
| <b>Function prototype</b>     | void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity); |
| <b>Function descriptions</b>  | configure TIMER channel output polarity  |
| <b>Precondition</b>           | -  |
| <b>The called functions</b>   | -  |
| <b>Input parameter{in}</b>    |  |
| <b>timer_periph</b>           | TIMER peripheral   |
| <i>TIMERx</i>                 | please refer to the following parameters   |
| <b>Input parameter{in}</b>    |  |
| <b>channel</b>                | channel to be configured   |
| <i>TIMER_CH_0</i>             | TIMER channel 0 (TIMERx(x=0..4,7..13))   |
| <i>TIMER_CH_1</i>             | TIMER channel 1 (TIMERx(x=0..4,7,8,11))  |
| <i>TIMER_CH_2</i>             | TIMER channel 2 (TIMERx(x=0..4,7))   |
| <i>TIMER_CH_3</i>             | IMER channel 3 (TIMERx(x=0..4,7))  |
| <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 3-488. 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(x=0,7)</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  |
| <i>TIMER_CH_2</i>              | TIMER channel 2  |
| <b>Input parameter{in}</b>     |  |
| <b>ocpolarity</b>              | channel complementary output polarity  |
| <i>TIMER_OCN_POLARITY_HIGH</i> | channel complementary output polarity is high  |
| <i>TIMER_OCN_POLARITY_LOW</i>  | channel complementary output polarity is low   |
| <b>Output parameter{out}</b>   |  |
| -                              | -  |
| <b>Return value</b>            |  |
| -                              | -  |

Example:

```
/* configure TIMER0 channel 0 complementary output polarity */
```

```
timer_channel_complementary_output_polarity_config (TIMER0, TIMER_CH_0,  
TIMER_OCN_POLARITY_HIGH);
```

## timer\_channel\_output\_state\_config

The description of timer\_channel\_output\_state\_config is shown as below:



Table 3-489. Function timer\_channel\_output\_state\_config

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | timer_channel_output_state_config  |
| <b>Function prototype</b>    | void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state); |
| <b>Function descriptions</b> | configure TIMER channel enable state   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>timer_periph</b>          | TIMER peripheral   |
| <i>TIMERx</i>                | please refer to the following parameters   |
| <b>Input parameter{in}</b>   |  |
| <b>channel</b>               | channel to be configured   |
| <i>TIMER_CH_0</i>            | TIMER channel 0 (TIMERx(x=0..4,7..13))   |
| <i>TIMER_CH_1</i>            | TIMER channel 1 (TIMERx(x=0..4,7,8,11))  |
| <i>TIMER_CH_2</i>            | TIMER channel 2 (TIMERx(x=0..4,7))   |
| <i>TIMER_CH_3</i>            | IMER channel 3 (TIMERx(x=0..4,7))  |
| <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 3-490. Function timer\_channel\_complementary\_output\_state\_config

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_channel_complementary_output_state_config   |
| <b>Function prototype</b>    | void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate); |
| <b>Function descriptions</b> | configure TIMER channel complementary output enable state   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>timer_periph</b>          | TIMER peripheral  |
| <i>TIMERx(x=0,7)</i>         | TIMER peripheral selection  |

| Input parameter{in}       |   |
|---------------------------|---|
| <b>channel</b>            | channel to be configured                        |
| <i>TIMER_CH_0</i>         | TIMER channel 0                                 |
| <i>TIMER_CH_1</i>         | TIMER channel 1                                 |
| <i>TIMER_CH_2</i>         | TIMER channel 2                                 |
| Input parameter{in}       |   |
| <b>state</b>              | TIMER channel complementary output enable state |
| <i>TIMER_CCXN_ENABLE</i>  | channel complementary enable                    |
| <i>TIMER_CCXN_DISABLE</i> | channel complementary disable                   |
| Output parameter{out}     |   |
| -                         | -   |
| Return value              |   |
| -                         | -   |

Example:

```
/* configure TIMER0 channel 0 complementary output enable state */
```

```
timer_channel_complementary_output_state_config (TIMER0, TIMER_CH_0,  
TIMER_CCXN_ENABLE);
```

## timer\_channel\_input\_struct\_para\_init

The description of timer\_channel\_input\_struct\_para\_init is shown as below:

**Table 3-491. 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>  | -   |
| Input parameter{in}          |   |
| <b>icpara</b>                | TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-445. Structure timer_ic_parameter_struct</a> . |
| Output parameter{out}        |   |
| -                            | -   |
| Return value                 |   |
| -                            | -   |

Example:

```
/* initialize TIMER channel input parameter struct with a default value */
```

```
timer_ic_parameter_struct timer_icinitpara;
```

```
timer_channel_input_struct_para_init(timer_icinitpara);
```

## timer\_input\_capture\_config

The description of timer\_input\_capture\_config is shown as below:

**Table 3-492. Function timer\_input\_capture\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_input_capture_config  |
| <b>Function prototype</b>    | void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);                                |
| <b>Function descriptions</b> | configure TIMER input capture parameter   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | timer_channel_input_capture_prescaler_config  |
| <b>Input parameter{in}</b>   |   |
| <b>timer_periph</b>          | TIMER peripheral  |
| <i>TIMERx</i>                | please refer to the following parameters  |
| <b>Input parameter{in}</b>   |   |
| <b>channel</b>               | channel to be configured  |
| <i>TIMER_CH_0</i>            | TIMER channel 0 (TIMERx(x=0..4,7..13))  |
| <i>TIMER_CH_1</i>            | TIMER channel 1 (TIMERx(x=0..4,7,8,11))   |
| <i>TIMER_CH_2</i>            | TIMER channel 2 (TIMERx(x=0..4,7))  |
| <i>TIMER_CH_3</i>            | IMER channel 3 (TIMERx(x=0..4,7))   |
| <b>Input parameter{in}</b>   |   |
| <b>icpara</b>                | TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-445. Structure timer_ic_parameter_struct</a> . |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* configure TIMER0 input capture parameter */
timer_ic_parameter_struct timer_icinitpara;
timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;
timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);
```

## timer\_channel\_input\_capture\_prescaler\_config

The description of timer\_channel\_input\_capture\_prescaler\_config is shown as below:

**Table 3-493. Function timer\_channel\_input\_capture\_prescaler\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_channel_input_capture_prescaler_config  |
| <b>Function prototype</b>    | void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler); |
| <b>Function descriptions</b> | configure TIMER channel input capture prescaler value   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>timer_periph</b>          | TIMER peripheral  |
| <i>TIMERx</i>                | please refer to the following parameters  |
| <b>Input parameter{in}</b>   |   |
| <b>channel</b>               | channel to be configured  |
| <i>TIMER_CH_0</i>            | TIMER channel 0 (TIMERx(x=0..4,7..13))  |
| <i>TIMER_CH_1</i>            | TIMER channel 1 (TIMERx(x=0..4,7,8,11))   |
| <i>TIMER_CH_2</i>            | TIMER channel 2 (TIMERx(x=0..4,7))  |
| <i>TIMER_CH_3</i>            | IMER channel 3 (TIMERx(x=0..4,7))   |
| <b>Input parameter{in}</b>   |   |
| <b>prescaler</b>             | channel input capture prescaler value   |
| <i>TIMER_IC_PSC_DIV1</i>     | no prescaler  |
| <i>TIMER_IC_PSC_DIV2</i>     | divided by 2  |
| <i>TIMER_IC_PSC_DIV4</i>     | divided by 4  |
| <i>TIMER_IC_PSC_DIV8</i>     | divided by 8  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* configure TIMER0 channel 0 input capture prescaler value */
```

```
timer_channel_input_capture_prescaler_config (TIMER0, TIMER_CH_0,  
TIMER_IC_PSC_DIV2);
```

## timer\_channel\_capture\_value\_register\_read

The description of timer\_channel\_capture\_value\_register\_read is shown as below:

**Table 3-494. Function timer\_channel\_capture\_value\_register\_read**

|                           |  |
|---------------------------|--|
| <b>Function name</b>      | timer_channel_capture_value_register_read  |
| <b>Function prototype</b> | uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel); |

|                              |  |
|------------------------------|--|
| <b>Function descriptions</b> | read TIMER channel capture compare register value      |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>timer_periph</b>          | TIMER peripheral                                       |
| <i>TIMERx</i>                | please refer to the following parameters               |
| <b>Input parameter{in}</b>   |  |
| <b>channel</b>               | channel to be configured                               |
| <i>TIMER_CH_0</i>            | TIMER channel 0 (TIMERx(x=0..4,7..13))                 |
| <i>TIMER_CH_1</i>            | TIMER channel 1 (TIMERx(x=0..4,7,8,11))                |
| <i>TIMER_CH_2</i>            | TIMER channel 2 (TIMERx(x=0..4,7))                     |
| <i>TIMER_CH_3</i>            | IMER channel 3 (TIMERx(x=0..4,7))                      |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>uint32_t</b>              | channel capture compare register value (0x0000~0xFFFF) |

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 3-495. Function timer\_input\_pwm\_capture\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | timer_input_pwm_capture_config   |
| <b>Function prototype</b>    | void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);                                |
| <b>Function descriptions</b> | configure TIMER input pwm capture function   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | timer_channel_input_capture_prescaler_config   |
| <b>Input parameter{in}</b>   |  |
| <b>timer_periph</b>          | TIMER peripheral   |
| <i>TIMERx(x=0..4,7,8,11)</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 3-445. Structure timer_ic_parameter_struct.</a> |

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

Example:

```

/* configure TIMER0 input pwm capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_pwm_capture_config (TIMER0, TIMER_CH_0, &timer_icinitpara);

```

## timer\_hall\_mode\_config

The description of timer\_hall\_mode\_config is shown as below:

**Table 3-496. Function timer\_hall\_mode\_config**

| Function name                   | timer_hall_mode_config  |
|---------------------------------|---|
| Function prototype              | void timer_hall_mode_config(uint32_t timer_periph, uint8_t hallmode); |
| Function descriptions           | configure TIMER hall sensor mode                                      |
| Precondition                    | -   |
| The called functions            | -   |
| Input parameter{in}             |   |
| timer_periph                    | TIMER peripheral  |
| TIMERx(x=0..4,7)                | TIMER peripheral selection  |
| Input parameter{in}             |   |
| hallmode                        | TIMER hall sensor mode state  |
| TIMER_HALLINTERFA<br>CE_ENABLE  | TIMER hall sensor mode enable   |
| TIMER_HALLINTERFA<br>CE_DISABLE | TIMER hall sensor mode disable  |
| Output parameter{out}           |   |
| -                               | -   |
| Return value                    |   |
| -                               | -   |

Example:

```

/* configure TIMER0 hall sensor mode */

```

timer\_hall\_mode\_config (TIMER0, TIMER\_HALLINTERFACE\_ENABLE);

## timer\_input\_trigger\_source\_select

The description of timer\_input\_trigger\_source\_select is shown as below:

**Table 3-497. Function timer\_input\_trigger\_source\_select**

|  |  |
|--|--|
| <b>Function name</b>                   | timer_input_trigger_source_select  |
| <b>Function prototype</b>              | void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger); |
| <b>Function descriptions</b>           | select TIMER input trigger source  |
| <b>Precondition</b>                    | SMC[2:0] = 000   |
| <b>The called functions</b>            | -  |
| <b>Input parameter{in}</b>             |  |
| <b>timer_periph</b>                    | TIMER peripheral   |
| <i>TIMERx</i>                          | please refer to the following parameters   |
| <b>Input parameter{in}</b>             |  |
| <b>intrigger</b>                       | trigger selection  |
| <i>TIMER_SMCFG_TRGS<br/>EL_ITI0</i>    | Internal trigger input 0 (ITI0, TIMERx(x=0..4,7,8,11) )                            |
| <i>TIMER_SMCFG_TRGS<br/>EL_ITI1</i>    | Internal trigger input 0 (ITI1, TIMERx(x=0..4,7,8,11) )                            |
| <i>TIMER_SMCFG_TRGS<br/>EL_ITI2</i>    | Internal trigger input 0 (ITI2, TIMERx(x=0..4,7,8,11) )                            |
| <i>TIMER_SMCFG_TRGS<br/>EL_ITI3</i>    | Internal trigger input 0 (ITI3, TIMERx(x=0..4,7,8,11) )                            |
| <i>TIMER_SMCFG_TRGS<br/>EL_CIOF_ED</i> | CIO edge flag (CIOF_ED, TIMERx(x=0..4,7,8,11) )                                    |
| <i>TIMER_SMCFG_TRGS<br/>EL_CIOFE0</i>  | channel 0 input Filtered output (CIOFE0, TIMERx(x=0..4,7,8,11) )                   |
| <i>TIMER_SMCFG_TRGS<br/>EL_C1FE1</i>   | channel 1 input Filtered output (C1FE1, TIMERx(x=0..4,7,8,11) )                    |
| <i>TIMER_SMCFG_TRGS<br/>EL_ETIFP</i>   | External trigger input filter output(ETIFP, TIMERx(x=0..4,7) )                     |
| <b>Output parameter{out}</b>           |  |
| -                                      | -  |
| <b>Return value</b>                    |  |
| -                                      | -  |

Example:

```
/* select TIMER0 input trigger source */
```

```
timer_input_trigger_source_select (TIMER0, TIMER_SMCFG_TRGSEL_ITI0);
```

## timer\_master\_output\_trigger\_source\_select

The description of timer\_master\_output\_trigger\_source\_select is shown as below:

**Table 3-498. Function timer\_master\_output\_trigger\_source\_select**

|                                 |   |
|---------------------------------|---|
| <b>Function name</b>            | timer_master_output_trigger_source_select   |
| <b>Function prototype</b>       | void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);  |
| <b>Function descriptions</b>    | select TIMER master mode output trigger source  |
| <b>Precondition</b>             | -   |
| <b>The called functions</b>     | -   |
| <b>Input parameter{in}</b>      |   |
| <b>timer_periph</b>             | TIMER peripheral  |
| <i>TIMERx(x=0,7)</i>            | TIMER peripheral selection  |
| <b>Input parameter{in}</b>      |   |
| <b>outrigger</b>                | master mode control   |
| <i>TIMER_TRI_OUT_SRC_RESET</i>  | Reset. When the UPG bit in the <i>TIMERx_SWEVG</i> register is set or a reset is generated by the slave mode controller, a TRGO pulse occurs. And in the latter case, the signal on TRGO is delayed compared to the actual reset  |
| <i>TIMER_TRI_OUT_SRC_ENABLE</i> | Enable. This mode is useful to start several timers at the same time or to control a window in which a slave timer is enabled. In this mode the master mode controller selects the counter enable signal as TRGO. The counter enable signal is set when CEN control bit is set or the trigger input in pause mode is high. There is a delay between the trigger input in pause mode and the TRGO output, except if the master-slave mode is selected. |
| <i>TIMER_TRI_OUT_SRC_UPDATE</i> | Update. In this mode the master mode controller selects the update event as TRGO.   |
| <i>TIMER_TRI_OUT_SRC_CC0</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 3-499. 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..4,7,8,11)</i>      | TIMER peripheral selection   |
| <b>Input parameter{in}</b>        |  |
| <b>slavemode</b>                  | slave mode   |
| <i>TIMER_SLAVE_MODE_DISABLE</i>   | slave mode disable   |
| <i>TIMER_QUAD_DECODER_MODE0</i>   | quadrature decoder mode 0  |
| <i>TIMER_QUAD_DECODER_MODE1</i>   | quadrature decoder mode 1  |
| <i>TIMER_QUAD_DECODER_MODE2</i>   | quadrature decoder mode 2  |
| <i>TIMER_SLAVE_MODE_RESTART</i>   | restart mode   |
| <i>TIMER_SLAVE_MODE_PAUSE</i>     | pause mode   |
| <i>TIMER_SLAVE_MODE_EVENT</i>     | event mode   |
| <i>TIMER_SLAVE_MODE_EXTERNAL0</i> | external clock mode 0  |
| <b>Output parameter{out}</b>      |  |
| -                                 | -  |
| <b>Return value</b>               |  |
| -                                 | -  |

Example:

```
/* select TIMER0 slave mode */
```

```
timer_slave_mode_select (TIMER0, TIMER_QUAD_DECODER_MODE0);
```

## timer\_master\_slave\_mode\_config

The description of timer\_master\_slave\_mode\_config is shown as below:

**Table 3-500. Function timer\_master\_slave\_mode\_config**

|  |  |
|--|--|
| <b>Function name</b>                   | timer_master_slave_mode_config   |
| <b>Function prototype</b>              | void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t masterslave); |
| <b>Function descriptions</b>           | configure TIMER master slave mode  |
| <b>Precondition</b>                    | -  |
| <b>The called functions</b>            | -  |
| <b>Input parameter{in}</b>             |  |
| <b>timer_periph</b>                    | TIMER peripheral   |
| <i>TIMERx(x=0..4,7,8,11)</i>           | 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 3-501. 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..4,7)</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 3-502. 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,<br>uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity); |
| <b>Function descriptions</b>    | configure TIMER quadrature decoder mode   |
| <b>Precondition</b>             | -   |
| <b>The called functions</b>     | -   |
| <b>Input parameter{in}</b>      |   |
| <b>timer_periph</b>             | TIMER peripheral  |
| <i>TIMERx(x=0..4,7)</i>         | TIMER peripheral selection  |
| <b>Input parameter{in}</b>      |   |
| <b>decomode</b>                 | quadrature decoder mode   |
| <i>TIMER_QUAD_DECODER_MODE0</i> | counter counts on CI0FE0 edge depending on CI1FE1 level   |

|                                  |  |
|----------------------------------|--|
| <i>TIMER_QUAD_DECODER_MODE1</i>  | counter counts on CI1FE1 edge depending on CI0FE0 level                                  |
| <i>TIMER_QUAD_DECODER_MODE2</i>  | counter counts on both CI0FE0 and CI1FE1 edges depending on the level of the other input |
| <b>Input parameter{in}</b>       |  |
| <b>ic0polarity</b>               | IC0 polarity   |
| <i>TIMER_IC_POLARITY_RISING</i>  | capture rising edge  |
| <i>TIMER_IC_POLARITY_FALLING</i> | capture falling edge   |
| <b>Input parameter{in}</b>       |  |
| <b>ic1polarity</b>               | IC1 polarity   |
| <i>TIMER_IC_POLARITY_RISING</i>  | capture rising edge  |
| <i>TIMER_IC_POLARITY_FALLING</i> | capture falling edge   |
| <b>Output parameter{out}</b>     |  |
| -                                | -  |
| <b>Return value</b>              |  |
| -                                | -  |

Example:

```
/* configure TIMER0 quadrature decoder mode */
```

```
timer_quadrature_decoder_mode_config (TIMER0, TIMER_QUAD_DECODER_MODE0,
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

## timer\_internal\_clock\_config

The description of timer\_internal\_clock\_config is shown as below:

**Table 3-503. Function timer\_internal\_clock\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | timer_internal_clock_config                              |
| <b>Function prototype</b>    | void timer_internal_clock_config(uint32_t timer_periph); |
| <b>Function descriptions</b> | configure TIMER internal clock mode                      |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>timer_periph</b>          | TIMER peripheral   |
| <i>TIMERx(x=0..4,7,8,11)</i> | TIMER peripheral selection                               |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure TIMER0 internal clock mode */
```

```
timer_internal_clock_config (TIMER0);
```

## timer\_internal\_trigger\_as\_external\_clock\_config

The description of timer\_internal\_trigger\_as\_external\_clock\_config is shown as below:

**Table 3-504. Function timer\_internal\_trigger\_as\_external\_clock\_config**

|                                 |  |
|---------------------------------|--|
| <b>Function name</b>            | timer_internal_trigger_as_external_clock_config  |
| <b>Function prototype</b>       | void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger); |
| <b>Function descriptions</b>    | configure TIMER the internal trigger as external clock input                                     |
| <b>Precondition</b>             | -  |
| <b>The called functions</b>     | timer_input_trigger_source_select  |
| <b>Input parameter{in}</b>      |  |
| <b>timer_periph</b>             | TIMER peripheral   |
| <i>TIMERx(x=0..4,7,8,11)</i>    | TIMER peripheral selection   |
| <b>Input parameter{in}</b>      |  |
| <b>intrigger</b>                | trigger selection  |
| <i>TIMER_SMCFG_TRGS_EL_ITI0</i> | Internal trigger input 0 (ITI0)  |
| <i>TIMER_SMCFG_TRGS_EL_ITI1</i> | Internal trigger input 0 (ITI1)  |
| <i>TIMER_SMCFG_TRGS_EL_ITI2</i> | Internal trigger input 0 (ITI2)  |
| <i>TIMER_SMCFG_TRGS_EL_ITI3</i> | Internal trigger input 0 (ITI3)  |
| <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 3-505. Function timer\_external\_trigger\_as\_external\_clock\_config**

|                      |   |
|----------------------|---|
| <b>Function name</b> | timer_external_trigger_as_external_clock_config |
|----------------------|---|

|  |   |
|--|---|
| <b>Function prototype</b>              | void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t extrigger, uint16_t expolarity, uint32_t extfilter); |
| <b>Function descriptions</b>           | configure TIMER the external trigger as external clock input  |
| <b>Precondition</b>                    | -   |
| <b>The called functions</b>            | timer_input_trigger_source_select   |
| <b>Input parameter{in}</b>             |   |
| <b>timer_periph</b>                    | TIMER peripheral  |
| <i>TIMERx(x=0..4,7,8,11)</i>           | TIMER peripheral selection  |
| <b>Input parameter{in}</b>             |   |
| <b>extrigger</b>                       | external trigger selection  |
| <i>TIMER_SMCFG_TRGS<br/>EL_CIOF_ED</i> | CIO edge flag (CIOF_ED)   |
| <i>TIMER_SMCFG_TRGS<br/>EL_CIOFE0</i>  | channel 0 input Filtered output (CIOFE0)  |
| <i>TIMER_SMCFG_TRGS<br/>EL_C1FE1</i>   | channel 1 input Filtered output (C1FE1)   |
| <b>Input parameter{in}</b>             |   |
| <b>expolarity</b>                      | external trigger polarity   |
| <i>TIMER_IC_POLARITY_<br/>RISING</i>   | active high or rising edge active   |
| <i>TIMER_IC_POLARITY_<br/>FALLING</i>  | active low or falling edge active   |
| <b>Input parameter{in}</b>             |   |
| <b>extfilter</b>                       | external trigger filter control (0~15)  |
| <b>Output parameter{out}</b>           |   |
| -                                      | -   |
| <b>Return value</b>                    |   |
| -                                      | -   |

Example:

```
/* configure TIMER0 the external trigger CIOFE0 as external clock input */
```

```
timer_external_trigger_as_external_clock_config (TIMER0,  
TIMER_SMCFG_TRGSEL_CIOFE0, TIMER_IC_POLARITY_RISING, 0);
```

### timer\_external\_clock\_mode0\_config

The description of timer\_external\_clock\_mode0\_config is shown as below:

**Table 3-506. Function timer\_external\_clock\_mode0\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | timer_external_clock_mode0_config  |
| <b>Function prototype</b>    | void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter); |
| <b>Function descriptions</b> | configure TIMER the external clock mode0   |

|                               |  |
|-------------------------------|--|
| <b>Precondition</b>           | -  |
| <b>The called functions</b>   | timer_external_trigger_config              |
| <b>Input parameter{in}</b>    |  |
| <b>timer_periph</b>           | TIMER peripheral                           |
| <i>TIMERx(x=0..4,7,8,11)</i>  | TIMER peripheral selection                 |
| <b>Input parameter{in}</b>    |  |
| <b>extprescaler</b>           | ETI external trigger prescaler             |
| <i>TIMER_EXT_TRI_PSC_OFF</i>  | no divided                                 |
| <i>TIMER_EXT_TRI_PSC_DIV2</i> | divided by 2                               |
| <i>TIMER_EXT_TRI_PSC_DIV4</i> | divided by 4                               |
| <i>TIMER_EXT_TRI_PSC_DIV8</i> | divided by 8                               |
| <b>Input parameter{in}</b>    |  |
| <b>expolarity</b>             | ETI external trigger polarity              |
| <i>TIMER_ETP_FALLING</i>      | active low or falling edge active          |
| <i>TIMER_ETP_RISING</i>       | active high or rising edge active          |
| <b>Input parameter{in}</b>    |  |
| <b>extfilter</b>              | ETI external trigger filter control (0~15) |
| <b>Output parameter{out}</b>  |  |
| -                             | -  |
| <b>Return value</b>           |  |
| -                             | -  |

Example:

```
/* configure TIMER0 the external clock mode0 */
```

```
timer_external_clock_mode0_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 0);
```

## timer\_external\_clock\_mode1\_config

The description of timer\_external\_clock\_mode1\_config is shown as below:

**Table 3-507. 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..4,7)</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 3-508. Function timer\_external\_clock\_mode1\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_external_clock_mode1_disable                              |
| <b>Function prototype</b>    | void timer_external_clock_mode1_disable(uint32_t timer_periph); |
| <b>Function descriptions</b> | disable TIMER the external clock mode1                          |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>timer_periph</b>          | TIMER peripheral  |
| <i>TIMERx(x=0..4,7)</i>      | TIMER peripheral selection                                      |
| <b>Output parameter{out}</b> |   |
| -                            | -   |



| Return value |   |
|--------------|---|
| -            | - |

Example:

```
/* disable TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_disable (TIMER0);
```

## timer\_write\_chxval\_register\_config

The description of timer\_write\_chxval\_register\_config is shown as below:

**Table 3-509. Function timer\_write\_chxval\_register\_config**

| Function name               | timer_write_chxval_register_config  |
|-----------------------------|---|
| Function prototype          | void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);                             |
| Function descriptions       | configure TIMER write CHxVAL register selection   |
| Precondition                | -   |
| The called functions        | -   |
| Input parameter{in}         |   |
| <b>timer_periph</b>         | TIMER peripheral  |
| <i>TIMERx(x=0..4,7..13)</i> | TIMER peripheral selection  |
| Input parameter{in}         |   |
| <b>ccsel</b>                | write CHxVAL register selection   |
| <i>TIMER_CHVSEL_DISABLE</i> | no effect   |
| <i>TIMER_CHVSEL_ENABLE</i>  | when write the CHxVAL register, if the write value is same as the CHxVAL value, the write access is ignored |
| Output parameter{out}       |   |
| -                           | -   |
| Return value                |   |
| -                           | -   |

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 3-510. Function timer\_output\_value\_selection\_config**

| Function name      | timer_output_value_selection_config   |
|--------------------|---|
| Function prototype | void timer_output_value_selection_config(uint32_t timer_periph, uint16_t outsel); |

|                              |   |
|------------------------------|---|
| <b>Function descriptions</b> | configure TIMER output value selection    |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>timer_periph</b>          | TIMER peripheral                          |
| <i>TIMERx (x=0,7)</i>        | TIMER peripheral selection                |
| <b>Input parameter{in}</b>   |   |
| <b>outsel</b>                | output value selection                    |
| <i>TIMER_OUTSEL_DISABLE</i>  | no effect                                 |
| <i>TIMER_OUTSEL_ENABLE</i>   | if POEN and IOS is 0, the output disabled |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* configure TIMER output value selection */
```

```
timer_output_value_selection_config(TIMER0, TIMER_OUTSEL_ENABLE);
```

### timer\_interrupt\_enable

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

**Table 3-511. Function timer\_interrupt\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_interrupt_enable  |
| <b>Function prototype</b>    | void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt); |
| <b>Function descriptions</b> | enable the TIMER interrupt  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>timer_periph</b>          | TIMER peripheral  |
| <i>TIMERx</i>                | please refer to the following parameters                                |
| <b>Input parameter{in}</b>   |   |
| <b>interrupt</b>             | timer interrupt enable source   |
| <i>TIMER_INT_UP</i>          | update interrupt enable, <i>TIMERx</i> (x=0..13)                        |
| <i>TIMER_INT_CH0</i>         | channel 0 interrupt enable, <i>TIMERx</i> (x=0..4,7..13)                |
| <i>TIMER_INT_CH1</i>         | channel 1 interrupt enable, <i>TIMERx</i> (x=0..4,7,8,11)               |
| <i>TIMER_INT_CH2</i>         | channel 2 interrupt enable, <i>TIMERx</i> (x=0..4,7)                    |
| <i>TIMER_INT_CH3</i>         | channel 3 interrupt enable , <i>TIMERx</i> (x=0..4,7)                   |
| <i>TIMER_INT_CMT</i>         | commutation interrupt enable, <i>TIMERx</i> (x=0,7)                     |
| <i>TIMER_INT_TRG</i>         | trigger interrupt enable, <i>TIMERx</i> (x=0..4,7,8,11)                 |

|                              |  |
|------------------------------|--|
| <i>TIMER_INT_BRK</i>         | break interrupt enable, <i>TIMERx</i> ( <i>x</i> =0,7) |
| <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 3-512. Function `timer_interrupt_disable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | <code>timer_interrupt_disable</code>   |
| <b>Function prototype</b>    | <code>void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt);</code> |
| <b>Function descriptions</b> | disable the TIMER interrupt  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>timer_periph</b>          | TIMER peripheral   |
| <i>TIMERx</i>                | please refer to the following parameters   |
| <b>Input parameter{in}</b>   |  |
| <b>interrupt</b>             | timer interrupt disable source   |
| <i>TIMER_INT_UP</i>          | update interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..13)                             |
| <i>TIMER_INT_CH0</i>         | channel 0 interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4,7..13)                     |
| <i>TIMER_INT_CH1</i>         | channel 1 interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)                    |
| <i>TIMER_INT_CH2</i>         | channel 2 interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4,7)                         |
| <i>TIMER_INT_CH3</i>         | channel 3 interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4,7)                         |
| <i>TIMER_INT_CMT</i>         | commutation interrupt disable, <i>TIMERx</i> ( <i>x</i> =0,7)                          |
| <i>TIMER_INT_TRG</i>         | trigger interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)                      |
| <i>TIMER_INT_BRK</i>         | break interrupt disable, <i>TIMERx</i> ( <i>x</i> =0,7)                                |
| <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 3-513. Function timer\_interrupt\_flag\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_interrupt_flag_get  |
| <b>Function prototype</b>    | FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t interrupt); |
| <b>Function descriptions</b> | get timer interrupt flag  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>timer_periph</b>          | TIMER peripheral  |
| <i>TIMERx</i>                | please refer to the following parameters  |
| <b>Input parameter{in}</b>   |   |
| <b>interrupt</b>             | the timer interrupt bits  |
| <i>TIMER_INT_FLAG_UP</i>     | update interrupt flag, <i>TIMERx</i> (x=0..13)                                  |
| <i>TIMER_INT_FLAG_CH0</i>    | channel 0 interrupt flag, <i>TIMERx</i> (x=0..4,7..13)                          |
| <i>TIMER_INT_FLAG_CH1</i>    | channel 1 interrupt flag, <i>TIMERx</i> (x=0..4,7,8,11)                         |
| <i>TIMER_INT_FLAG_CH2</i>    | channel 2 interrupt flag, <i>TIMERx</i> (x=0..4,7)                              |
| <i>TIMER_INT_FLAG_CH3</i>    | channel 3 interrupt flag, <i>TIMERx</i> (x=0..4,7)                              |
| <i>TIMER_INT_FLAG_CMT</i>    | channel commutation interrupt flag, <i>TIMERx</i> (x=0,7)                       |
| <i>TIMER_INT_FLAG_TRG</i>    | trigger interrupt flag, <i>TIMERx</i> (x=0,7,8,11)                              |
| <i>TIMER_INT_FLAG_BRK</i>    | break interrupt flag, <i>TIMERx</i> (x=0,7)                                     |
| <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 3-514. Function timer\_interrupt\_flag\_clear**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_interrupt_flag_clear  |
| <b>Function prototype</b>    | void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t interrupt); |
| <b>Function descriptions</b> | clear TIMER interrupt flag  |

|                              |  |
|------------------------------|--|
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>timer_periph</b>          | TIMER peripheral   |
| <i>TIMERx</i>                | please refer to the following parameters                           |
| <b>Input parameter{in}</b>   |  |
| <b>interrupt</b>             | the timer interrupt bits   |
| <i>TIMER_INT_FLAG_UP</i>     | update interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..13)            |
| <i>TIMER_INT_FLAG_CH0</i>    | channel 0 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7..13)    |
| <i>TIMER_INT_FLAG_CH1</i>    | channel 1 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)   |
| <i>TIMER_INT_FLAG_CH2</i>    | channel 2 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)        |
| <i>TIMER_INT_FLAG_CH3</i>    | channel 3 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)        |
| <i>TIMER_INT_FLAG_CMT</i>    | channel commutation interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7) |
| <i>TIMER_INT_FLAG_TRG</i>    | trigger interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7,8,11)        |
| <i>TIMER_INT_FLAG_BRK</i>    | break interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7)               |
| <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 3-515. Function timer\_flag\_get**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | timer_flag_get   |
| <b>Function prototype</b>    | FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag); |
| <b>Function descriptions</b> | get TIMER flags  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>timer_periph</b>          | TIMER peripheral   |
| <i>TIMERx</i>                | please refer to the following parameters                         |
| <b>Input parameter{in}</b>   |  |
| <b>flag</b>                  | the timer interrupt flags  |
| <i>TIMER_FLAG_UP</i>         | update flag, <i>TIMERx</i> ( <i>x</i> =0..13)                    |
| <i>TIMER_FLAG_CH0</i>        | channel 0 flag, <i>TIMERx</i> ( <i>x</i> =0..4,7..13)            |
| <i>TIMER_FLAG_CH1</i>        | channel 1 flag, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)           |

|                              |  |
|------------------------------|--|
| <i>TIMER_FLAG_CH2</i>        | channel 2 flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)                  |
| <i>TIMER_FLAG_CH3</i>        | channel 3 flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)                  |
| <i>TIMER_FLAG_CMT</i>        | channel commutation flag, <i>TIMERx</i> ( <i>x</i> =0,7)           |
| <i>TIMER_FLAG_TRG</i>        | trigger flag, <i>TIMERx</i> ( <i>x</i> =0,7,8,11)                  |
| <i>TIMER_FLAG_BRK</i>        | break flag, <i>TIMERx</i> ( <i>x</i> =0,7)                         |
| <i>TIMER_FLAG_CH0O</i>       | channel 0 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..4,7..11)  |
| <i>TIMER_FLAG_CH1O</i>       | channel 1 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11) |
| <i>TIMER_FLAG_CH2O</i>       | channel 2 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)      |
| <i>TIMER_FLAG_CH3O</i>       | channel 3 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)      |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>FlagStatus</b>            | SET or RESET   |

Example:

```
/* get TIMERO update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get (TIMERO, TIMER_FLAG_UP);
```

## timer\_flag\_clear

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

**Table 3-516. Function timer\_flag\_clear**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | timer_flag_clear   |
| <b>Function prototype</b>    | void timer_flag_clear(uint32_t timer_periph, uint32_t flag); |
| <b>Function descriptions</b> | clear TIMER flags  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>timer_periph</b>          | TIMER peripheral   |
| <i>TIMERx</i>                | please refer to the following parameters                     |
| <b>Input parameter{in}</b>   |  |
| <b>flag</b>                  | the timer interrupt flags                                    |
| <i>TIMER_FLAG_UP</i>         | update flag, <i>TIMERx</i> ( <i>x</i> =0..13)                |
| <i>TIMER_FLAG_CH0</i>        | channel 0 flag, <i>TIMERx</i> ( <i>x</i> =0..4,7..13)        |
| <i>TIMER_FLAG_CH1</i>        | channel 1 flag, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)       |
| <i>TIMER_FLAG_CH2</i>        | channel 2 flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)            |
| <i>TIMER_FLAG_CH3</i>        | channel 3 flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)            |
| <i>TIMER_FLAG_CMT</i>        | channel commutation flag, <i>TIMERx</i> ( <i>x</i> =0,7)     |
| <i>TIMER_FLAG_TRG</i>        | trigger flag, <i>TIMERx</i> ( <i>x</i> =0,7,8,11)            |
| <i>TIMER_FLAG_BRK</i>        | break flag, <i>TIMERx</i> ( <i>x</i> =0,7)                   |

|                              |   |
|------------------------------|---|
| <i>TIMER_FLAG_CH0O</i>       | channel 0 overcapture flag, $TIMERx(x=0..4,7..11)$  |
| <i>TIMER_FLAG_CH1O</i>       | channel 1 overcapture flag, $TIMERx(x=0..4,7,8,11)$ |
| <i>TIMER_FLAG_CH2O</i>       | channel 2 overcapture flag, $TIMERx(x=0..4,7)$      |
| <i>TIMER_FLAG_CH3O</i>       | channel 3 overcapture flag, $TIMERx(x=0..4,7)$      |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* clear TIMER0 update flags */
timer_flag_clear (TIMER0, TIMER_FLAG_UP);
```

## 3.21. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter [3.21.1](#), the USART firmware functions are introduced in chapter [3.21.2](#).

### 3.21.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

**Table 3-517. USART Registers**

| Registers   | Descriptions                      |
|-------------|-----------------------------------|
| USART_STAT0 | Status register 0                 |
| USART_DATA  | Data register                     |
| USART_BAUD  | Baud rate register                |
| USART_CTL0  | Control register 0                |
| USART_CTL1  | Control register 1                |
| USART_CTL2  | Control register 2                |
| USART_GP    | Guard time and prescaler register |
| USART_CTL3  | Control register 3                |
| USART_RT    | Receiver timeout register         |
| USART_STAT1 | Status register 1                 |
| USART_CHC   | Coherence control register        |

### 3.21.2. Descriptions of Peripheral functions

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

Table 3-518. USART firmware function

| Function name                           | Function description  |
|---|---|
| usart_deinit                            | reset USART/UART  |
| usart_baudrate_set                      | configure USART baud rate value                                     |
| usart_parity_config                     | configure USART parity  |
| usart_word_length_set                   | configure USART word length   |
| usart_stop_bit_set                      | configure USART stop bit length                                     |
| usart_enable                            | enable USART  |
| usart_disable                           | disable USART   |
| usart_transmit_config                   | configure USART transmitter   |
| usart_receive_config                    | configure USART receiver  |
| usart_data_first_config                 | data is transmitted/received with the LSB/MSB first                 |
| usart_invert_config                     | configure USART inverted  |
| usart_receiver_timeout_enable           | enable receiver timeout   |
| usart_receiver_timeout_disable          | disable receiver timeout  |
| usart_receiver_timeout_threshold_config | configure receiver timeout threshold                                |
| usart_data_transmit                     | USART transmit data function  |
| usart_data_receive                      | USART receive data function   |
| usart_address_config                    | configure the address of the USART in wake up by address match mode |
| usart_mute_mode_enable                  | enable mute mode  |
| usart_mute_mode_disable                 | disable mute mode   |
| usart_mute_mode_wakeup_config           | configure wakeup method in mute mode                                |
| usart_lin_mode_enable                   | enable LIN mode   |
| usart_lin_mode_disable                  | disable LIN mode  |
| usart_lin_break_dection_length_config   | configure LIN break frame length                                    |
| usart_send_break                        | send break frame  |
| usart_halfduplex_enable                 | enable half duplex mode   |
| usart_halfduplex_disable                | disable half duplex mode  |
| usart_synchronous_clock_enable          | enable CK pin in synchronous mode                                   |
| usart_synchronous_clock_disable         | disable CK pin in synchronous mode                                  |
| usart_synchronous_clock_config          | configure USART synchronous mode parameters                         |
| usart_guard_time_config                 | configure guard time value in smartcard mode                        |
| usart_smartcard_mode_enable             | enable smartcard mode   |
| usart_smartcard_mode_disable            | disable smartcard mode  |
| usart_smartcard_mode_nack_enable        | enable NACK in smartcard mode                                       |
| usart_smartcard_mode_nack_disable       | disable NACK in smartcard mode                                      |
| usart_smartcard_autoretry_config        | configure smartcard auto-retry number                               |
| usart_block_length_config               | configure block length  |
| usart_irda_mode_enable                  | enable IrDA mode  |



| Function name                        | Function description  |
|--------------------------------------|---|
| usart_irda_mode_disable              | disable IrDA mode   |
| usart_prescaler_config               | configure the peripheral clock prescaler in USART IrDA low-power mode |
| usart_irda_lowpower_config           | configure IrDA low-power  |
| usart_hardware_flow_rts_config       | configure hardware flow control RTS                                   |
| usart_hardware_flow_cts_config       | configure hardware flow control CTS                                   |
| usart_dma_receive_config             | configure USART DMA reception   |
| usart_dma_transmit_config            | configure USART DMA transmission                                      |
| usart_hardware_flow_coherence_config | configure hardware flow control coherence mode                        |
| usart_flag_get                       | get flag in STAT0/STAT1 register                                      |
| usart_flag_clear                     | clear flag in STAT0/STAT1 register                                    |
| usart_interrupt_enable               | enable USART interrupt  |
| usart_interrupt_disable              | disable USART interrupt   |
| usart_interrupt_flag_get             | get USART interrupt flag status                                       |
| usart_interrupt_flag_clear           | clear USART interrupt flag  |

## Enum usart\_flag\_enum

Table 3-519. usart\_flag\_enum

| Member name      | Function description       |
|------------------|----------------------------|
| USART_FLAG_CTS   | 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 flag         |
| USART_FLAG_NERR  | noise error flag           |
| USART_FLAG_FERR  | frame error flag           |
| USART_FLAG_PERR  | parity error flag          |
| USART_FLAG_BSY   | busy flag                  |
| USART_FLAG_EB    | end of block flag          |
| USART_FLAG_RT    | receiver timeout flag      |
| USART_FLAG_EPERR | early parity error flag    |

## Enum usart\_interrupt\_flag\_enum

Table 3-520. usart\_interrupt\_flag\_enum

| Member name         | Function description                    |
|---------------------|---|
| USART_INT_FLAG_PERR | parity error interrupt flag             |
| USART_INT_FLAG_TBE  | transmitter buffer empty interrupt flag |

| Member name                   | Function description  |
|-------------------------------|---|
| USART_INT_FLAG_TC             | transmission complete interrupt flag                        |
| USART_INT_FLAG_RBNE           | read data buffer not empty interrupt flag                   |
| USART_INT_FLAG_RBNE_ORE<br>RR | read data buffer not empty interrupt and overrun error flag |
| USART_INT_FLAG_IDLE           | IDLE line detected interrupt flag                           |
| USART_INT_FLAG_LBD            | LIN break detected interrupt flag                           |
| USART_INT_FLAG_CTS            | CTS interrupt flag  |
| USART_INT_FLAG_ERR_ORER<br>R  | overrun error interrupt flag                                |
| USART_INT_FLAG_ERR_NERR       | noise error interrupt flag                                  |
| USART_INT_FLAG_ERR_FERR       | frame error interrupt flag                                  |
| USART_INT_FLAG_EB             | end of block interrupt flag                                 |
| USART_INT_FLAG_RT             | receive timeout interrupt flag                              |

## Enum `usart_interrupt_enum`

**Table 3-521. `usart_interrupt_enum`**

| Member name    | Function description   |
|----------------|--|
| 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_CTS  | CTS interrupt  |
| USART_INT_ERR  | error interrupt  |
| USART_INT_EB   | end of block interrupt   |
| USART_INT_RT   | receive timeout interrupt  |

## Enum `usart_invert_enum`

**Table 3-522. `usart_invert_enum`**

| Member name         | Function description         |
|---------------------|------------------------------|
| USART_DINV_ENABLE   | data bit level inversion     |
| USART_DINV_DISABLE  | data bit level not inversion |
| USART_TXPIN_ENABLE  | TX pin level inversion       |
| USART_TXPIN_DISABLE | TX pin level not inversion   |
| USART_RXPIN_ENABLE  | RX pin level inversion       |
| USART_RXPIN_DISABLE | RX pin level not inversion   |

## `usart_deinit`

The description of `usart_deinit` is shown as below:

**Table 3-523. Function `usart_deinit`**

|                                  |  |
|----------------------------------|--|
| <b>Function name</b>             | <code>usart_deinit</code>  |
| <b>Function prototype</b>        | <code>void usart_deinit(uint32_t usart_periph);</code>                       |
| <b>Function descriptions</b>     | reset USART/UART   |
| <b>Precondition</b>              | -  |
| <b>The called functions</b>      | <code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code> |
| <b>Input parameter{in}</b>       |  |
| <b><code>usart_periph</code></b> | USARTx/UARTx peripheral  |
| <i>USARTx</i>                    | x=0,1,2  |
| <i>UARTx</i>                     | x=3,4  |
| <b>Output parameter{out}</b>     |  |
| -                                | -  |
| <b>Return value</b>              |  |
| -                                | -  |

Example:

```
/* reset USART0 */
usart_deinit (USART0);
```

## **`usart_baudrate_set`**

The description of `usart_baudrate_set` is shown as below:

**Table 3-524. Function `usart_baudrate_set`**

|                                  |  |
|----------------------------------|--|
| <b>Function name</b>             | <code>usart_baudrate_set</code>  |
| <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>              | -  |
| <b>The called functions</b>      | <code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code>   |
| <b>Input parameter{in}</b>       |  |
| <b><code>usart_periph</code></b> | USARTx/UARTx peripheral  |
| <i>USARTx</i>                    | x=0,1,2  |
| <i>UARTx</i>                     | x=3,4  |
| <b>Input parameter{in}</b>       |  |
| <b><code>baudval</code></b>      | baud rate value  |
| <b>Output parameter{out}</b>     |  |
| -                                | -  |
| <b>Return value</b>              |  |
| -                                | -  |

Example:

```
/* configure USART0 baud rate value */
usart_baudrate_set(USART0, 115200);
```

## usart\_parity\_config

The description of usart\_parity\_config is shown as below:

**Table 3-525. Function usart\_parity\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | usart_parity_config  |
| <b>Function prototype</b>    | void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg); |
| <b>Function descriptions</b> | configure USART parity   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | USARTx/UARTx peripheral  |
| <i>USARTx</i>                | x=0,1,2  |
| <i>UARTx</i>                 | x=3,4  |
| <b>Input parameter{in}</b>   |  |
| <b>paritycfg</b>             | configure USART parity   |
| <i>USART_PM_NONE</i>         | no parity  |
| <i>USART_PM_ODD</i>          | odd parity   |
| <i>USART_PM_EVEN</i>         | even parity  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure USART parity */
```

```
usart_parity_config(USART0, USART_PM_EVEN);
```

## usart\_word\_length\_set

The description of usart\_word\_length\_set is shown as below:

**Table 3-526. 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>          | USARTx/UARTx peripheral   |
| <i>USARTx</i>                | x=0,1,2   |
| <i>UARTx</i>                 | x=3,4   |
| <b>Input parameter{in}</b>   |   |
| <b>wlen</b>                  | USART word length   |

|                              |        |
|------------------------------|--------|
| <i>USART_WL_8BIT</i>         | 8 bits |
| <i>USART_WL_9BIT</i>         | 9 bits |
| <b>Output parameter{out}</b> |        |
| -                            | -      |
| <b>Return value</b>          |        |
| -                            | -      |

Example:

```
/* configure USART0 word length */
```

```
usart_word_length_set(USART0, USART_WL_9BIT);
```

## usart\_stop\_bit\_set

The description of usart\_stop\_bit\_set is shown as below:

**Table 3-527. 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>          | USARTx/UARTx peripheral  |
| <i>USARTx</i>                | x=0,1,2  |
| <i>UARTx</i>                 | x=3,4  |
| <b>Input parameter{in}</b>   |  |
| <b>stblen</b>                | USART stop bit   |
| <i>USART_STB_1BIT</i>        | 1 bit  |
| <i>USART_STB_0_5BIT</i>      | 0.5 bit, not available for UARTx(x=3,4)                          |
| <i>USART_STB_2BIT</i>        | 2 bits   |
| <i>USART_STB_1_5BIT</i>      | 1.5 bits, not available for UARTx(x=3,4)                         |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure USART0 stop bit length */
```

```
usart_stop_bit_set(USART0, USART_STB_1_5BIT);
```

## usart\_enable

The description of usart\_enable is shown as below:

**Table 3-528. Function `usart_enable`**

|                                  |  |
|----------------------------------|--|
| <b>Function name</b>             | <code>usart_enable</code>                              |
| <b>Function prototype</b>        | <code>void usart_enable(uint32_t usart_periph);</code> |
| <b>Function descriptions</b>     | enable USART   |
| <b>Precondition</b>              | -  |
| <b>The called functions</b>      | -  |
| <b>Input parameter{in}</b>       |  |
| <b><code>usart_periph</code></b> | USARTx/UARTx peripheral                                |
| <i>USARTx</i>                    | x=0,1,2  |
| <i>UARTx</i>                     | x=3,4  |
| <b>Output parameter{out}</b>     |  |
| -                                | -  |
| <b>Return value</b>              |  |
| -                                | -  |

Example:

```
/* enable USART0 */
usart_enable(USART0);
```

## **`usart_disable`**

The description of `usart_disable` is shown as below:

**Table 3-529. Function `usart_disable`**

|                                  |   |
|----------------------------------|---|
| <b>Function name</b>             | <code>usart_disable</code>                              |
| <b>Function prototype</b>        | <code>void usart_disable(uint32_t usart_periph);</code> |
| <b>Function descriptions</b>     | disable USART   |
| <b>Precondition</b>              | -   |
| <b>The called functions</b>      | -   |
| <b>Input parameter{in}</b>       |   |
| <b><code>usart_periph</code></b> | USARTx/UARTx peripheral                                 |
| <i>USARTx</i>                    | x=0,1,2   |
| <i>UARTx</i>                     | x=3,4   |
| <b>Output parameter{out}</b>     |   |
| -                                | -   |
| <b>Return value</b>              |   |
| -                                | -   |

Example:

```
/* disable USART0 */
usart_disable(USART0);
```

## usart\_transmit\_config

The description of usart\_transmit\_config is shown as below:

**Table 3-530. Function usart\_transmit\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | usart_transmit_config   |
| <b>Function prototype</b>    | void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig); |
| <b>Function descriptions</b> | configure USART transmitter   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | USARTx/UARTx peripheral   |
| USARTx                       | x=0,1,2   |
| UARTx                        | x=3,4   |
| <b>Input parameter{in}</b>   |   |
| <b>txconfig</b>              | enable or disable USART transmitter                                   |
| USART_TRANSMIT_ENABLE        | enable USART transmission   |
| USART_TRANSMIT_DISABLE       | enable USART transmission   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* configure USART0 transmitter */
```

```
usart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

## usart\_receive\_config

The description of usart\_receive\_config is shown as below:

**Table 3-531. Function usart\_receive\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | usart_receive_config   |
| <b>Function prototype</b>    | void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig); |
| <b>Function descriptions</b> | configure USART receiver   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | USARTx/UARTx peripheral  |
| USARTx                       | x=0,1,2  |
| UARTx                        | x=3,4  |
| <b>Input parameter{in}</b>   |  |

|                              |                                  |
|------------------------------|----------------------------------|
| <b>rxconfig</b>              | enable or disable USART receiver |
| <i>USART_RECEIVE_ENABLE</i>  | enable USART reception           |
| <i>USART_RECEIVE_DISABLE</i> | 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 3-532. 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>          | USARTx peripheral   |
| <i>USARTx</i>                | x=0,1,2   |
| <b>Input parameter{in}</b>   |   |
| <b>msbf</b>                  | LSB first or MSB first  |
| <i>USART_MSBF_LSB</i>        | LSB first   |
| <i>USART_MSBF_MSB</i>        | MSB first   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* configure LSB of data first */

usart_data_first_config(USART0, USART_MSBF_LSB);
```

## usart\_invert\_config

The description of usart\_invert\_config is shown as below:



**Table 3-533. Function `usart_invert_config`**

|                                       |   |
|---------------------------------------|---|
| <b>Function name</b>                  | <code>usart_invert_config</code>  |
| <b>Function prototype</b>             | <code>void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);</code> |
| <b>Function descriptions</b>          | configure USART inversion   |
| <b>Precondition</b>                   | -   |
| <b>The called functions</b>           | -   |
| <b>Input parameter{in}</b>            |   |
| <b>usart_periph</b>                   | USARTx peripheral   |
| <i>USARTx</i>                         | x=0,1,2   |
| <b>Input parameter{in}</b>            |   |
| <i>invertpara</i>                     | refer to Table 3-522. <code>usart_invert_enum</code>  |
| <i>USART_DINV_ENAB</i><br><i>E</i>    | data bit level inversion  |
| <i>USART_DINV_DISAB</i><br><i>E</i>   | data bit level not inversion  |
| <i>USART_TXPIN_ENAB</i><br><i>LE</i>  | TX pin level inversion  |
| <i>USART_TXPIN_DISAB</i><br><i>LE</i> | TX pin level not inversion  |
| <i>USART_RXPIN_ENAB</i><br><i>LE</i>  | RX pin level inversion  |
| <i>USART_RXPIN_DISAB</i><br><i>LE</i> | RX pin level not inversion  |
| <b>Output parameter{out}</b>          |   |
| -                                     | -   |
| <b>Return value</b>                   |   |
| -                                     | -   |

Example:

```
/* configure USART inversion */
```

```
usart_invert_config(USART0, USART_DINV_ENABLE);
```

## **`usart_receiver_timeout_enable`**

The description of `usart_receiver_timeout_enable` is shown as below:

**Table 3-534. Function `usart_receiver_timeout_enable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | <code>usart_receiver_timeout_enable</code>                              |
| <b>Function prototype</b>    | <code>void usart_receiver_timeout_enable(uint32_t usart_periph);</code> |
| <b>Function descriptions</b> | enable receiver timeout   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |

| Input parameter{in}   |                   |
|-----------------------|-------------------|
| <b>usart_periph</b>   | USARTx peripheral |
| <i>USARTx</i>         | x=0,1,2           |
| Output parameter{out} |                   |
| -                     | -                 |
| Return value          |                   |
| -                     | -                 |

Example:

```
/* enable receiver timeout of USART */
usart_receiver_timeout_enable(USART0);
```

## usart\_receiver\_timeout\_disable

The description of usart\_receiver\_timeout\_disable is shown as below:

**Table 3-535. Function usart\_receiver\_timeout\_disable**

| <b>Function name</b>         | usart_receiver_timeout_disable                              |
|------------------------------|---|
| <b>Function prototype</b>    | void usart_receiver_timeout_disable(uint32_t usart_periph); |
| <b>Function descriptions</b> | disable receiver timeout                                    |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| Input parameter{in}          |   |
| <b>usart_periph</b>          | USARTx peripheral   |
| <i>USARTx</i>                | x=0,1,2   |
| Output parameter{out}        |   |
| -                            | -   |
| Return value                 |   |
| -                            | -   |

Example:

```
/* disable receiver timeout of USART */
usart_receiver_timeout_disable(USART0);
```

## usart\_receiver\_timeout\_threshold\_config

The description of usart\_receiver\_timeout\_threshold\_config is shown as below:

**Table 3-536. Function usart\_receiver\_timeout\_threshold\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | usart_receiver_timeout_threshold_config   |
| <b>Function prototype</b>    | void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout); |
| <b>Function descriptions</b> | configure receiver timeout threshold  |
| <b>Precondition</b>          | -   |

|                              |                   |
|------------------------------|-------------------|
| <b>The called functions</b>  | -                 |
| <b>Input parameter{in}</b>   |                   |
| <b>usart_periph</b>          | USARTx peripheral |
| <i>USARTx</i>                | x=0,1,2           |
| <b>Input parameter{in}</b>   |                   |
| <b>rtimeout</b>              | timeout value     |
| <i>0-0xFFFFF</i>             | timeout value     |
| <b>Output parameter{out}</b> |                   |
| -                            | -                 |
| <b>Return value</b>          |                   |
| -                            | -                 |

Example:

```
/* set the receiver timeout threshold of USART0 */
```

```
usart_receiver_timeout_threshold_config(USART0, 115200*3);
```

## usart\_data\_transmit

The description of usart\_data\_transmit is shown as below:

**Table 3-537. Function usart\_data\_transmit**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | usart_data_transmit   |
| <b>Function prototype</b>    | void usart_data_transmit(uint32_t usart_periph, uint16_t data); |
| <b>Function descriptions</b> | USART transmit data function                                    |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | USARTx/UARTx peripheral   |
| <i>USARTx</i>                | x=0,1,2   |
| <i>UARTx</i>                 | x=3,4   |
| <b>Input parameter{in}</b>   |   |
| <b>data</b>                  | data of transmission  |
| <i>0-0xFF</i>                | data of transmission  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* USART0 transmit data */
```

```
usart_data_transmit(USART0, 0xAA);
```

## usart\_data\_receive

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

**Table 3-538. Function usart\_data\_receive**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | usart_data_receive                                  |
| <b>Function prototype</b>    | uint16_t usart_data_receive(uint32_t usart_periph); |
| <b>Function descriptions</b> | USART receive data function                         |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | USARTx/UARTx peripheral                             |
| <i>USARTx</i>                | x=0,1,2   |
| <i>UARTx</i>                 | x=3,4   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>uint16_t</b>              | data of received(0-0xFF)                            |

Example:

```
/* USART0 receive data */
```

```
uint16_t temp;
```

```
temp = usart_data_receive(USART0);
```

## usart\_address\_config

The description of usart\_address\_config is shown as below:

**Table 3-539. 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 in wake up by address match mode |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | USARTx/UARTx peripheral   |
| <i>USARTx</i>                | x=0,1,2   |
| <i>UARTx</i>                 | x=3,4   |
| <b>Input parameter{in}</b>   |   |
| <b>addr</b>                  | address of USART/UART   |
| <i>0-0xFF</i>                | address of USART/UART   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |

| Return value |   |
|--------------|---|
| -            | - |

Example:

```
/* configure address of the USART0 */
```

```
usart_address_config(USART0, 0x00);
```

## usart\_mute\_mode\_enable

The description of usart\_mute\_mode\_enable is shown as below:

**Table 3-540. Function usart\_mute\_mode\_enable**

| Function name         | usart_mute_mode_enable                              |
|-----------------------|---|
| Function prototype    | void usart_mute_mode_enable(uint32_t usart_periph); |
| Function descriptions | enable mute mode                                    |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |
| usart_periph          | USARTx/UARTx peripheral                             |
| USARTx                | x=0,1,2   |
| UARTx                 | x=3,4   |
| Output parameter{out} |   |
| -                     | -   |
| Return value          |   |
| -                     | -   |

Example:

```
/* enable USART0 receiver in mute mode */
```

```
usart_mute_mode_enable(USART0);
```

## usart\_mute\_mode\_disable

The description of usart\_mute\_mode\_disable is shown as below:

**Table 3-541. Function usart\_mute\_mode\_disable**

| Function name         | usart_mute_mode_disable                              |
|-----------------------|--|
| Function prototype    | void usart_mute_mode_disable(uint32_t usart_periph); |
| Function descriptions | disable mute mode                                    |
| Precondition          | -  |
| The called functions  | -  |
| Input parameter{in}   |  |
| usart_periph          | USARTx/UARTx peripheral                              |
| USARTx                | x=0,1,2  |
| UARTx                 | x=3,4  |

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

Example:

```
/* disable USART0 receiver in mute mode */
```

```
usart_mute_mode_disable(USART0);
```

## usart\_mute\_mode\_wakeup\_config

The description of usart\_mute\_mode\_wakeup\_config is shown as below:

**Table 3-542. Function usart\_mute\_mode\_wakeup\_config**

| Function name         | usart_mute_mode_wakeup_config  |
|-----------------------|--|
| Function prototype    | void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod); |
| Function descriptions | configure wakeup method in mute mode   |
| Precondition          | -  |
| The called functions  | -  |
| Input parameter{in}   |  |
| usart_periph          | USARTx/UARTx peripheral  |
| USARTx                | x=0,1,2  |
| UARTx                 | x=3,4  |
| Input parameter{in}   |  |
| wmethod               | two methods be used to enter or exit the mute mode                           |
| USART_WM_IDLE         | idle line  |
| USART_WM_ADDR         | address mask   |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

Example:

```
/* configure USART0 wakeup method in mute mode */
```

```
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

## usart\_lin\_mode\_enable

The description of usart\_lin\_mode\_enable is shown as below:

**Table 3-543. Function usart\_lin\_mode\_enable**

| Function name      | usart_lin_mode_enable                              |
|--------------------|--|
| Function prototype | 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>          | USARTx/UARTx peripheral |
| <i>USARTx</i>                | x=0,1,2                 |
| <i>UARTx</i>                 | x=3,4                   |
| <b>Output parameter{out}</b> |                         |
| -                            | -                       |
| <b>Return value</b>          |                         |
| -                            | -                       |

Example:

```
/* USART0 LIN mode enable */
```

```
usart_lin_mode_enable(USART0);
```

## usart\_lin\_mode\_disable

The description of usart\_lin\_mode\_disable is shown as below:

**Table 3-544. 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>          | USARTx/UARTx peripheral                             |
| <i>USARTx</i>                | x=0,1,2   |
| <i>UARTx</i>                 | x=3,4   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* USART0 LIN mode disable */
```

```
usart_lin_mode_disable(USART0);
```

## usart\_lin\_break\_dection\_length\_config

The description of usart\_lin\_break\_dection\_length\_config is shown as below:

**Table 3-545. Function `usart_lin_break_dection_length_config`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | <code>usart_lin_break_dection_length_config</code>  |
| <b>Function prototype</b>    | <code>void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lblen);</code> |
| <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>          | USARTx/UARTx peripheral   |
| <i>USARTx</i>                | x=0,1,2   |
| <i>UARTx</i>                 | x=3,4   |
| <b>Input parameter{in}</b>   |   |
| <b>lblen</b>                 | two methods be used to enter or exit the mute mode  |
| <i>USART_LBLEN_10B</i>       | break frame length is 10 bits   |
| <i>USART_LBLEN_11B</i>       | break frame length is 11 bits   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* configure LIN break frame length */
```

```
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

## **usart\_send\_break**

The description of `usart_send_break` is shown as below:

**Table 3-546. Function `usart_send_break`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | <code>usart_send_break</code>                              |
| <b>Function prototype</b>    | <code>void usart_send_break(uint32_t usart_periph);</code> |
| <b>Function descriptions</b> | send break frame   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | USARTx/UARTx peripheral                                    |
| <i>USARTx</i>                | x=0,1,2  |
| <i>UARTx</i>                 | x=3,4  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:



```
/* USART0 send break frame */
```

```
usart_send_break(USART0);
```

## usart\_halfduplex\_enable

The description of usart\_halfduplex\_enable is shown as below:

**Table 3-547. Function usart\_halfduplex\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | usart_halfduplex_enable                              |
| <b>Function prototype</b>    | void usart_halfduplex_enable(uint32_t usart_periph); |
| <b>Function descriptions</b> | enable half duplex mode                              |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | USARTx/UARTx peripheral                              |
| <i>USARTx</i>                | x=0,1,2  |
| <i>UARTx</i>                 | x=3,4  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable USART0 half duplex mode*/
```

```
usart_halfduplex_enable(USART0);
```

## usart\_halfduplex\_disable

The description of usart\_halfduplex\_disable is shown as below:

**Table 3-548. Function usart\_halfduplex\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | usart_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>   |   |
| <b>usart_periph</b>          | USARTx/UARTx peripheral                               |
| <i>USARTx</i>                | x=0,1,2   |
| <i>UARTx</i>                 | x=3,4   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable USART0 half duplex mode*/

usart_halfduplex_disable(USART0);
```

## usart\_synchronous\_clock\_enable

The description of usart\_synchronous\_clock\_enable is shown as below:

**Table 3-549. Function usart\_synchronous\_clock\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | usart_synchronous_clock_enable                              |
| <b>Function prototype</b>    | void usart_synchronous_clock_enable(uint32_t usart_periph); |
| <b>Function descriptions</b> | enable CK pin in synchronous mode                           |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | USARTx peripheral   |
| <i>USARTx</i>                | x=0,1,2   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable USART0 CK pin in synchronous mode */

usart_synchronous_clock_enable(USART0);
```

## usart\_synchronous\_clock\_disable

The description of usart\_synchronous\_clock\_disable is shown as below:

**Table 3-550. Function usart\_synchronous\_clock\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | usart_synchronous_clock_disable                              |
| <b>Function prototype</b>    | void usart_synchronous_clock_disable(uint32_t usart_periph); |
| <b>Function descriptions</b> | disable CK pin in synchronous mode                           |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | USARTx peripheral  |
| <i>USARTx</i>                | x=0,1,2  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable USART0 CK pin in synchronous mode */
```

```
usart_synchronous_clock_disable(USART0);
```

## usart\_synchronous\_clock\_config

The description of usart\_synchronous\_clock\_config is shown as below:

**Table 3-551. Function usart\_synchronous\_clock\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | usart_synchronous_clock_config   |
| <b>Function prototype</b>    | void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl); |
| <b>Function descriptions</b> | configure USART synchronous mode parameters  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | USARTx peripheral  |
| <i>USARTx</i>                | x=0,1,2  |
| <b>Input parameter{in}</b>   |  |
| <b>clen</b>                  | CK length  |
| <i>USART_CLEN_NONE</i>       | there are 7 CK pulses for an 8 bit frame and 8 CK pulses for a 9 bit frame                             |
| <i>USART_CLEN_EN</i>         | there are 8 CK pulses for an 8 bit frame and 9 CK pulses for a 9 bit frame                             |
| <b>Input parameter{in}</b>   |  |
| <b>cph</b>                   | clock phase  |
| <i>USART_CPH_1CK</i>         | first clock transition is the first data capture edge  |
| <i>USART_CPH_2CK</i>         | second clock transition is the first data capture edge   |
| <b>Input parameter{in}</b>   |  |
| <b>cpl</b>                   | clock polarity   |
| <i>USART_CPL_LOW</i>         | steady low value on CK pin   |
| <i>USART_CPL_HIGH</i>        | steady high value on CK pin  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure USART0 synchronous mode parameters */
```

```
usart_synchronous_clock_config(USART0,USART_CLEN_EN,USART_CPH_2CK,  
USART_CPL_HIGH);
```

## usart\_guard\_time\_config

The description of usart\_guard\_time\_config is shown as below:

**Table 3-552. Function usart\_guard\_time\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | usart_guard_time_config  |
| <b>Function prototype</b>    | void usart_guard_time_config(uint32_t usart_periph, uint8_t gaut); |
| <b>Function descriptions</b> | configure guard time value in smartcard mode                       |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | USARTx peripheral  |
| <i>USARTx</i>                | x=0,1,2  |
| <b>Input parameter{in}</b>   |  |
| <b>gaut</b>                  | guard time value   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure USART0 guard time value in smartcard mode */
```

```
usart_guard_time_config(USART0, 0x55);
```

## usart\_smartcard\_mode\_enable

The description of usart\_smartcard\_mode\_enable is shown as below:

**Table 3-553. Function usart\_smartcard\_mode\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | usart_smartcard_mode_enable                              |
| <b>Function prototype</b>    | void usart_smartcard_mode_enable(uint32_t usart_periph); |
| <b>Function descriptions</b> | enable smartcard mode                                    |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | USARTx peripheral  |
| <i>USARTx</i>                | x=0,1,2  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* USART0 smartcard mode enable */
```

```
usart_smartcard_mode_enable(USART0);
```

## usart\_smartcard\_mode\_disable

The description of usart\_smartcard\_mode\_disable is shown as below:

**Table 3-554. Function usart\_smartcard\_mode\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | usart_smartcard_mode_disable                              |
| <b>Function prototype</b>    | void usart_smartcard_mode_disable(uint32_t usart_periph); |
| <b>Function descriptions</b> | disable smartcard mode                                    |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | USARTx peripheral   |
| <i>USARTx</i>                | x=0,1,2   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* USART0 smartcard mode disable */
usart_smartcard_mode_disable(USART0);
```

## usart\_smartcard\_mode\_nack\_enable

The description of usart\_smartcard\_mode\_nack\_enable is shown as below:

**Table 3-555. Function usart\_smartcard\_mode\_nack\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | usart_smartcard_mode_nack_enable                              |
| <b>Function prototype</b>    | void usart_smartcard_mode_nack_enable(uint32_t usart_periph); |
| <b>Function descriptions</b> | enable NACK in smartcard mode                                 |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | USARTx peripheral   |
| <i>USARTx</i>                | x=0,1,2   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_enable(USART0);
```

## usart\_smartcard\_mode\_nack\_disable

The description of usart\_smartcard\_mode\_nack\_disable is shown as below:

**Table 3-556. Function usart\_smartcard\_mode\_nack\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | usart_smartcard_mode_nack_disable                              |
| <b>Function prototype</b>    | void usart_smartcard_mode_nack_disable(uint32_t usart_periph); |
| <b>Function descriptions</b> | disable NACK in smartcard mode                                 |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | USARTx peripheral  |
| <i>USARTx</i>                | x=0,1,2  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_disable(USART0);
```

## usart\_smartcard\_autoretry\_config

The description of usart\_smartcard\_autoretry\_config is shown as below:

**Table 3-557. Function usart\_smartcard\_autoretry\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | usart_smartcard_autoretry_config   |
| <b>Function prototype</b>    | void usart_smartcard_autoretry_config(uint32_t usart_periph, uint8_t scrtnum); |
| <b>Function descriptions</b> | configure smartcard auto-retry number  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | USARTx peripheral  |
| <i>USARTx</i>                | x=0,1,2  |
| <b>Input parameter{in}</b>   |  |
| <b>scrtnum</b>               | smartcard auto-retry number  |
| <i>0-0xFF</i>                | smartcard auto-retry number  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure smartcard auto-retry number */

usart_smartcard_autoretry_config (USART0, 0xFF);
```

## usart\_block\_length\_config

The description of usart\_block\_length\_config is shown as below:

**Table 3-558. Function usart\_block\_length\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | usart_block_length_config  |
| <b>Function prototype</b>    | void usart_block_length_config(uint32_t usart_periph, uint8_t bl); |
| <b>Function descriptions</b> | configure block length   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | USARTx peripheral  |
| <i>USARTx</i>                | x=0,1,2  |
| <b>Input parameter{in}</b>   |  |
| <b>bl</b>                    | block length   |
| <i>0-0xFF</i>                | block length   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure block length in Smartcard T=1 reception */

usart_block_length_config(USART0, 0xFF);
```

## usart\_irda\_mode\_enable

The description of usart\_irda\_mode\_enable is shown as below:

**Table 3-559. Function usart\_irda\_mode\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | usart_irda_mode_enable                              |
| <b>Function prototype</b>    | void usart_irda_mode_enable(uint32_t usart_periph); |
| <b>Function descriptions</b> | enable IrDA mode                                    |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | USARTx/UARTx peripheral                             |
| <i>USARTx</i>                | x=0,1,2   |
| <i>UARTx</i>                 | x=3,4   |

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

Example:

```
/* enable USART0 IrDA mode */
```

```
usart_irda_mode_enable(USART0);
```

## usart\_irda\_mode\_disable

The description of usart\_irda\_mode\_disable is shown as below:

**Table 3-560. Function usart\_irda\_mode\_disable**

| Function name         | usart_irda_mode_disable                              |
|-----------------------|--|
| Function prototype    | void usart_irda_mode_disable(uint32_t usart_periph); |
| Function descriptions | disable IrDA mode                                    |
| Precondition          | -  |
| The called functions  | -  |
| Input parameter{in}   |  |
| usart_periph          | USARTx/UARTx peripheral                              |
| USARTx                | x=0,1,2  |
| UARTx                 | x=3,4  |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

Example:

```
/* disable USART0 IrDA mode */
```

```
usart_irda_mode_disable(USART0);
```

## usart\_prescaler\_config

The description of usart\_prescaler\_config is shown as below:

**Table 3-561. Function usart\_prescaler\_config**

| Function name         | usart_prescaler_config  |
|-----------------------|---|
| Function prototype    | void usart_prescaler_config(uint32_t usart_periph, uint8_t psc);      |
| Function descriptions | configure the peripheral clock prescaler in USART IrDA low-power mode |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |
| usart_periph          | USARTx/UARTx peripheral   |



|                              |           |
|------------------------------|-----------|
| <i>USARTx</i>                | x=0,1,2   |
| <i>UARTx</i>                 | x=3,4     |
| <b>Input parameter{in}</b>   |           |
| <b>psc</b>                   | 0x00-0xFF |
| <b>Output parameter{out}</b> |           |
| -                            | -         |
| <b>Return value</b>          |           |
| -                            | -         |

Example:

```
/* configure the USART0 peripheral clock prescaler */
```

```
usart_prescaler_config(USART0, 0x00);
```

## usart\_irda\_lowpower\_config

The description of usart\_irda\_lowpower\_config is shown as below:

**Table 3-562. Function usart\_irda\_lowpower\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | usart_irda_lowpower_config   |
| <b>Function prototype</b>    | void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp); |
| <b>Function descriptions</b> | configure IrDA low-power   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | USARTx/UARTx peripheral  |
| <i>USARTx</i>                | x=0,1,2  |
| <i>UARTx</i>                 | x=3,4  |
| <b>Input parameter{in}</b>   |  |
| <b>irlp</b>                  | IrDA low-power or normal   |
| <i>USART_IRLP_LOW</i>        | low-power  |
| <i>USART_IRLP_NORMAL</i>     | normal   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure USART0 IrDA low-power */
```

```
usart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

## usart\_hardware\_flow\_rts\_config

The description of usart\_hardware\_flow\_rts\_config is shown as below:

**Table 3-563. Function usart\_hardware\_flow\_rts\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | usart_hardware_flow_rts_config  |
| <b>Function prototype</b>    | void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig); |
| <b>Function descriptions</b> | configure hardware flow control RTS   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | USARTx peripheral   |
| <i>USARTx</i>                | x=0,1,2   |
| <b>Input parameter{in}</b>   |   |
| <b>rtsconfig</b>             | enable or disable RTS   |
| <i>USART_RTS_ENABLE</i>      | enable RTS  |
| <i>USART_RTS_DISABLE</i>     | disable RTS   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* configure USART0 hardware flow control RTS */
```

```
usart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);
```

## usart\_hardware\_flow\_cts\_config

The description of usart\_hardware\_flow\_cts\_config is shown as below:

**Table 3-564. 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 RTS   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | USARTx peripheral   |
| <i>USARTx</i>                | x=0,1,2   |
| <b>Input parameter{in}</b>   |   |
| <b>ctsconfig</b>             | enable or disable CTS   |

|                              |             |
|------------------------------|-------------|
| <i>USART_CTS_ENABLE</i>      | enable CTS  |
| <i>USART_CTS_DISABLE</i>     | disable CTS |
| <b>Output parameter{out}</b> |             |
| -                            | -           |
| <b>Return value</b>          |             |
| -                            | -           |

Example:

```
/* configure USART0 hardware flow control CTS */
```

```
usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

## usart\_dma\_receive\_config

The description of usart\_dma\_receive\_config is shown as below:

**Table 3-565. 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 dmaconfig); |
| <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>              | USARTx/UARTx peripheral   |
| <i>USARTx</i>                    | x=0,1,2   |
| <i>UARTx</i>                     | x=3   |
| <b>Input parameter{in}</b>       |   |
| <b>dmaconfig</b>                 | USART DMA mode  |
| <i>USART_RECEIVE_DMA_ENABLE</i>  | enable USART DMA for reception  |
| <i>USART_RECEIVE_DMA_DISABLE</i> | disable USART DMA for reception   |
| <b>Output parameter{out}</b>     |   |
| -                                | -   |
| <b>Return value</b>              |   |
| -                                | -   |

Example:

```
/* USART0 DMA enable for reception */
```

```
usart_dma_receive_config(USART0, USART_RECEIVE_DMA_ENABLE);
```

## usart\_dma\_transmit\_config

The description of usart\_dma\_transmit\_config is shown as below:

**Table 3-566. 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 dmaconfig); |
| <b>Function descriptions</b> | configure USART DMA transmission   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | USARTx/UARTx peripheral  |
| USARTx                       | x=0,1,2  |
| UARTx                        | x=3  |
| <b>Input parameter{in}</b>   |  |
| <b>dmaconfig</b>             | USART DMA mode   |
| USART_TRANSMIT_DMA_ENABLE    | enable USART DMA for transmission  |
| USART_TRANSMIT_DMA_DISABLE   | disable USART DMA for transmission   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* USART0 DMA enable for transmission */
```

```
usart_dma_transmit_config(USART0, USART_TRANSMIT_DMA_ENABLE);
```

## usart\_hardware\_flow\_coherence\_config

The description of usart\_hardware\_flow\_coherence\_config is shown as below:

**Table 3-567. Function usart\_hardware\_flow\_coherence\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | usart_hardware_flow_coherence_config  |
| <b>Function prototype</b>    | void usart_hardware_flow_coherence_config(uint32_t usart_periph, uint32_t hcm); |
| <b>Function descriptions</b> | configure hardware flow control coherence mode                                  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | USARTx/UARTx peripheral   |
| USARTx                       | x=0,1,2   |
| UARTx                        | x=3   |

| Input parameter{in}             |  |
|---------------------------------|--|
| <b>hcm</b>                      | Hardware flow control coherence mode                       |
| <i>USART_RTS_NONE_COHERENCE</i> | nRTS signal equals to RBNE bit in USART_STAT0 register     |
| <i>USART_RTS_COHERENCE</i>      | nRTS signal is set when the last data bit has been sampled |
| Output parameter{out}           |  |
| -                               | -  |
| Return value                    |  |
| -                               | -  |

Example:

```
/* configure hardware flow control coherence mode */
usart_hardware_flow_coherence_config(USART0, USART_RTS_COHERENCE);
```

## usart\_flag\_get

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

**Table 3-568. 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 STAT0/STAT1 register  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| Input parameter{in}          |   |
| <b>usart_periph</b>          | USARTx/UARTx peripheral   |
| <i>USARTx</i>                | x=0,1,2   |
| <i>UARTx</i>                 | x=3,4   |
| Input parameter{in}          |   |
| <b>flag</b>                  | USART flags, refer to <a href="#">Table 3-519. usart_flag_enum</a>      |
| Output parameter{out}        |   |
| -                            | -   |
| Return value                 |   |
| <b>FlagStatus</b>            | SET or RESET  |

Example:

```
/* get flag USART0 state */
FlagStatus status;
status = usart_flag_get(USART0, USART_FLAG_TBE);
```

## usart\_flag\_clear

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

**Table 3-569. 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 STAT0/STAT1 register                                  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | USARTx/UARTx peripheral   |
| USARTx                       | x=0,1,2   |
| UARTx                        | x=3,4   |
| <b>Input parameter{in}</b>   |   |
| <b>flag</b>                  | USART flags, refer to <a href="#">Table 3-519. usart_flag_enum</a>  |
| USART_FLAG_CTSF              | CTS change flag   |
| USART_FLAG_LBDF              | LIN break detected flag   |
| USART_FLAG_TC                | transmission complete   |
| USART_FLAG_RBNE              | read data buffer not empty  |
| USART_FLAG_EB                | end of block flag   |
| USART_FLAG_RT                | receiver timeout flag   |
| USART_FLAG_EPERR             | early parity error flag   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* clear USART0 flag */
usart_flag_clear(USART0,USART_FLAG_TC);
```

## usart\_interrupt\_enable

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

**Table 3-570. Function usart\_interrupt\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | usart_interrupt_enable  |
| <b>Function prototype</b>    | void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt); |
| <b>Function descriptions</b> | enable USART interrupt  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |

|                              |   |
|------------------------------|---|
| <b>usart_periph</b>          | USARTx/UARTx peripheral   |
| <i>USARTx</i>                | x=0,1,2   |
| <i>UARTx</i>                 | x=3,4   |
| <b>Input parameter{in}</b>   |   |
| <b>interrupt</b>             | USART interrupt, refer to <a href="#">Table 3-521. usart_interrupt_enum</a> |
| <i>USART_INT_PERR</i>        | parity error interrupt  |
| <i>USART_INT_TBE</i>         | transmitter buffer empty interrupt  |
| <i>USART_INT_TC</i>          | transmission complete interrupt   |
| <i>USART_INT_RBNE</i>        | read data buffer not empty interrupt and overrun error interrupt            |
| <i>USART_INT_IDLE</i>        | IDLE line detected interrupt  |
| <i>USART_INT_LBD</i>         | LIN break detected interrupt  |
| <i>USART_INT_ERR</i>         | error interrupt   |
| <i>USART_INT_CTS</i>         | CTS interrupt   |
| <i>USART_INT_RT</i>          | receive timeout event interrupt   |
| <i>USART_INT_EB</i>          | end of block event interrupt  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable USART0 TBE interrupt */
```

```
usart_interrupt_enable(USART0, USART_INT_TBE);
```

### usart\_interrupt\_disable

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

**Table 3-571. Function usart\_interrupt\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | usart_interrupt_disable  |
| <b>Function prototype</b>    | void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt); |
| <b>Function descriptions</b> | disable USART interrupt  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | USARTx/UARTx peripheral  |
| <i>USARTx</i>                | x=0,1,2  |
| <i>UARTx</i>                 | x=3,4  |
| <b>Input parameter{in}</b>   |  |
| <b>int_flag</b>              | USART interrupt flag, refer to <a href="#">Table 3-521. usart_interrupt_enum</a>     |
| <i>USART_INT_PERR</i>        | parity error interrupt   |
| <i>USART_INT_TBE</i>         | transmitter buffer empty interrupt   |

|                              |  |
|------------------------------|--|
| <i>USART_INT_TC</i>          | transmission complete interrupt                                  |
| <i>USART_INT_RBNE</i>        | read data buffer not empty interrupt and overrun error interrupt |
| <i>USART_INT_IDLE</i>        | IDLE line detected interrupt                                     |
| <i>USART_INT_LBD</i>         | LIN break detected interrupt                                     |
| <i>USART_INT_ERR</i>         | error interrupt  |
| <i>USART_INT_CTS</i>         | CTS interrupt  |
| <i>USART_INT_RT</i>          | receive timeout event interrupt                                  |
| <i>USART_INT_EB</i>          | end of block event interrupt                                     |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable USART0 TBE interrupt */
```

```
usart_interrupt_disable(USART0, USART_INT_TBE);
```

## usart\_interrupt\_flag\_get

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

**Table 3-572. Function usart\_interrupt\_flag\_get**

|                                  |  |
|----------------------------------|--|
| <b>Function name</b>             | usart_interrupt_flag_get   |
| <b>Function prototype</b>        | FlagStatus usart_interrupt_flag_get(uint32_t usart_periph,<br>usart_interrupt_flag_enum int_flag); |
| <b>Function descriptions</b>     | get USART interrupt flag status  |
| <b>Precondition</b>              | -  |
| <b>The called functions</b>      | -  |
| <b>Input parameter{in}</b>       |  |
| <b>usart_periph</b>              | USARTx/UARTx peripheral  |
| <i>USARTx</i>                    | x=0,1,2  |
| <i>UARTx</i>                     | x=3,4  |
| <b>Input parameter{in}</b>       |  |
| <b>int_flag</b>                  | USART interrupt flag, refer to <a href="#">Table 3-520. usart_interrupt_flag_enum</a>              |
| <i>USART_INT_FLAG_PERRR</i>      | parity error interrupt and flag  |
| <i>USART_INT_FLAG_TBNE</i>       | transmitter buffer empty interrupt and flag  |
| <i>USART_INT_FLAG_TC</i>         | transmission complete interrupt and flag   |
| <i>USART_INT_FLAG_RBNE</i>       | read data buffer not empty interrupt and flag  |
| <i>USART_INT_FLAG_RBNE_ORERR</i> | read data buffer not empty interrupt and overrun error flag  |



|                                       |                                       |
|---------------------------------------|---------------------------------------|
| <code>USART_INT_FLAG_IDLE</code>      | IDLE line detected interrupt and flag |
| <code>USART_INT_FLAG_LBD</code>       | LIN break detected interrupt and flag |
| <code>USART_INT_FLAG_CTS</code>       | CTS interrupt and flag                |
| <code>USART_INT_FLAG_ERR_ORERR</code> | error interrupt and overrun error     |
| <code>USART_INT_FLAG_ERR_NERR</code>  | error interrupt and noise error flag  |
| <code>USART_INT_FLAG_ERR_FERR</code>  | error interrupt and frame error flag  |
| <code>USART_INT_FLAG_EB</code>        | end of block event interrupt flag     |
| <code>USART_INT_FLAG_RT</code>        | receive timeout event interrupt 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 3-573. 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   |
| <b>Precondition</b>             | -  |
| <b>The called functions</b>     | -  |
| <b>Input parameter{in}</b>      |  |
| <b>usart_periph</b>             | USARTx/UARTx peripheral  |
| <code>USARTx</code>             | x=0,1,2  |
| <code>UARTx</code>              | x=3,4  |
| <b>Input parameter{in}</b>      |  |
| <b>int_flag</b>                 | USART interrupt flag, refer to <a href="#">Table 3-520. usart_interrupt_flag_enum</a>                    |
| <code>USART_INT_FLAG_CTS</code> | CTS change flag  |

|   |                                      |
|---|--------------------------------------|
| <code>USART_INT_FLAG_LB</code><br><code>D</code>  | LIN break detected flag              |
| <code>USART_INT_FLAG_TC</code>                    | transmission complete                |
| <code>USART_INT_FLAG_RB</code><br><code>NE</code> | read data buffer not empty           |
| <code>USART_INT_FLAG_EB</code>                    | end of block event interrupt flag    |
| <code>USART_INT_FLAG_RT</code>                    | receive timeout event interrupt flag |
| <b>Output parameter{out}</b>                      |                                      |
| -   | -                                    |
| <b>Return value</b>                               |                                      |
| -   | -                                    |

Example:

```
/* clear the USART0 interrupt enable bit status */
```

```
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

## 3.22. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.22.1](#), the FWDGT firmware functions are introduced in chapter [3.22.2](#).

### 3.22.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

**Table 3-574. WWDGT Registers**

| Registers  | Descriptions           |
|------------|------------------------|
| WWDGT_CTL  | Control register       |
| WWDGT_CFG  | Configuration register |
| WWDGT_STAT | Status register        |

### 3.22.2. Descriptions of Peripheral functions

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

**Table 3-575. WWDGT firmware function**

| Function name                     | Function description   |
|-----------------------------------|--|
| <code>wwdgt_deinit</code>         | reset the window watchdog timer configuration                      |
| <code>wwdgt_enable</code>         | start the window watchdog timer counter                            |
| <code>wwdgt_counter_update</code> | configure the window watchdog timer counter value                  |
| <code>wwdgt_config</code>         | configure counter value, window value, and prescaler divider value |

| Function name          | Function description                        |
|------------------------|---|
| 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 3-576. Function wwdgt\_deinit**

|                       |   |
|-----------------------|---|
| Function name         | wwdgt_deinit                                  |
| Function prototype    | void wwdgt_deinit(void);                      |
| Function descriptions | reset the window watchdog timer configuration |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |
| -                     | -   |
| Output parameter{out} |   |
| -                     | -   |
| Return value          |   |
| -                     | -   |

Example:

```
/* reset the window watchdog timer configuration */
wwdgt_deinit ( );
```

## wwdgt\_enable

The description of wwdgt\_enable is shown as below:

**Table 3-577. Function wwdgt\_enable**

|                       |   |
|-----------------------|---|
| Function name         | wwdgt_enable                            |
| Function prototype    | void wwdgt_enable (void);               |
| Function descriptions | start the window watchdog timer counter |
| Precondition          | -                                       |
| The called functions  | -                                       |
| Input parameter{in}   |   |
| -                     | -                                       |
| Output parameter{out} |   |
| -                     | -                                       |
| Return value          |   |
| -                     | -                                       |

Example:

```
/* start the window watchdog timer counter */
```

```
wwdgt_enable ( );
```

## wwdgt\_counter\_update

The description of wwdgt\_counter\_update is shown as below:

**Table 3-578. Function wwdgt\_counter\_update**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | wwdgt_counter_update                               |
| <b>Function prototype</b>    | void wwdgt_counter_update(uint16_t counter_value); |
| <b>Function descriptions</b> | configure the window watchdog timer counter value  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>counter_value</b>         | 0x00 - 0x7F  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* update WWDGT counter to 0x7F */
```

```
wwdgt_counter_update(127);
```

## wwdgt\_config

The description of wwdgt\_config is shown as below:

**Table 3-579. Function wwdgt\_config**

|                                |   |
|--------------------------------|---|
| <b>Function name</b>           | wwdgt_config  |
| <b>Function prototype</b>      | void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler); |
| <b>Function descriptions</b>   | configure counter value, window value, and prescaler divider value        |
| <b>Precondition</b>            | -   |
| <b>The called functions</b>    | -   |
| <b>Input parameter{in}</b>     |   |
| <b>counter</b>                 | 0x00 - 0x7F   |
| <b>Input parameter{in}</b>     |   |
| <b>window</b>                  | 0x00 - 0x7F   |
| <b>Input parameter{in}</b>     |   |
| <b>prescaler</b>               | wwdgt prescaler value   |
| <i>WWDGT_CFG_PSC_D<br/>IV1</i> | the time base of window watchdog counter = (PCLK1/4096)/1                 |
| <i>WWDGT_CFG_PSC_D<br/>IV2</i> | the time base of window watchdog counter = (PCLK1/4096)/2                 |

|                                      |   |
|--------------------------------------|---|
| <i>WWDGT_CFG_PSC_D</i><br><i>IV4</i> | the time base of window watchdog counter = (PCLK1/4096)/4 |
| <i>WWDGT_CFG_PSC_D</i><br><i>IV8</i> | the time base of window watchdog counter = (PCLK1/4096)/8 |
| <b>Output parameter{out}</b>         |   |
| -                                    | -   |
| <b>Return value</b>                  |   |
| -                                    | -   |

Example:

```
/* configure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
```

```
wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

## wwdgt\_interrupt\_enable

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

**Table 3-580. Function wwdgt\_interrupt\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | wwdgt_interrupt_enable                 |
| <b>Function prototype</b>    | void wwdgt_interrupt_enable(void);     |
| <b>Function descriptions</b> | enable early wakeup interrupt of WWDGT |
| <b>Precondition</b>          | -                                      |
| <b>The called functions</b>  | -                                      |
| <b>Input parameter{in}</b>   |  |
| -                            | -                                      |
| <b>Output parameter{out}</b> |  |
| -                            | -                                      |
| <b>Return value</b>          |  |
| -                            | -                                      |

Example:

```
/* enable early wakeup interrupt of WWDGT */
```

```
wwdgt_interrupt_enable ( );
```

## wwdgt\_flag\_get

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

**Table 3-581. Function wwdgt\_flag\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | wwdgt_flag_get                              |
| <b>Function prototype</b>    | FlagStatus wwdgt_flag_get(void);            |
| <b>Function descriptions</b> | check early wakeup interrupt state of WWDGT |
| <b>Precondition</b>          | -   |

|                              |              |
|------------------------------|--------------|
| <b>The called functions</b>  | -            |
| <b>Input parameter{in}</b>   |              |
| -                            | -            |
| <b>Output parameter{out}</b> |              |
| -                            | -            |
| <b>Return value</b>          |              |
| <b>FlagStatus</b>            | SET or RESET |

Example:

```
/* test if the counter value update has reached the 0x40 */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get ( );
```

```
if(status == RESET)
```

```
{
```

```
...
```

```
}else
```

```
{
```

```
...
```

```
}
```

## wwdgt\_flag\_clear

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

**Table 3-582. Function wwdgt\_flag\_clear**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | wwdgt_flag_clear                            |
| <b>Function prototype</b>    | void wwdgt_flag_clear(void);                |
| <b>Function descriptions</b> | clear early wakeup interrupt state of WWDGT |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* clear early wakeup interrupt state of WWDGT */
```

`wwdgt_flag_clear( );`

## 4. Revision history

Table 4-1. Revision history

| Revision No. | Description   | Date         |
|--------------|---|--------------|
| 1.0          | Initial Release   | Dec.26, 2017 |
| 1.1          | Revision contents: the folder structure diagram in section 2.1 is replaced; the folder description in section 2.1.2 is modified; section 2.1.3 is deleted; 2-3 drawing in section 2.1.4 is replaced; the file description in section 2.2 is modified; the header format of some drawings is modified  | Sep.30.2020  |
| 1.2          | Revision contents: CAN peripheral is deleted  | Dec.31.2020  |
| 1.3          | <ol style="list-style-type: none"> <li>1. Change function<br/>void pmu_to_standbymode(uint8_t standbymodecmd)</li> <li>2. Change all function qspi_xxx() to spi_quad_xxx()</li> <li>3. Add function<br/>fwdgt_prescaler_value_config()<br/>fwdgt_reload_value_config()</li> <li>4. Delete USBFS chapter</li> <li>5. Delete function<br/>dbg_trace_pin_mode_set(), cannot support ETM</li> </ol> | Jul.31.2022  |
| 1.4          | <ol style="list-style-type: none"> <li>1. Change Usart functions:<br/>usart_data_transmit<br/>usart_guard_time_config<br/>usart_smartcard_autoretry_config<br/>usart_block_length_config<br/>usart_dma_receive_config<br/>usart_dma_transmit_config</li> <li>2. Change Timer decoder name</li> </ol>  | Dec.31.2022  |
| 1.5          | The entire DAC chapter has been modified  | Dec.31,2023  |



## 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.