

**GigaDevice Semiconductor Inc.**

**GD32F20x  
ARM® Cortex®-M3 32-bit MCU**

**Firmware Library  
User Guide**

Revision 1.3

(Jul. 2022)

# 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>27</b>  |
| <b>1.1. Rules of User Manual and Firmware Library .....</b>           | <b>27</b>  |
| <b>1.1.1. Peripherals.....</b>  | <b>27</b>  |
| <b>1.1.2. Naming rules.....</b>                                       | <b>28</b>  |
| <b>2. Firmware Library Overview.....</b>                              | <b>30</b>  |
| <b>2.1. File Structure of Firmware Library .....</b>                  | <b>30</b>  |
| <b>2.1.1. Examples Folder .....</b>                                   | <b>31</b>  |
| <b>2.1.2. Firmware Folder.....</b>                                    | <b>31</b>  |
| <b>2.1.3. Template Folder .....</b>                                   | <b>31</b>  |
| <b>2.1.4. Utilities Folder .....</b>                                  | <b>35</b>  |
| <b>2.2. File descriptions of Firmware Library .....</b>               | <b>35</b>  |
| <b>3. Firmware Library of Standard Peripherals .....</b>              | <b>37</b>  |
| <b>3.1. Overview of Firmware Library of Standard Peripherals.....</b> | <b>37</b>  |
| <b>3.2. ADC .....</b>   | <b>37</b>  |
| <b>3.2.1. Descriptions of Peripheral registers.....</b>               | <b>37</b>  |
| <b>3.2.2. Descriptions of Peripheral functions .....</b>              | <b>38</b>  |
| <b>3.3. BKP.....</b>  | <b>65</b>  |
| <b>3.3.1. Descriptions of Peripheral registers.....</b>               | <b>65</b>  |
| <b>3.3.2. Descriptions of Peripheral functions .....</b>              | <b>65</b>  |
| <b>3.4. CAN .....</b>   | <b>79</b>  |
| <b>3.4.1. Descriptions of Peripheral registers.....</b>               | <b>79</b>  |
| <b>3.4.2. Descriptions of Peripheral functions .....</b>              | <b>80</b>  |
| <b>3.5. CAU .....</b>   | <b>103</b> |
| <b>3.5.1. Descriptions of Peripheral registers.....</b>               | <b>103</b> |
| <b>3.5.2. Descriptions of Peripheral functions .....</b>              | <b>103</b> |
| <b>3.6. CRC .....</b>   | <b>125</b> |
| <b>3.6.1. Descriptions of Peripheral registers.....</b>               | <b>125</b> |
| <b>3.6.2. Descriptions of Peripheral functions .....</b>              | <b>126</b> |
| <b>3.7. DAC .....</b>   | <b>130</b> |
| <b>3.7.1. Descriptions of Peripheral registers.....</b>               | <b>130</b> |

---

|  |            |
|--|------------|
| 3.7.2. Descriptions of Peripheral functions .....  | 130        |
| <b>3.8. DBG .....</b>                              | <b>145</b> |
| 3.8.1. Descriptions of Peripheral registers .....  | 145        |
| 3.8.2. Descriptions of Peripheral functions .....  | 145        |
| <b>3.9. DCI .....</b>                              | <b>151</b> |
| 3.9.1. Descriptions of Peripheral registers .....  | 151        |
| 3.9.2. Descriptions of Peripheral functions .....  | 151        |
| <b>3.10. DMA .....</b>                             | <b>164</b> |
| 3.10.1. Descriptions of Peripheral registers ..... | 164        |
| 3.10.2. Descriptions of Peripheral functions ..... | 165        |
| <b>3.11. ENET .....</b>                            | <b>186</b> |
| 3.11.1. Descriptions of Peripheral registers ..... | 186        |
| 3.11.2. Descriptions of Peripheral functions ..... | 188        |
| <b>3.12. EXMC .....</b>                            | <b>267</b> |
| 3.12.1. Descriptions of Peripheral registers ..... | 268        |
| 3.12.2. Descriptions of Peripheral functions ..... | 268        |
| <b>3.13. EXTI .....</b>                            | <b>303</b> |
| 3.13.1. Descriptions of Peripheral registers ..... | 303        |
| 3.13.2. Descriptions of Peripheral functions ..... | 304        |
| <b>3.14. FMC .....</b>                             | <b>311</b> |
| 3.14.1. Descriptions of Peripheral registers ..... | 312        |
| 3.14.2. Descriptions of Peripheral functions ..... | 312        |
| <b>3.15. FWDGT .....</b>                           | <b>331</b> |
| 3.15.1. Descriptions of Peripheral registers ..... | 331        |
| 3.15.2. Descriptions of Peripheral functions ..... | 331        |
| <b>3.16. GPIO .....</b>                            | <b>336</b> |
| 3.16.1. Descriptions of Peripheral registers ..... | 336        |
| 3.16.2. Descriptions of Peripheral functions ..... | 336        |
| <b>3.17. HAU .....</b>                             | <b>356</b> |
| 3.17.1. Descriptions of Peripheral registers ..... | 356        |
| 3.17.2. Descriptions of Peripheral functions ..... | 356        |
| <b>3.18. I2C .....</b>                             | <b>373</b> |
| 3.18.1. Descriptions of Peripheral registers ..... | 373        |
| 3.18.2. Descriptions of Peripheral functions ..... | 374        |
| <b>3.19. MISC .....</b>                            | <b>395</b> |
| 3.19.1. Descriptions of Peripheral registers ..... | 395        |
| 3.19.2. Descriptions of Peripheral functions ..... | 396        |
| <b>3.20. PMU .....</b>                             | <b>403</b> |
| 3.20.1. Descriptions of Peripheral registers ..... | 403        |

---

|  |            |
|--|------------|
| 3.20.2. Descriptions of Peripheral functions ..... | 403        |
| <b>3.21. RCU .....</b>                             | <b>410</b> |
| 3.21.1. Descriptions of Peripheral registers ..... | 410        |
| 3.21.2. Descriptions of Peripheral functions ..... | 411        |
| <b>3.22. RTC .....</b>                             | <b>447</b> |
| 3.22.1. Descriptions of Peripheral registers ..... | 447        |
| 3.22.2. Descriptions of Peripheral functions ..... | 448        |
| <b>3.23. SDIO .....</b>                            | <b>456</b> |
| 3.23.1. Descriptions of Peripheral registers ..... | 456        |
| 3.23.2. Descriptions of Peripheral functions ..... | 456        |
| <b>3.24. SPI .....</b>                             | <b>487</b> |
| 3.24.1. Descriptions of Peripheral registers ..... | 487        |
| 3.24.2. Descriptions of Peripheral functions ..... | 487        |
| <b>3.25. TIMER.....</b>                            | <b>511</b> |
| 3.25.1. Descriptions of Peripheral registers ..... | 511        |
| 3.25.2. Descriptions of Peripheral functions ..... | 511        |
| <b>3.26. TLI .....</b>                             | <b>565</b> |
| 3.26.1. Descriptions of Peripheral registers ..... | 565        |
| 3.26.2. Descriptions of Peripheral functions ..... | 566        |
| <b>3.27. TRNG.....</b>                             | <b>582</b> |
| 3.27.1. Descriptions of Peripheral registers ..... | 583        |
| 3.27.2. Descriptions of Peripheral functions ..... | 583        |
| <b>3.28. USART.....</b>                            | <b>588</b> |
| 3.28.1. Descriptions of Peripheral registers ..... | 588        |
| 3.28.2. Descriptions of Peripheral functions ..... | 589        |
| <b>3.29. WWDGT.....</b>                            | <b>621</b> |
| 3.29.1. Descriptions of Peripheral registers ..... | 621        |
| 3.29.2. Descriptions of Peripheral functions ..... | 621        |
| <b>4. Revision history .....</b>                   | <b>627</b> |

## List of Figures

|   |           |
|---|-----------|
| <b>Figure 2-1. File structure of firmware library of GD32F20x .....</b> | <b>30</b> |
| <b>Figure 2-2. Select peripheral example files .....</b>                | <b>32</b> |
| <b>Figure 2-3. Copy the peripheral example files .....</b>              | <b>33</b> |
| <b>Figure 2-4. Open the project file .....</b>                          | <b>34</b> |
| <b>Figure 2-5. Configure project files .....</b>                        | <b>34</b> |
| <b>Figure 2-6. Compile-debug-download .....</b>                         | <b>35</b> |

## List of Tables

|   |           |
|---|-----------|
| <b>Table 1-1. Peripherals .....</b>                                   | <b>27</b> |
| <b>Table 2-1. Function descriptions of Firmware Library.....</b>      | <b>35</b> |
| <b>Table 3-1. Peripheral function format of Firmware Library.....</b> | <b>37</b> |
| <b>Table 3-2. ADC Registers .....</b>                                 | <b>37</b> |
| <b>Table 3-3. ADC firmware function.....</b>                          | <b>38</b> |
| <b>Table 3-4. Function adc_deinit.....</b>                            | <b>39</b> |
| <b>Table 3-5. Function adc_mode_config .....</b>                      | <b>40</b> |
| <b>Table 3-6. Function adc_special_function_config.....</b>           | <b>41</b> |
| <b>Table 3-7. Function adc_data_alignment_config.....</b>             | <b>41</b> |
| <b>Table 3-8. Function adc_enable .....</b>                           | <b>42</b> |
| <b>Table 3-9. Function adc_disable .....</b>                          | <b>43</b> |
| <b>Table 3-10. Function adc_calibration_enable.....</b>               | <b>43</b> |
| <b>Table 3-11. Function adc_tempsensor_vrefint_enable .....</b>       | <b>44</b> |
| <b>Table 3-12. Function adc_tempsensor_vrefint_disable.....</b>       | <b>44</b> |
| <b>Table 3-13. Function adc_dma_mode_enable.....</b>                  | <b>45</b> |
| <b>Table 3-14. Function adc_dma_mode_disable.....</b>                 | <b>45</b> |
| <b>Table 3-15. Function adc_discontinuous_mode_config .....</b>       | <b>46</b> |
| <b>Table 3-16. Function adc_channel_length_config.....</b>            | <b>46</b> |
| <b>Table 3-17. Function adc_regular_channel_config .....</b>          | <b>47</b> |
| <b>Table 3-18. Function adc_inserted_channel_config .....</b>         | <b>48</b> |
| <b>Table 3-19. Function adc_inserted_channel_offset_config .....</b>  | <b>49</b> |
| <b>Table 3-20. Function adc_external_trigger_source_config .....</b>  | <b>50</b> |
| <b>Table 3-21. Function adc_external_trigger_config.....</b>          | <b>52</b> |
| <b>Table 3-22. Function adc_software_trigger_enable .....</b>         | <b>53</b> |
| <b>Table 3-23. Function adc_regular_data_read .....</b>               | <b>54</b> |
| <b>Table 3-24. Function adc_inserted_data_read .....</b>              | <b>54</b> |
| <b>Table 3-25. Function adc_sync_mode_convert_value_read .....</b>    | <b>55</b> |
| <b>Table 3-26. Function adc_watchdog_single_channel_enable.....</b>   | <b>55</b> |
| <b>Table 3-27. Function adc_watchdog_group_channel_enable .....</b>   | <b>56</b> |
| <b>Table 3-28. Function adc_watchdog_disable .....</b>                | <b>57</b> |
| <b>Table 3-29. Function adc_watchdog_threshold_config .....</b>       | <b>57</b> |
| <b>Table 3-30. Function adc_resolution_config .....</b>               | <b>58</b> |
| <b>Table 3-31. Function adc_oversample_mode_config .....</b>          | <b>58</b> |
| <b>Table 3-32. Function adc_oversample_mode_enable .....</b>          | <b>60</b> |
| <b>Table 3-33. Function adc_oversample_mode_disable .....</b>         | <b>61</b> |
| <b>Table 3-34. Function adc_flag_get .....</b>                        | <b>61</b> |
| <b>Table 3-35. Function adc_flag_clear .....</b>                      | <b>62</b> |
| <b>Table 3-36. Function adc_interrupt_enable .....</b>                | <b>62</b> |
| <b>Table 3-37. Function adc_interrupt_disable .....</b>               | <b>63</b> |
| <b>Table 3-38. Function adc_interrupt_flag_get.....</b>               | <b>64</b> |

|   |    |
|---|----|
| Table 3-39. Function adc_interrupt_flag_clear .....           | 64 |
| Table 3-40. BKP Registers .....                               | 65 |
| Table 3-41. BKP firmware function.....                        | 65 |
| Table 3-42. Enum bkp_data_register_enum .....                 | 66 |
| Table 3-43. Enum bkp_tamper_enum .....                        | 67 |
| Table 3-44. Function bkp_deinit.....                          | 67 |
| Table 3-45. Function bkp_data_write .....                     | 68 |
| Table 3-46. Function bkp_data_read.....                       | 68 |
| Table 3-47. Function bkp_rtc_calibration_output_enable.....   | 69 |
| Table 3-48. Function bkp_rtc_calibration_output_disable.....  | 70 |
| Table 3-49. Function bkp_rtc_signal_output_enable.....        | 70 |
| Table 3-50. Function bkp_rtc_signal_output_disable.....       | 70 |
| Table 3-51. Function bkp_rtc_output_select .....              | 71 |
| Table 3-52. Function bkp_rtc_clock_output_select .....        | 72 |
| Table 3-53. Function bkp_rtc_clock_calibration_direction..... | 72 |
| Table 3-54. Function bkp_rtc_calibration_value_set .....      | 73 |
| Table 3-55. Function bkp_tamper_detection_enable .....        | 73 |
| Table 3-56. Function bkp_tamper_detection_disable .....       | 74 |
| Table 3-57. Function bkp_tamper_active_level_set .....        | 74 |
| Table 3-58. Function bkp_waveform_detect_config .....         | 75 |
| Table 3-59. Function bkp_flag_get .....                       | 76 |
| Table 3-60. Function bkp_flag_clear .....                     | 76 |
| Table 3-61. Function bkp_tamper_interrupt_enable .....        | 77 |
| Table 3-62. Function bkp_tamper_interrupt_disable .....       | 77 |
| Table 3-63. Function bkp_interrupt_flag_get .....             | 78 |
| Table 3-64. Function bkp_interrupt_flag_clear .....           | 78 |
| Table 3-65. CAN Registers .....                               | 79 |
| Table 3-66. CAN firmware function .....                       | 80 |
| Table 3-67. can_parameter_struct .....                        | 81 |
| Table 3-68. can_trasnmit_message_struct .....                 | 81 |
| Table 3-69. can_receive_message_struct .....                  | 81 |
| Table 3-70. can_filter_parameter_struct.....                  | 82 |
| Table 3-71. can_flag_enum.....                                | 82 |
| Table 3-72. can_interrupt_flag_enum .....                     | 83 |
| Table 3-73. can_error_enum.....                               | 83 |
| Table 3-74. can_transmit_state_enum .....                     | 84 |
| Table 3-75. can_struct_type_enum .....                        | 84 |
| Table 3-76. Function can_deinit.....                          | 84 |
| Table 3-77. Function can_struct_para_init .....               | 85 |
| Table 3-78. Function can_init .....                           | 85 |
| Table 3-79. Function can_filter_init.....                     | 86 |
| Table 3-80. Function can1_filter_start_bank .....             | 87 |
| Table 3-81. Function can_debug_freeze_enable .....            | 88 |
| Table 3-82. Function can_debug_freeze_disable .....           | 88 |

---

|   |     |
|---|-----|
| Table 3-83. Function can_time_trigger_mode_enable.....    | 89  |
| Table 3-84. Function can_time_trigger_mode_disable.....   | 89  |
| Table 3-85. Function can_message_transmit.....            | 90  |
| Table 3-86. Function can_transmit_states .....            | 91  |
| Table 3-87. Function can_transmission_stop .....          | 91  |
| Table 3-88. Function can_message_receive .....            | 92  |
| Table 3-89. Function can_fifo_release .....               | 93  |
| Table 3-90. Function can_receive_message_length_get ..... | 93  |
| Table 3-91. Function can_working_mode_set.....            | 94  |
| Table 3-92. Function can_wakeup .....                     | 94  |
| Table 3-93. Function can_error_get .....                  | 95  |
| Table 3-94. Function can_receive_error_number_get .....   | 95  |
| Table 3-95. Function can_transmit_error_number_get .....  | 96  |
| Table 3-96. Function can_flag_get .....                   | 97  |
| Table 3-97. Function can_flag_clear .....                 | 98  |
| Table 3-98. Function can_interrupt_enable .....           | 99  |
| Table 3-99. Function can_interrupt_disable .....          | 100 |
| Table 3-100. Function can_interrupt_flag_get.....         | 101 |
| Table 3-101. Function can_interrupt_flag_clear .....      | 102 |
| Table 3-102. CAU Registers .....                          | 103 |
| Table 3-103. CAU firmware function .....                  | 104 |
| Table 3-104. Structure cau_key_parameter_struct.....      | 104 |
| Table 3-105. Structure cau_iv_parameter_struct.....       | 105 |
| Table 3-106. Enumeration cau_text_struct .....            | 105 |
| Table 3-107. Function cau_deinit.....                     | 105 |
| Table 3-108. Function cau_enable.....                     | 106 |
| Table 3-109. Function cau_disable.....                    | 106 |
| Table 3-110. Function cau_dma_enable .....                | 106 |
| Table 3-111. Function cau_dma_disable .....               | 107 |
| Table 3-112. Function cau_init .....                      | 108 |
| Table 3-113. Function cau_aes_keysizes_config.....        | 109 |
| Table 3-114. Function cau_key_init .....                  | 109 |
| Table 3-115. Function cau_key_parameter_init.....         | 110 |
| Table 3-116. Function cau_iv_init .....                   | 110 |
| Table 3-117. Function cau_iv_parameter_init.....          | 111 |
| Table 3-118. Function cau_fifo_flush .....                | 112 |
| Table 3-119. Function cau_enable_state_get .....          | 112 |
| Table 3-120. Function cau_data_write .....                | 113 |
| Table 3-121. Function cau_data_read.....                  | 113 |
| Table 3-122. Function cau_aes_ecb .....                   | 114 |
| Table 3-123. Function cau_aes_cbc .....                   | 115 |
| Table 3-124. Function cau_aes_ctr .....                   | 116 |
| Table 3-125. Function cau_tdes_ecb .....                  | 118 |
| Table 3-126. Function cau_tdes_cbc .....                  | 119 |

---

|   |     |
|---|-----|
| Table 3-127. Function cau_des_ecb.....                              | 120 |
| Table 3-128. Function cau_des_cbc.....                              | 121 |
| Table 3-129. Function cau_flag_get .....                            | 123 |
| Table 3-130. Function cau_interrupt_enable .....                    | 124 |
| Table 3-131. Function cau_interrupt_disable .....                   | 124 |
| Table 3-132. Function cau_interrupt_flag_get.....                   | 125 |
| Table 3-133. CRC Registers .....                                    | 125 |
| Table 3-134. CRC firmware function .....                            | 126 |
| Table 3-135. Function crc_deinit .....                              | 126 |
| Table 3-136. Function crc_data_register_reset.....                  | 126 |
| Table 3-137. Function crc_data_register_read.....                   | 127 |
| Table 3-138. Function crc_free_data_register_read.....              | 127 |
| Table 3-139. Function crc_free_data_register_write.....             | 128 |
| Table 3-140. Function crc_single_data_calculate .....               | 128 |
| Table 3-141. Function crc_block_data_calculate .....                | 129 |
| Table 3-142. DAC Registers .....                                    | 130 |
| Table 3-143. DAC firmware function .....                            | 130 |
| Table 3-144. Function dac_deinit.....                               | 131 |
| Table 3-145. Function dac_enable.....                               | 132 |
| Table 3-146. Function dac_disable.....                              | 132 |
| Table 3-147. Function dac_dma_enable.....                           | 133 |
| Table 3-148. Function dac_dma_disable .....                         | 133 |
| Table 3-149. Function dac_output_buffer_enable .....                | 134 |
| Table 3-150. Function dac_output_buffer_disable .....               | 134 |
| Table 3-151. Function dac_output_value_get .....                    | 135 |
| Table 3-152. Function dac_data_set .....                            | 135 |
| Table 3-153. Function dac_trigger_enable .....                      | 136 |
| Table 3-154. Function dac_trigger_disable .....                     | 136 |
| Table 3-155. Function dac_trigger_source_config .....               | 137 |
| Table 3-156. Function dac_software_trigger_enable .....             | 138 |
| Table 3-157. Function dac_software_trigger_disable .....            | 138 |
| Table 3-158. Function dac_wave_mode_config .....                    | 139 |
| Table 3-159. Function dac_wave_bit_width_config .....               | 139 |
| Table 3-160. Function dac_lfsr_noise_config .....                   | 140 |
| Table 3-161. Function dac_triangle_noise_config .....               | 141 |
| Table 3-162. Function dac_concurrent_enable .....                   | 141 |
| Table 3-163. Function dac_concurrent_disable .....                  | 142 |
| Table 3-164. Function dac_concurrent_software_trigger_enable .....  | 142 |
| Table 3-165. Function dac_concurrent_software_trigger_disable ..... | 143 |
| Table 3-166. Function dac_concurrent_output_buffer_enable .....     | 143 |
| Table 3-167. Function dac_concurrent_output_buffer_disable .....    | 144 |
| Table 3-168. Function dac_concurrent_data_set .....                 | 144 |
| Table 3-169. DBG Registers.....                                     | 145 |
| Table 3-170. DBG firmware function .....                            | 145 |

---

|  |     |
|--|-----|
| Table 3-171. Enum dbg_periph_enum .....                  | 145 |
| Table 3-172. Function dbg_id_get .....                   | 146 |
| Table 3-173. Function dbg_low_power_enable.....          | 147 |
| Table 3-174. Function dbg_low_power_disable.....         | 147 |
| Table 3-175. Function dbg_periph_enable.....             | 148 |
| Table 3-176. Function dbg_periph_disable.....            | 148 |
| Table 3-177. Function dbg_trace_pin_enable .....         | 149 |
| Table 3-178. Function dbg_trace_pin_disable .....        | 150 |
| Table 3-179. Function dbg_trace_pin_mode_set.....        | 150 |
| Table 3-180. DCI Registers.....                          | 151 |
| Table 3-181. DCI firmware function .....                 | 151 |
| Table 3-182. Structure dci_parameter_struct.....         | 152 |
| Table 3-183. Function dci_deinit.....                    | 152 |
| Table 3-184. Function dci_init .....                     | 153 |
| Table 3-185. Function dci_enable .....                   | 154 |
| Table 3-186. Function dci_disable .....                  | 154 |
| Table 3-187. Function dci_capture_enable .....           | 155 |
| Table 3-188. Function dci_capture_disable .....          | 155 |
| Table 3-189. Function dci_jpeg_enable .....              | 156 |
| Table 3-190. Function dci_jpeg_disable .....             | 156 |
| Table 3-191. Function dci_crop_window_enable.....        | 157 |
| Table 3-192. Function dci_crop_window_disable.....       | 157 |
| Table 3-193. Function dci_crop_window_config .....       | 158 |
| Table 3-194. Function dci_embedded_sync_enable .....     | 158 |
| Table 3-195. Function dci_embedded_sync_disable .....    | 159 |
| Table 3-196. Function dci_sync_codes_config .....        | 159 |
| Table 3-197. Function dci_sync_codes_unmask_config ..... | 160 |
| Table 3-198. Function dci_data_read .....                | 160 |
| Table 3-199. Function dci_flag_get .....                 | 161 |
| Table 3-200. Function dci_interrupt_enable .....         | 162 |
| Table 3-201. Function dci_interrupt_disable .....        | 162 |
| Table 3-202. Function dci_interrupt_flag_get.....        | 163 |
| Table 3-203. Function dci_interrupt_flag_clear .....     | 164 |
| Table 3-204. DMA Registers.....                          | 164 |
| Table 3-205. DMA firmware function .....                 | 165 |
| Table 3-206. Structure dma_parameter_struct.....         | 166 |
| Table 3-207. Function dma_deinit .....                   | 166 |
| Table 3-208. Function dma_struct_para_init .....         | 167 |
| Table 3-209. Function dma_init .....                     | 167 |
| Table 3-210. Function dma_circulation_enable .....       | 168 |
| Table 3-211. Function dma_circulation_disable .....      | 169 |
| Table 3-212. Function dma_memory_to_memory_enable .....  | 169 |
| Table 3-213. Function dma_memory_to_memory_disable ..... | 170 |
| Table 3-214. Function dma_channel_enable .....           | 171 |

---

|  |     |
|--|-----|
| Table 3-215. Function <code>dma_channel_disable</code> .....                       | 171 |
| Table 3-216. Function <code>dma_periph_address_config</code> .....                 | 172 |
| Table 3-217. Function <code>dma_memory_address_config</code> .....                 | 173 |
| Table 3-218. Function <code>dma_transfer_number_config</code> .....                | 173 |
| Table 3-219. Function <code>dma_transfer_number_get</code> .....                   | 174 |
| Table 3-220. Function <code>dma_priority_config</code> .....                       | 175 |
| Table 3-221. Function <code>dma_memory_width_config</code> .....                   | 175 |
| Table 3-222. Function <code>dma_periph_width_config</code> .....                   | 176 |
| Table 3-223. Function <code>dma_memory_increase_enable</code> .....                | 177 |
| Table 3-224. Function <code>dma_memory_increase_disable</code> .....               | 178 |
| Table 3-225. Function <code>dma_periph_increase_enable</code> .....                | 178 |
| Table 3-226. Function <code>dma_periph_increase_disable</code> .....               | 179 |
| Table 3-227. Function <code>dma_transfer_direction_config</code> .....             | 180 |
| Table 3-228. Function <code>dma_1_channel_5_fulldata_transfer_enable</code> .....  | 180 |
| Table 3-229. Function <code>dma_1_channel_5_fulldata_transfer_disable</code> ..... | 181 |
| Table 3-230. Function <code>dma_flag_get</code> .....                              | 181 |
| Table 3-231. Function <code>dma_flag_clear</code> .....                            | 182 |
| Table 3-232. Function <code>dma_interrupt_enable</code> .....                      | 183 |
| Table 3-233. Function <code>dma_interrupt_disable</code> .....                     | 184 |
| Table 3-234. Function <code>dma_interrupt_flag_get</code> .....                    | 184 |
| Table 3-235. Function <code>dma_interrupt_flag_clear</code> .....                  | 185 |
| Table 3-236. ENET Registers .....  | 186 |
| Table 3-237. ENET firmware function .....  | 188 |
| Table 3-238. Structure <code>enet_descriptors_struct</code> .....                  | 191 |
| Table 3-239. Structure <code>enet_ptp_systime_struct</code> .....                  | 191 |
| Table 3-240. Enum <code>enet_flag_enum</code> .....                                | 192 |
| Table 3-241. Enum <code>enet_flag_clear_enum</code> .....                          | 194 |
| Table 3-242. Enum <code>enet_int_enum</code> .....                                 | 194 |
| Table 3-243. Enum <code>enet_int_flag_enum</code> .....                            | 196 |
| Table 3-244. Enum <code>enet_int_flag_clear_enum</code> .....                      | 197 |
| Table 3-245. Enum <code>enet_desc_reg_enum</code> .....                            | 198 |
| Table 3-246. Enum <code>enet_msc_counter_enum</code> .....                         | 198 |
| Table 3-247. Enum <code>enet_option_enum</code> .....                              | 199 |
| Table 3-248. Enum <code>enet_mediamode_enum</code> .....                           | 199 |
| Table 3-249. Enum <code>enet_chksumconf_enum</code> .....                          | 200 |
| Table 3-250. Enum <code>enet_frmrecept_enum</code> .....                           | 200 |
| Table 3-251. Enum <code>enet_registers_type_enum</code> .....                      | 200 |
| Table 3-252. Enum <code>enet_dmadirection_enum</code> .....                        | 201 |
| Table 3-253. Enum <code>enet_phydirection_enum</code> .....                        | 201 |
| Table 3-254. Enum <code>enet_regdirection_enum</code> .....                        | 201 |
| Table 3-255. Enum <code>enet_macaddress_enum</code> .....                          | 201 |
| Table 3-256. Enum <code>enet_descstate_enum</code> .....                           | 201 |
| Table 3-257. Function <code>enet_deinit</code> .....                               | 202 |
| Table 3-258. Function <code>enet_initpara_config</code> .....                      | 202 |

---

|   |     |
|---|-----|
| Table 3-259. Function enet_init.....                          | 206 |
| Table 3-260. Function enet_software_reset.....                | 207 |
| Table 3-261. Function enet_rxframe_size_get.....              | 207 |
| Table 3-262. Function enet_descriptors_chain_init .....       | 208 |
| Table 3-263. Function enet_descriptors_ring_init.....         | 209 |
| Table 3-264. Function enet_frame_receive .....                | 209 |
| Table 3-265. Function enet_frame_transmit .....               | 210 |
| Table 3-266. Function enet_transmit_checksum_config .....     | 210 |
| Table 3-267. Function enet_enable .....                       | 211 |
| Table 3-268. Function enet_disable .....                      | 212 |
| Table 3-269. Function enet_mac_address_set.....               | 212 |
| Table 3-270. Function enet_mac_address_get .....              | 213 |
| Table 3-271. Function enet_flag_get.....                      | 214 |
| Table 3-272. Function enet_flag_clear.....                    | 216 |
| Table 3-273. Function enet_interrupt_enable .....             | 217 |
| Table 3-274. Function enet_interrupt_disable.....             | 219 |
| Table 3-275. Function enet_interrupt_flag_get .....           | 220 |
| Table 3-276. Function enet_interrupt_flag_clear .....         | 222 |
| Table 3-277. Function enet_tx_enable .....                    | 223 |
| Table 3-278. Function enet_tx_disable .....                   | 224 |
| Table 3-279. Function enet_rx_enable .....                    | 224 |
| Table 3-280. Function enet_rx_disable .....                   | 225 |
| Table 3-281. Function enet_registers_get.....                 | 225 |
| Table 3-282. Function enet_address_filter_enable.....         | 226 |
| Table 3-283. Function enet_address_filter_disable .....       | 227 |
| Table 3-284. Function enet_address_filter_config .....        | 227 |
| Table 3-285. Function enet_phy_config .....                   | 229 |
| Table 3-286. Function enet_phy_write_read.....                | 229 |
| Table 3-287. Function enet_phyloopback_enable .....           | 230 |
| Table 3-288. Function enet_phyloopback_disable .....          | 230 |
| Table 3-289. Function enet_forward_feature_enable .....       | 231 |
| Table 3-290. Function enet_forward_feature_disable .....      | 231 |
| Table 3-291. Function enet_fliter_feature_enable .....        | 232 |
| Table 3-292. Function enet_fliter_feature_disable .....       | 233 |
| Table 3-293. Function enet_pauseframe_generate .....          | 234 |
| Table 3-294. Function enet_pauseframe_detect_config .....     | 234 |
| Table 3-295. Function enet_pauseframe_config.....             | 235 |
| Table 3-296. Function enet_flowcontrol_threshold_config ..... | 236 |
| Table 3-297. Function enet_flowcontrol_feature_enable .....   | 237 |
| Table 3-298. Function enet_flowcontrol_feature_disable .....  | 238 |
| Table 3-299. Function enet_dmaprocess_state_get .....         | 238 |
| Table 3-300. Function enet_dmaprocess_resume.....             | 239 |
| Table 3-301. Function enet_rxprocess_check_recovery.....      | 240 |
| Table 3-302. Function enet_txfifo_flush .....                 | 240 |

---

|   |     |
|---|-----|
| Table 3-303. Function enet_current_desc_address_get .....             | 241 |
| Table 3-304. Function enet_desc_information_get .....                 | 242 |
| Table 3-305. Function enet_missed_frame_counter_get .....             | 243 |
| Table 3-306. Function enet_desc_flag_get .....                        | 243 |
| Table 3-307. Function enet_desc_flag_set .....                        | 245 |
| Table 3-308. Function enet_desc_flag_clear .....                      | 246 |
| Table 3-309. Function enet_desc_receive_complete_bit_enable .....     | 247 |
| Table 3-310. Function enet_desc_receive_complete_bit_disable .....    | 248 |
| Table 3-311. Function enet_rxframe_drop .....                         | 248 |
| Table 3-312. Function enet_dma_feature_enable .....                   | 249 |
| Table 3-313. Function enet_dma_feature_disable .....                  | 249 |
| Table 3-314. Function enet_ptp_normal_descriptors_chain_init .....    | 250 |
| Table 3-315. Function enet_ptp_normal_descriptors_ring_init .....     | 251 |
| Table 3-316. Function enet_ptpframe_receive_normal_mode .....         | 251 |
| Table 3-317. Function enet_ptpframe_transmit_normal_mode .....        | 252 |
| Table 3-318. Function enet_wum_filter_register_pointer_reset .....    | 253 |
| Table 3-319. Function enet_wum_filter_config .....                    | 253 |
| Table 3-320. Function enet_wum_feature_enable .....                   | 254 |
| Table 3-321. Function enet_wum_feature_disable .....                  | 254 |
| Table 3-322. Function enet_msc_counters_reset .....                   | 255 |
| Table 3-323. Function enet_msc_feature_enable .....                   | 255 |
| Table 3-324. Function enet_msc_feature_disable .....                  | 256 |
| Table 3-325. Function enet_msc_counters_get .....                     | 257 |
| Table 3-326. Function enet_ptp_subsecond_2_nanosecond .....           | 258 |
| Table 3-327. Function enet_ptp_nanosecond_2_subsecond .....           | 258 |
| Table 3-328. Function enet_ptp_feature_enable .....                   | 259 |
| Table 3-329. Function enet_ptp_feature_disable .....                  | 259 |
| Table 3-330. Function enet_ptp_timestamp_function_config .....        | 260 |
| Table 3-331. Function enet_ptp_subsecond_increment_config .....       | 261 |
| Table 3-332. Function enet_ptp_timestamp_addend_config .....          | 261 |
| Table 3-333. Function enet_ptp_timestamp_update_config .....          | 262 |
| Table 3-334. Function enet_ptp_expected_time_config .....             | 262 |
| Table 3-335. Function enet_ptp_system_time_get .....                  | 263 |
| Table 3-336. Function enet_ptp_start .....                            | 263 |
| Table 3-337. Function enet_ptp_finecorrection_adjfreq .....           | 264 |
| Table 3-338. Function enet_ptp_coarsecorrection_systime_update .....  | 265 |
| Table 3-339. Function enet_ptp_finecorrection_settime .....           | 266 |
| Table 3-340. Function enet_ptp_flag_get .....                         | 266 |
| Table 3-341. Function enet_initpara_reset .....                       | 267 |
| Table 3-342. EXMC Registers .....                                     | 268 |
| Table 3-343. EXMC firmware function .....                             | 268 |
| Table 3-344. Structure exmc_norsram_timing_parameter_struct .....     | 270 |
| Table 3-345. Structure exmc_norsram_parameter_struct .....            | 270 |
| Table 3-346. Structure exmc_nand_pccard_timing_parameter_struct ..... | 271 |

---

|  |     |
|--|-----|
| Table 3-347. Structure exmc_nand_parameter_struct.....           | 271 |
| Table 3-348. Structure exmc_pccard_parameter_struct .....        | 271 |
| Table 3-349. Structure exmc_sdram_timing_parameter_struct.....   | 272 |
| Table 3-350. Structure exmc_sdram_parameter_struct .....         | 272 |
| Table 3-351. Structure exmc_sdram_command_parameter_struct ..... | 273 |
| Table 3-352. Structure exmc_sqipssram_parameter_struct.....      | 273 |
| Table 3-353. Function exmc_norsram_deinit .....                  | 273 |
| Table 3-354. Function exmc_norsram_struct_para_init.....         | 274 |
| Table 3-355. Function exmc_norsram_init.....                     | 274 |
| Table 3-356. Function exmc_norsram_enable .....                  | 276 |
| Table 3-357. Function exmc_norsram_disable .....                 | 276 |
| Table 3-358. Function exmc_nand_deinit .....                     | 277 |
| Table 3-359. Function exmc_nand_struct_para_init.....            | 277 |
| Table 3-360. Function exmc_nand_init.....                        | 278 |
| Table 3-361. Function exmc_nand_enable .....                     | 279 |
| Table 3-362. Function exmc_nand_disable .....                    | 279 |
| Table 3-363. Function exmc_nand_ecc_config.....                  | 280 |
| Table 3-364. Function exmc_ecc_get .....                         | 281 |
| Table 3-365. Function exmc_pccard_deinit.....                    | 281 |
| Table 3-366. Function exmc_pccard_struct_para_init.....          | 282 |
| Table 3-367. Function exmc_pccard_init .....                     | 282 |
| Table 3-368. Function exmc_pccard_enable .....                   | 283 |
| Table 3-369. Function exmc_pccard_disable .....                  | 284 |
| Table 3-370. Function exmc_sdram_deinit .....                    | 284 |
| Table 3-371. Function exmc_sdram_struct_para_init.....           | 285 |
| Table 3-372. Function exmc_sdram_init .....                      | 285 |
| Table 3-373. Function exmc_sdram_struct_command_para_init.....   | 286 |
| Table 3-374. Function exmc_sdram_command_config.....             | 287 |
| Table 3-375. Function exmc_sdram_refresh_count_set .....         | 287 |
| Table 3-376. Function exmc_sdram_autorefresh_number_set.....     | 288 |
| Table 3-377. Function exmc_sdram_write_protection_config .....   | 288 |
| Table 3-378. Function exmc_sdram_bankstatus_get .....            | 289 |
| Table 3-379. Function exmc_sdram_readsampel_config.....          | 290 |
| Table 3-380. Function exmc_sdram_readsampel_enable .....         | 291 |
| Table 3-381. Function exmc_sdram_readsampel_disable .....        | 291 |
| Table 3-382. Function exmc_sqipssram_deinit .....                | 291 |
| Table 3-383. Function exmc_sqipssram_struct_para_init.....       | 292 |
| Table 3-384. Function exmc_sqipssram_init.....                   | 293 |
| Table 3-385. Function exmc_sqipssram_read_command_set.....       | 293 |
| Table 3-386. Function exmc_sqipssram_write_command_set.....      | 294 |
| Table 3-387. Function exmc_sqipssram_read_id_command_send .....  | 295 |
| Table 3-388. Function exmc_sqipssram_write_cmd_send.....         | 295 |
| Table 3-389. Function exmc_sqipssram_low_id_get.....             | 296 |
| Table 3-390. Function exmc_sqipssram_low_id_get.....             | 296 |

|  |     |
|--|-----|
| Table 3-391. Function exmc_sqipssram_send_command_state_get..... | 297 |
| Table 3-392. Function exmc_flag_get .....                        | 298 |
| Table 3-393. Function exmc_flag_clear.....                       | 299 |
| Table 3-394. Function exmc_interrupt_enable .....                | 300 |
| Table 3-395. Function exmc_interrupt_disable .....               | 301 |
| Table 3-396. Function exmc_interrupt_flag_get .....              | 301 |
| Table 3-397. Function exmc_interrupt_flag_clear .....            | 302 |
| Table 3-398. EXTI Registers.....                                 | 304 |
| Table 3-399. EXTI firmware function .....                        | 304 |
| Table 3-400. Enum exti_line_enum .....                           | 304 |
| Table 3-401. Enum exti_mode_enum .....                           | 305 |
| Table 3-402. Enum exti_trig_type_enum .....                      | 305 |
| Table 3-403. Function exti_deinit.....                           | 305 |
| Table 3-404. Function exti_init .....                            | 306 |
| Table 3-405. Function exti_flag_get .....                        | 306 |
| Table 3-406. Function exti_flag_clear .....                      | 307 |
| Table 3-407. Function exti_interrupt_enable .....                | 307 |
| Table 3-408. Function exti_interrupt_disable .....               | 308 |
| Table 3-409. Function exti_event_enable.....                     | 308 |
| Table 3-410. Function exti_event_disable.....                    | 309 |
| Table 3-411. Function exti_software_interrupt_enable.....        | 309 |
| Table 3-412. Function exti_software_interrupt_disable .....      | 310 |
| Table 3-413. Function exti_interrupt_flag_get.....               | 310 |
| Table 3-414. Function exti_interrupt_flag_clear .....            | 311 |
| Table 3-415. FMC Registers .....                                 | 312 |
| Table 3-416. FMC firmware function .....                         | 312 |
| Table 3-417. fmc_state_enum.....                                 | 313 |
| Table 3-418. fmc_interrupt_enum .....                            | 313 |
| Table 3-419. fmc_flag_enum.....                                  | 314 |
| Table 3-420. fmc_interrupt_flag_enum .....                       | 314 |
| Table 3-421. Function fmc_wscnt_set .....                        | 315 |
| Table 3-422. Function fmc_unlock.....                            | 315 |
| Table 3-423. Function fmc_bank0_unlock .....                     | 316 |
| Table 3-424. Function fmc_bank1_unlock .....                     | 316 |
| Table 3-425. Function fmc_lock .....                             | 317 |
| Table 3-426. Function fmc_bank0_lock .....                       | 317 |
| Table 3-427. Function fmc_bank1_lock .....                       | 318 |
| Table 3-428. Function fmc_page_erase .....                       | 318 |
| Table 3-429. Function fmc_mass_erase .....                       | 319 |
| Table 3-430. Function fmc_bank0_erase .....                      | 319 |
| Table 3-431. Function fmc_bank1_erase .....                      | 320 |
| Table 3-432. Function fmc_word_program .....                     | 320 |
| Table 3-433. Function fmc_halfword_program .....                 | 321 |
| Table 3-434. Function ob_unlock.....                             | 321 |

---

|  |     |
|--|-----|
| Table 3-435. Function ob_lock .....                            | 322 |
| Table 3-436. Function ob_erase .....                           | 322 |
| Table 3-437. Function ob_write_protection_enable .....         | 323 |
| Table 3-438. Function ob_security_protection_config .....      | 323 |
| Table 3-439. Function ob_user_write .....                      | 324 |
| Table 3-440. Function ob_data_program .....                    | 325 |
| Table 3-441. Function ob_user_get .....                        | 325 |
| Table 3-442. Function ob_data_get .....                        | 326 |
| Table 3-443. Function ob_write_protection_get .....            | 326 |
| Table 3-444. Function ob_spc_get .....                         | 327 |
| Table 3-445. Function fmc_flag_get .....                       | 327 |
| Table 3-446. Function fmc_flag_clear .....                     | 328 |
| Table 3-447. Function fmc_interrupt_enable .....               | 329 |
| Table 3-448. Function fmc_interrupt_disable .....              | 329 |
| Table 3-449. Function fmc_interrupt_flag_get .....             | 330 |
| Table 3-450. Function fmc_interrupt_flag_clear .....           | 330 |
| Table 3-451. FWDGT Registers .....                             | 331 |
| Table 3-452. FWDGT firmware function .....                     | 331 |
| Table 3-453. Function fwdgt_write_enable .....                 | 331 |
| Table 3-454. Function fwdgt_write_disable .....                | 332 |
| Table 3-455. Function fwdgt_enable .....                       | 332 |
| Table 3-456. Function fwdgt_prescaler_value_config .....       | 333 |
| Table 3-457. Function fwdgt_reload_value_config .....          | 333 |
| Table 3-458. Function fwdgt_counter_reload .....               | 334 |
| Table 3-459. Function fwdgt_config .....                       | 334 |
| Table 3-460. Function fwdgt_flag_get fwdgt_write_disable ..... | 335 |
| Table 3-461. GPIO Registers .....                              | 336 |
| Table 3-462. GPIO firmware function .....                      | 336 |
| Table 3-463. Function gpio_deinit .....                        | 337 |
| Table 3-464. Function gpio_afio_deinit .....                   | 338 |
| Table 3-465. Function gpio_init .....                          | 338 |
| Table 3-466. Function gpio_bit_set .....                       | 339 |
| Table 3-467. Function gpio_bit_reset .....                     | 340 |
| Table 3-468. Function gpio_bit_write .....                     | 340 |
| Table 3-469. Function gpio_port_write .....                    | 341 |
| Table 3-470. Function gpio_input_bit_get .....                 | 342 |
| Table 3-471. Function gpio_input_port_get .....                | 342 |
| Table 3-472. Function gpio_output_bit_get .....                | 343 |
| Table 3-473. Function gpio_output_port_get .....               | 343 |
| Table 3-474. Function gpio_pin_remap_config .....              | 344 |
| Table 3-475. Function gpio_pin_remap1_config .....             | 346 |
| Table 3-476. Function gpio_ethernet_phy_select .....           | 352 |
| Table 3-477. Function gpio_exti_source_select .....            | 353 |
| Table 3-478. Function gpio_event_output_config .....           | 354 |

|  |     |
|--|-----|
| Table 3-479. Function gpio_event_output_enable .....           | 354 |
| Table 3-480. Function gpio_event_output_disable .....          | 355 |
| Table 3-481. Function gpio_pin_lock .....                      | 355 |
| Table 3-482. HAU Registers .....                               | 356 |
| Table 3-483. HAU firmware function .....                       | 356 |
| Table 3-484. Structure hau_init_parameter_struct .....         | 357 |
| Table 3-485. Structure hau_digest_parameter_struct .....       | 358 |
| Table 3-486. Function hau_deinit .....                         | 358 |
| Table 3-487. Function hau_init .....                           | 358 |
| Table 3-488. Function hau_init_parameter_init .....            | 359 |
| Table 3-489. Function hau_reset .....                          | 359 |
| Table 3-490. Function hau_last_word_validbits_num_config ..... | 360 |
| Table 3-491. Function hau_data_write .....                     | 360 |
| Table 3-492. Function hau_infifo_words_num_get .....           | 361 |
| Table 3-493. Function hau_digest_read .....                    | 361 |
| Table 3-494. Function hau_digest_calculation_enable .....      | 362 |
| Table 3-495. Function hau_multiple_single_dma_config .....     | 362 |
| Table 3-496. Function hau_dma_enable .....                     | 363 |
| Table 3-497. Function hau_dma_disable .....                    | 364 |
| Table 3-498. Function hau_hash_sha_1 .....                     | 364 |
| Table 3-499. Function hau_hmac_sha_1 .....                     | 365 |
| Table 3-500. Function hau_hash_sha_224 .....                   | 365 |
| Table 3-501. Function hau_hmac_sha_224 .....                   | 366 |
| Table 3-502. Function hau_hash_sha_256 .....                   | 367 |
| Table 3-503. Function hau_hmac_sha_256 .....                   | 368 |
| Table 3-504. Function hau_hash_md5 .....                       | 368 |
| Table 3-505. Function hau_hmac_md5 .....                       | 369 |
| Table 3-506. Function hau_flag_get .....                       | 370 |
| Table 3-507. Function hau_flag_clear .....                     | 370 |
| Table 3-508. Function hau_interrupt_enable .....               | 371 |
| Table 3-509. Function hau_interrupt_disable .....              | 372 |
| Table 3-510. Function hau_interrupt_flag_get .....             | 372 |
| Table 3-511. Function hau_interrupt_flag_clear .....           | 373 |
| Table 3-512. I2C Registers .....                               | 374 |
| Table 3-513. I2C firmware function .....                       | 374 |
| Table 3-514. Function i2c_deinit .....                         | 375 |
| Table 3-515. Function i2c_clock_config .....                   | 375 |
| Table 3-516. Function i2c_mode_addr_config .....               | 376 |
| Table 3-517. Function i2c_smbus_type_config .....              | 377 |
| Table 3-518. Function i2c_ack_config .....                     | 377 |
| Table 3-519. Function i2c_ackpos_config .....                  | 378 |
| Table 3-520. Function i2c_master_addressing .....              | 379 |
| Table 3-521. Function i2c_dualaddr_enable .....                | 379 |
| Table 3-522. Function i2c_dualaddr_disable .....               | 380 |

|  |     |
|--|-----|
| Table 3-523. Function i2c_enable .....                         | 380 |
| Table 3-524. Function i2c_disable .....                        | 381 |
| Table 3-525. Function i2c_start_on_bus .....                   | 381 |
| Table 3-526. Function i2c_stop_on_bus .....                    | 382 |
| Table 3-527. Function i2c_data_transmit .....                  | 382 |
| Table 3-528. Function i2c_data_receive .....                   | 383 |
| Table 3-529. Function i2c_dma_enable .....                     | 383 |
| Table 3-530. Function i2c_dma_last_transfer_config .....       | 384 |
| Table 3-531. Function i2c_stretch_scl_low_config .....         | 385 |
| Table 3-532. Function i2c_slave_response_to_gcall_config ..... | 385 |
| Table 3-533. Function i2c_software_reset_config .....          | 386 |
| Table 3-534. Function i2c_pec_enable .....                     | 387 |
| Table 3-535. Function i2c_pec_transfer_enable .....            | 387 |
| Table 3-536. Function i2c_pec_value_get .....                  | 388 |
| Table 3-537. Function i2c_smbus_issue_alert .....              | 388 |
| Table 3-538. Function i2c_smbus_arp_enable .....               | 389 |
| Table 3-539. Function i2c_flag_get .....                       | 390 |
| Table 3-540. Function i2c_flag_clear .....                     | 391 |
| Table 3-541. Function i2c_interrupt_enable .....               | 391 |
| Table 3-542. Function i2c_interrupt_disable .....              | 392 |
| Table 3-543. Function i2c_interrupt_flag_get .....             | 393 |
| Table 3-544. Function i2c_interrupt_flag_clear .....           | 394 |
| Table 3-545. NVIC Registers .....                              | 395 |
| Table 3-546. SysTick Registers .....                           | 396 |
| Table 3-547. IRQn_Type .....                                   | 396 |
| Table 3-548. MISC firmware function .....                      | 398 |
| Table 3-549. Function nvic_priority_group_set .....            | 398 |
| Table 3-550. Function nvic_irq_enable .....                    | 399 |
| Table 3-551. Function nvic_irq_disable .....                   | 400 |
| Table 3-552. Function nvic_vector_table_set .....              | 400 |
| Table 3-553. Function system_lowpower_set .....                | 401 |
| Table 3-554. Function system_lowpower_reset .....              | 401 |
| Table 3-555. Function systick_clksource_set .....              | 402 |
| Table 3-556. PMU Registers .....                               | 403 |
| Table 3-557. PMU firmware function .....                       | 403 |
| Table 3-558. Function pmu_deinit .....                         | 404 |
| Table 3-559. Function pmu_lvd_select .....                     | 404 |
| Table 3-560. Function pmu_lvd_disable .....                    | 405 |
| Table 3-561. Function pmu_to_sleepmode .....                   | 405 |
| Table 3-562. Function pmu_to_deepsleepmode .....               | 406 |
| Table 3-563. Function pmu_to_standbymode .....                 | 406 |
| Table 3-564. Function pmu_wakeup_pin_enable .....              | 407 |
| Table 3-565. Function pmu_wakeup_pin_disable .....             | 407 |
| Table 3-566. Function pmu_backup_write_enable .....            | 408 |

---

|  |     |
|--|-----|
| Table 3-567. Function pmu_backup_write_disable .....         | 408 |
| Table 3-568. Function pmu_flag_get.....                      | 409 |
| Table 3-569. Function pmu_flag_clear.....                    | 409 |
| Table 3-570. RCU Registers .....                             | 410 |
| Table 3-571. RCU firmware function .....                     | 411 |
| Table 3-572. rcu_periph_enum.....                            | 412 |
| Table 3-573. rcu_periph_sleep_enum.....                      | 413 |
| Table 3-574. rcu_periph_reset_enum .....                     | 413 |
| Table 3-575. rcu_flag_enum .....                             | 414 |
| Table 3-576. rcu_int_flag_enum.....                          | 414 |
| Table 3-577. rcu_int_flag_clear_enum.....                    | 415 |
| Table 3-578. rcu_int_enum.....                               | 415 |
| Table 3-579. rcu_osc1_type_enum.....                         | 416 |
| Table 3-580. rcu_clock_freq_enum .....                       | 416 |
| Table 3-581. Function rcu_deinit .....                       | 416 |
| Table 3-582. Function rcu_periph_clock_enable .....          | 417 |
| Table 3-583. Function rcu_periph_clock_disable .....         | 418 |
| Table 3-584. Function rcu_periph_clock_sleep_enable .....    | 419 |
| Table 3-585. Function rcu_periph_clock_sleep_disable .....   | 420 |
| Table 3-586. Function rcu_periph_reset_enable.....           | 420 |
| Table 3-587. Function rcu_periph_reset_disable .....         | 421 |
| Table 3-588. Function rcu_bkp_reset_enable .....             | 422 |
| Table 3-589. Function rcu_bkp_reset_disable .....            | 423 |
| Table 3-590. Function rcu_system_clock_source_config.....    | 423 |
| Table 3-591. Function rcu_system_clock_source_get .....      | 424 |
| Table 3-592. Function rcu_ahb_clock_config .....             | 424 |
| Table 3-593. Function rcu_apb1_clock_config .....            | 425 |
| Table 3-594. Function rcu_apb2_clock_config .....            | 426 |
| Table 3-595. Function rcu_ckout0_config.....                 | 426 |
| Table 3-596. Function rcu_ckout1_config.....                 | 427 |
| Table 3-597. Function rcu_pll_config .....                   | 428 |
| Table 3-598. Function rcu_pdev0_config .....                 | 429 |
| Table 3-599. Function rcu_pdev1_config .....                 | 430 |
| Table 3-600. Function rcu_pll1_config .....                  | 430 |
| Table 3-601. Function rcu_pll2_config .....                  | 431 |
| Table 3-602. Function rcu_adc_clock_config.....              | 431 |
| Table 3-603. Function rcu_usbf_trng_clock_config.....        | 432 |
| Table 3-604. Function rcu_rtc_clock_config .....             | 432 |
| Table 3-605. Function rcu_i2s1_clock_config.....             | 433 |
| Table 3-606. Function rcu_i2s2_clock_config.....             | 434 |
| Table 3-607. Function rcu_pllt_config .....                  | 434 |
| Table 3-608. Function rcu_pllt_vco_config.....               | 435 |
| Table 3-609. Function rcu_tli_clock_config .....             | 436 |
| Table 3-610. Function rcu_lxtal_drive_capability_config..... | 436 |

---

|   |     |
|---|-----|
| Table 3-611. Function rcu_osc1_stab_wait.....               | 437 |
| Table 3-612. Function rcu_osc1_on .....                     | 438 |
| Table 3-613. Function rcu_osc1_off.....                     | 438 |
| Table 3-614. Function rcu_osc1_bypass_mode_enable .....     | 439 |
| Table 3-615. Function rcu_osc1_bypass_mode_disable .....    | 439 |
| Table 3-616. Function rcu_hxtal_clock_monitor_enable .....  | 440 |
| Table 3-617. Function rcu_hxtal_clock_monitor_disable ..... | 440 |
| Table 3-618. Function rcu_irc8m_adjust_value_set.....       | 441 |
| Table 3-619. Function rcu_deepsleep_voltage_set.....        | 441 |
| Table 3-620. Function rcu_clock_freq_get.....               | 442 |
| Table 3-621. Function rcu_flag_get.....                     | 443 |
| Table 3-622. Function rcu_all_reset_flag_clear .....        | 444 |
| Table 3-623. Function rcu_interrupt_enable.....             | 444 |
| Table 3-624. Function rcu_interrupt_disable.....            | 445 |
| Table 3-625. Function rcu_interrupt_flag_get .....          | 445 |
| Table 3-626. Function rcu_interrupt_flag_clear .....        | 446 |
| Table 3-627. RTC Registers .....                            | 448 |
| Table 3-628. RTC firmware function .....                    | 448 |
| Table 3-629. Function rtc_configuration_mode_enter.....     | 448 |
| Table 3-630. Function rtc_configuration_mode_exit .....     | 449 |
| Table 3-631. Function rtc_lwoff_wait .....                  | 449 |
| Table 3-632. Function rtc_register_sync_wait .....          | 450 |
| Table 3-633. Function rtc_counter_get.....                  | 450 |
| Table 3-634. Function rtc_counter_set.....                  | 451 |
| Table 3-635. Function rtc_prescaler_set .....               | 451 |
| Table 3-636. Function rtc_alarm_config .....                | 452 |
| Table 3-637. Function rtc_divider_get .....                 | 453 |
| Table 3-638. Function rtc_flag_get .....                    | 453 |
| Table 3-639. Function rtc_flag_clear .....                  | 454 |
| Table 3-640. Function rtc_interrupt_enable .....            | 454 |
| Table 3-641. Function rtc_interrupt_disable .....           | 455 |
| Table 3-642. SDIO Registers .....                           | 456 |
| Table 3-643. SDIO firmware function .....                   | 456 |
| Table 3-644. Function sdio_deinit .....                     | 458 |
| Table 3-645. Function sdio_clock_config .....               | 458 |
| Table 3-646. Function sdio_hardware_clock_enable .....      | 459 |
| Table 3-647. Function sdio_hardware_clock_disable .....     | 460 |
| Table 3-648. Function sdio_bus_mode_set .....               | 460 |
| Table 3-649. Function sdio_power_state_set.....             | 461 |
| Table 3-650. Function sdio_power_state_get.....             | 461 |
| Table 3-651. Function sdio_clock_enable.....                | 462 |
| Table 3-652. Function sdio_clock_disable.....               | 462 |
| Table 3-653. Function sdio_command_response_config .....    | 463 |
| Table 3-654. Function sdio_wait_type_set.....               | 464 |

---

|   |     |
|---|-----|
| Table 3-655. Function sdio_csm_enable .....                       | 464 |
| Table 3-656. Function sdio_csm_disable.....                       | 465 |
| Table 3-657. Function sdio_command_index_get.....                 | 465 |
| Table 3-658. Function sdio_response_get.....                      | 466 |
| Table 3-659. Function sdio_data_config .....                      | 466 |
| Table 3-660. Function sdio_data_transfer_config .....             | 468 |
| Table 3-661. Function sdio_dsm_enable.....                        | 468 |
| Table 3-662. Function sdio_dsm_disable.....                       | 469 |
| Table 3-663. Function sdio_data_write .....                       | 469 |
| Table 3-664. Function sdio_data_read.....                         | 470 |
| Table 3-665. Function sdio_data_counter_get .....                 | 470 |
| Table 3-666. Function sdio_data_counter_get.....                  | 471 |
| Table 3-667. Function sdio_dma_enable.....                        | 471 |
| Table 3-668. Function sdio_dma_disable.....                       | 472 |
| Table 3-669. Function sdio_readwait_enable .....                  | 472 |
| Table 3-670. Function sdio_readwait_disable .....                 | 473 |
| Table 3-671. Function sdio_stop_readwait_enable .....             | 473 |
| Table 3-672. Function sdio_stop_readwait_disable .....            | 474 |
| Table 3-673. Function sdio_readwait_type_set.....                 | 474 |
| Table 3-674. Function sdio_operation_enable .....                 | 475 |
| Table 3-675. Function sdio_operation_disable .....                | 475 |
| Table 3-676. Function sdio_suspend_enable .....                   | 476 |
| Table 3-677. Function sdio_suspend_disable.....                   | 476 |
| Table 3-678. Function sdio_ceata_command_enable .....             | 477 |
| Table 3-679. Function sdio_ceata_command_disable .....            | 477 |
| Table 3-680. Function sdio_ceata_interrupt_enable .....           | 478 |
| Table 3-681. Function sdio_ceata_interrupt_disable .....          | 478 |
| Table 3-682. Function sdio_ceata_command_completion_enable .....  | 479 |
| Table 3-683. Function sdio_ceata_command_completion_disable ..... | 479 |
| Table 3-684. Function sdio_flag_get .....                         | 480 |
| Table 3-685. Function sdio_flag_clear.....                        | 481 |
| Table 3-686. Function sdio_interrupt_enable .....                 | 482 |
| Table 3-687. Function sdio_interrupt_disable .....                | 483 |
| Table 3-688. Function sdio_interrupt_flag_get .....               | 484 |
| Table 3-689. Function sdio_interrupt_flag_clear .....             | 486 |
| Table 3-690. SPI/I2S Registers .....                              | 487 |
| Table 3-691. SPI/I2S firmware function.....                       | 487 |
| Table 3-692. spi_parameter_struct .....                           | 488 |
| Table 3-693. Function spi_i2s_deinit .....                        | 489 |
| Table 3-694. Function spi_struct_para_init .....                  | 489 |
| Table 3-695. Function spi_init .....                              | 490 |
| Table 3-696. Function spi_enable .....                            | 491 |
| Table 3-697. Function spi_disable .....                           | 491 |
| Table 3-698. Function i2s_init .....                              | 492 |

---

|   |     |
|---|-----|
| Table 3-699. Function i2s_psc_config .....                    | 493 |
| Table 3-700. Function i2s_enable .....                        | 494 |
| Table 3-701. Function i2s_disable .....                       | 494 |
| Table 3-702. Function spi_nss_output_enable .....             | 495 |
| Table 3-703. Function spi_nss_output_disable .....            | 495 |
| Table 3-704. Function spi_nss_internal_high .....             | 496 |
| Table 3-705. Function spi_nss_internal_low .....              | 496 |
| Table 3-706. Function spi_dma_enable .....                    | 497 |
| Table 3-707. Function spi_dma_disable .....                   | 498 |
| Table 3-708. Function spi_i2s_data_frame_format_config .....  | 498 |
| Table 3-709. Function spi_bidirectional_transfer_config ..... | 499 |
| Table 3-710. Function spi_i2s_data_transmit .....             | 500 |
| Table 3-711. Function spi_i2s_data_receive .....              | 500 |
| Table 3-712. Function spi_crc_polynomial_set .....            | 501 |
| Table 3-713. Function spi_crc_polynomial_get .....            | 501 |
| Table 3-714. Function spi_crc_on .....                        | 502 |
| Table 3-715. Function spi_crc_off .....                       | 502 |
| Table 3-716. Function spi_crc_next .....                      | 503 |
| Table 3-717. Function spi_crc_get .....                       | 503 |
| Table 3-718. Function spi_quad_enable .....                   | 504 |
| Table 3-719. Function spi_quad_enable .....                   | 504 |
| Table 3-720. Function spi_quad_write_enable .....             | 505 |
| Table 3-721. Function spi_quad_read_enable .....              | 505 |
| Table 3-722. Function spi_quad_io23_output_enable .....       | 506 |
| Table 3-723. Function spi_quad_io23_output_disable .....      | 506 |
| Table 3-724. Function spi_i2s_flag_get .....                  | 507 |
| Table 3-725. Function spi_i2s_interrupt_enable .....          | 508 |
| Table 3-726. Function spi_i2s_interrupt_disable .....         | 509 |
| Table 3-727. Function spi_i2s_interrupt_flag_get .....        | 509 |
| Table 3-728. Function spi_crc_error_clear .....               | 510 |
| Table 3-729. TIMERx Registers .....                           | 511 |
| Table 3-730. TIMERx firmware function .....                   | 511 |
| Table 3-731. Structure timer_parameter_struct .....           | 514 |
| Table 3-732. Structure timer_break_parameter_struct .....     | 514 |
| Table 3-733. Structure timer_oc_parameter_struct .....        | 514 |
| Table 3-734. Structure timer_ic_parameter_struct .....        | 515 |
| Table 3-735. Function timer_deinit .....                      | 515 |
| Table 3-736. Function timer_struct_para_init .....            | 515 |
| Table 3-737. Function timer_init .....                        | 516 |
| Table 3-738. Function timer_enable .....                      | 517 |
| Table 3-739. Function timer_disable .....                     | 517 |
| Table 3-740. Function timer_auto_reload_shadow_enable .....   | 518 |
| Table 3-741. Function timer_auto_reload_shadow_disable .....  | 518 |
| Table 3-742. Function timer_update_event_enable .....         | 519 |

---

|  |     |
|--|-----|
| Table 3-743. Function timer_update_event_disable .....                         | 519 |
| Table 3-744. Function timer_counter_alignment .....                            | 520 |
| Table 3-745. Function timer_counter_up_direction .....                         | 521 |
| Table 3-746. timer_counter_down_direction .....                                | 521 |
| Table 3-747. Function timer_prescaler_config .....                             | 522 |
| Table 3-748. Function timer_repetition_value_config .....                      | 522 |
| Table 3-749. Function timer_autoreload_value_config .....                      | 523 |
| Table 3-750. Function timer_counter_value_config .....                         | 524 |
| Table 3-751. Function timer_counter_read .....                                 | 524 |
| Table 3-752. Function timer_prescaler_read .....                               | 525 |
| Table 3-753. Function timer_single_pulse_mode_config .....                     | 525 |
| Table 3-754. Function timer_update_source_config .....                         | 526 |
| Table 3-755. Function timer_dma_enable .....                                   | 527 |
| Table 3-756. Function timer_dma_disable .....                                  | 527 |
| Table 3-757. Function timer_channel_dma_request_source_select .....            | 528 |
| Table 3-758. Function timer_dma_transfer_config .....                          | 529 |
| Table 3-759. Function timer_event_software_generate .....                      | 530 |
| Table 3-760. Function timer_break_struct_para_init .....                       | 531 |
| Table 3-761. Function timer_break_config .....                                 | 532 |
| Table 3-762. Function timer_break_enable .....                                 | 533 |
| Table 3-763. Function timer_break_disable .....                                | 533 |
| Table 3-764. Function timer_automatic_output_enable .....                      | 534 |
| Table 3-765. Function timer_automatic_output_disable .....                     | 534 |
| Table 3-766. Function timer_primary_output_config .....                        | 535 |
| Table 3-767. Function timer_channel_control_shadow_config .....                | 535 |
| Table 3-768. Function timer_channel_control_shadow_update_config .....         | 536 |
| Table 3-769. Function timer_channel_output_struct_para_init .....              | 537 |
| Table 3-770. Function timer_channel_output_config .....                        | 537 |
| Table 3-771. Function timer_channel_output_mode_config .....                   | 538 |
| Table 3-772. Function timer_channel_output_pulse_value_config .....            | 539 |
| Table 3-773. Function timer_channel_output_shadow_config .....                 | 540 |
| Table 3-774. Function timer_channel_output_fast_config .....                   | 541 |
| Table 3-775. Function timer_channel_output_clear_config .....                  | 542 |
| Table 3-776. Function timer_channel_output_polarity_config .....               | 542 |
| Table 3-777. Function timer_channel_complementary_output_polarity_config ..... | 543 |
| Table 3-778. Function timer_channel_output_state_config .....                  | 544 |
| Table 3-779. Function timer_channel_complementary_output_state_config .....    | 545 |
| Table 3-780. Function timer_channel_input_struct_para_init .....               | 546 |
| Table 3-781. Function timer_input_capture_config .....                         | 546 |
| Table 3-782. Function timer_channel_input_capture_prescaler_config .....       | 547 |
| Table 3-783. Function timer_channel_capture_value_register_read .....          | 548 |
| Table 3-784. Function timer_input_pwm_capture_config .....                     | 549 |
| Table 3-785. Function timer_hall_mode_config .....                             | 549 |
| Table 3-786. Function timer_input_trigger_source_select .....                  | 550 |

---

|   |     |
|---|-----|
| Table 3-787. Function timer_master_output_trigger_source_select .....       | 551 |
| Table 3-788. Function timer_slave_mode_select .....                         | 552 |
| Table 3-789. Function timer_master_slave_mode_config .....                  | 553 |
| Table 3-790. Function timer_external_trigger_config .....                   | 554 |
| Table 3-791. Function timer_quadrature_decoder_mode_config.....             | 555 |
| Table 3-792. Function timer_internal_clock_config .....                     | 556 |
| Table 3-793. Function timer_internal_trigger_as_external_clock_config ..... | 556 |
| Table 3-794. Function timer_external_trigger_as_external_clock_config ..... | 557 |
| Table 3-795. Function timer_external_clock_mode0_config .....               | 558 |
| Table 3-796. Function timer_external_clock_mode1_config .....               | 559 |
| Table 3-797. Function timer_external_clock_mode1_disable .....              | 560 |
| Table 3-798. Function timer_flag_get .....                                  | 560 |
| Table 3-799. Function timer_flag_clear .....                                | 561 |
| Table 3-800. Function timer_interrupt_enable .....                          | 562 |
| Table 3-801. Function timer_interrupt_disable .....                         | 563 |
| Table 3-802. Function timer_interrupt_flag_get.....                         | 563 |
| Table 3-803. Function timer_interrupt_flag_clear.....                       | 564 |
| Table 3-804. TLI Registers .....  | 565 |
| Table 3-805. TLI firmware function .....                                    | 566 |
| Table 3-806. Structure tli_parameter_struct .....                           | 567 |
| Table 3-807. Structure tli_layer_parameter_struct .....                     | 567 |
| Table 3-808. Structure tli_layer_lut_parameter_struct .....                 | 568 |
| Table 3-809. Enumeration tli_layer_ppf_enum.....                            | 568 |
| Table 3-810. Function tli_deinit.....                                       | 568 |
| Table 3-811. Function tli_init .....  | 569 |
| Table 3-812. Function tli_dither_config .....                               | 570 |
| Table 3-813. Function tli_enable .....                                      | 570 |
| Table 3-814. Function tli_disable .....                                     | 571 |
| Table 3-815. Function tli_reload_config .....                               | 571 |
| Table 3-816. Function tli_line_mark_set.....                                | 572 |
| Table 3-817. Function tli_current_pos_get .....                             | 572 |
| Table 3-818. Function tli_layer_enable .....                                | 573 |
| Table 3-819. Function tli_layer_disable .....                               | 573 |
| Table 3-820. Function tli_color_key_enable .....                            | 574 |
| Table 3-821. Function tli_color_key_disable .....                           | 574 |
| Table 3-822. Function tli_lut_enable .....                                  | 575 |
| Table 3-823. Function tli_lut_disable .....                                 | 575 |
| Table 3-824. Function tli_layer_init .....                                  | 576 |
| Table 3-825. Function tli_layer_window_offset_modify .....                  | 577 |
| Table 3-826. Function tli_lut_init .....                                    | 578 |
| Table 3-827. Function tli_ckey_init.....                                    | 579 |
| Table 3-828. Function tli_flag_get .....                                    | 579 |
| Table 3-829. Function tli_interrupt_enable .....                            | 580 |
| Table 3-830. Function tli_interrupt_disable .....                           | 581 |

---

|   |     |
|---|-----|
| Table 3-831. Function tli_interrupt_flag_get.....                   | 581 |
| Table 3-832. Function tli_interrupt_flag_clear.....                 | 582 |
| Table 3-833. TRNG Registers .....                                   | 583 |
| Table 3-834. TRNG firmware function.....                            | 583 |
| Table 3-835. Enumeration trng_flag_enum.....                        | 583 |
| Table 3-836. Enumeration trng_int_flag_enum .....                   | 583 |
| Table 3-837. Function trng_deinit.....                              | 584 |
| Table 3-838. Function trng_enable .....                             | 584 |
| Table 3-839. Function trng_disable .....                            | 584 |
| Table 3-840. Function trng_get_true_random_data .....               | 585 |
| Table 3-841. Function trng_flag_get .....                           | 585 |
| Table 3-842. Function trng_interrupt_enable .....                   | 586 |
| Table 3-843. Function trng_interrupt_disable .....                  | 587 |
| Table 3-844. Function trng_interrupt_flag_get.....                  | 587 |
| Table 3-845. Function trng_interrupt_flag_clear.....                | 588 |
| Table 3-846. USART Registers .....                                  | 588 |
| Table 3-847. USART firmware function.....                           | 589 |
| Table 3-848. usart_flag_enum .....                                  | 590 |
| Table 3-849. usart_interrupt_flag_enum .....                        | 591 |
| Table 3-850. usart_interrupt_enum .....                             | 591 |
| Table 3-851. usart_invert_enum.....                                 | 591 |
| Table 3-852. Function usart_deinit .....                            | 592 |
| Table 3-853. Function usart_baudrate_set .....                      | 592 |
| Table 3-854. Function usart_parity_config .....                     | 593 |
| Table 3-855. Function usart_word_length_set.....                    | 593 |
| Table 3-856. Function usart_stop_bit_set.....                       | 594 |
| Table 3-857. Function usart_enable .....                            | 595 |
| Table 3-858. Function usart_disable .....                           | 595 |
| Table 3-859. Function usart_transmit_config.....                    | 596 |
| Table 3-860. Function usart_receive_config .....                    | 596 |
| Table 3-861. Function usart_data_first_config .....                 | 597 |
| Table 3-862. Function usart_invert_config .....                     | 598 |
| Table 3-863. Function usart_receiver_timeout_enable.....            | 598 |
| Table 3-864. Function usart_receiver_timeout_disable.....           | 599 |
| Table 3-865. Function usart_receiver_timeout_threshold_config ..... | 599 |
| Table 3-866. Function usart_data_transmit .....                     | 600 |
| Table 3-867. Function usart_data_receive .....                      | 600 |
| Table 3-868. Function usart_address_config .....                    | 601 |
| Table 3-869. Function usart_mute_mode_enable.....                   | 602 |
| Table 3-870. Function usart_mute_mode_disable.....                  | 602 |
| Table 3-871. Function usart_mute_mode_wakeup_config .....           | 603 |
| Table 3-872. Function usart_lin_mode_enable .....                   | 603 |
| Table 3-873. Function usart_lin_mode_disable .....                  | 604 |
| Table 3-874. Function usart_lin_break_dection_length_config .....   | 604 |

---

|   |     |
|---|-----|
| Table 3-875. Function usart_send_break .....                  | 605 |
| Table 3-876. Function usart_halfduplex_enable .....           | 605 |
| Table 3-877. Function usart_halfduplex_disable .....          | 606 |
| Table 3-878. Function usart_synchronous_clock_enable .....    | 607 |
| Table 3-879. Function usart_synchronous_clock_disable .....   | 607 |
| Table 3-880. Function usart_synchronous_clock_config .....    | 608 |
| Table 3-881. Function usart_guard_time_config .....           | 608 |
| Table 3-882. Function usart_smartcard_mode_enable .....       | 609 |
| Table 3-883. Function usart_smartcard_mode_disable .....      | 609 |
| Table 3-884. Function usart_smartcard_mode_nack_enable .....  | 610 |
| Table 3-885. Function usart_smartcard_mode_nack_disable ..... | 610 |
| Table 3-886. Function usart_smartcard_autoretry_config .....  | 611 |
| Table 3-887. Function usart_block_length_config .....         | 612 |
| Table 3-888. Function usart_irda_mode_enable .....            | 612 |
| Table 3-889. Function usart_irda_mode_disable .....           | 613 |
| Table 3-890. Function usart_prescaler_config .....            | 613 |
| Table 3-891. Function usart_irda_lowpower_config .....        | 614 |
| Table 3-892. Function usart_hardware_flow_rts_config .....    | 614 |
| Table 3-893. Function usart_hardware_flow_cts_config .....    | 615 |
| Table 3-894. Function usart_dma_enable .....                  | 616 |
| Table 3-895. Function usart_dma_disable .....                 | 616 |
| Table 3-896. Function usart_flag_get .....                    | 617 |
| Table 3-897. Function usart_flag_clear .....                  | 618 |
| Table 3-898. Function usart_interrupt_enable .....            | 618 |
| Table 3-899. Function usart_interrupt_disable .....           | 619 |
| Table 3-900. Function usart_interrupt_flag_get .....          | 620 |
| Table 3-901. Function usart_interrupt_flag_clear .....        | 620 |
| Table 3-902. WWDGT Registers .....                            | 621 |
| Table 3-903. WWDGT firmware function .....                    | 621 |
| Table 3-904. Function wwdgt_deinit .....                      | 622 |
| Table 3-905. Function wwdgt_enable .....                      | 622 |
| Table 3-906. Function wwdgt_counter_update .....              | 623 |
| Table 3-907. Function wwdgt_config .....                      | 623 |
| Table 3-908. Function wwdgt_flag_get .....                    | 624 |
| Table 3-909. Function wwdgt_flag_clear .....                  | 624 |
| Table 3-910. Function wwdgt_interrupt_enable .....            | 625 |
| Table 3-911. Function wwdgt_interrupt_flag_get .....          | 625 |
| Table 4-1. Revision history .....                             | 627 |

## 1. Introduction

This manual introduces firmware library of GD32F20x 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 GD32F20x 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                |
| CAN         | Controller area network         |
| CAU         | Cryptographic Acceleration Unit |
| CRC         | CRC calculation unit            |

| Peripherals | Descriptions   |
|-------------|--|
| DAC         | Digital-to-analog converter                              |
| DBG         | Debug  |
| DCI         | Digital camera interface                                 |
| DMA         | Direct memory access controller                          |
| ENET        | Ethernet   |
| EXMC        | External memory controller                               |
| EXTI        | Interrupt/event controller                               |
| FMC         | Flash memory controller                                  |
| FWDGT       | Free watchdog timer                                      |
| GPIO/APIO   | General-purpose and alternate-function I/Os              |
| HAU         | Hash Acceleration Unit                                   |
| I2C         | Inter-integrated circuit interface                       |
| MISC        | Nested Vectored Interrupt Controller                     |
| PMU         | Power management unit                                    |
| RCU         | Reset and clock unit                                     |
| RTC         | Real-time Clock  |
| SDIO        | Secure digital input/output interface                    |
| SPI/I2S     | Serial peripheral interface/Inter-IC sound               |
| TIMER       | TIMER  |
| TLI         | TFT-LCD interface  |
| TRNG        | True random number generator                             |
| 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 “gd32f20x\_”, such as: gd32f20x\_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;

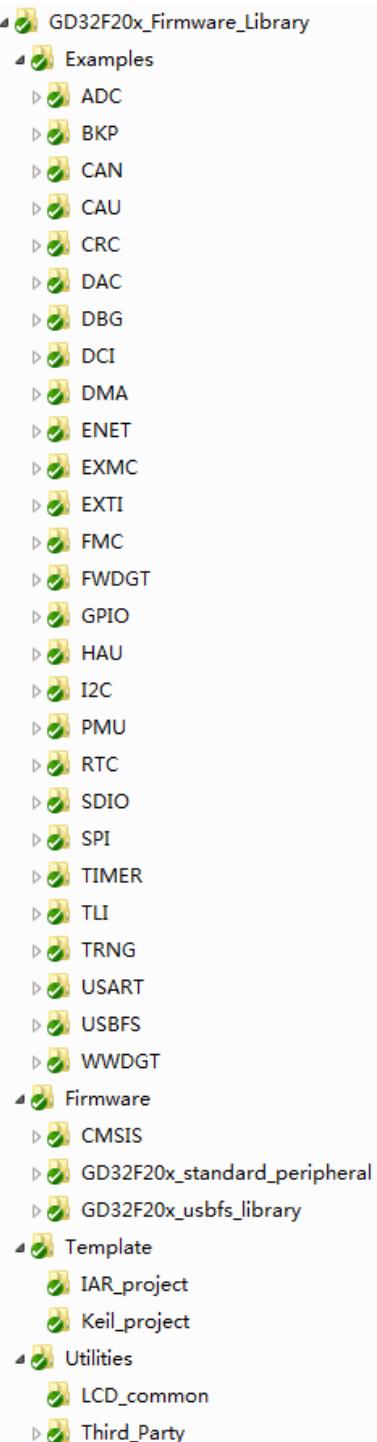
- The naming of peripheral functions are started with the peripheral abbreviation added with an underline, when the function name is concluded by several words, underscores should be adapted among words, and all the peripheral functions are written in lowercase.

## 2. Firmware Library Overview

### 2.1. File Structure of Firmware Library

GD32F20x\_Firmware\_Library, the file structure is shown as below:

**Figure 2-1. File structure of firmware library of GD32F20x**



### 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;
- gd32f20x\_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);
- gd32f20x\_it.c: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- gd32f20x\_it.h: the header file include all the prototypes of the interrupt service routines;
- systick.c: the source file include the precise time delay functions by using systick;
- systick.h: the header file include the prototype of the precise time delay functions by using systick;
- main.c: example code. Note: all the examples are not influenced by software IDEs.

### 2.1.2. Firmware Folder

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

- CMSIS subfolder includes the Cortex M3 kernel support files, the startup file based on the Cortex M3 kernel processor, the global header file of GD32F20x and system configuration file;
- GD32F20x\_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;
- GD32F20x\_usbfs\_driver 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.

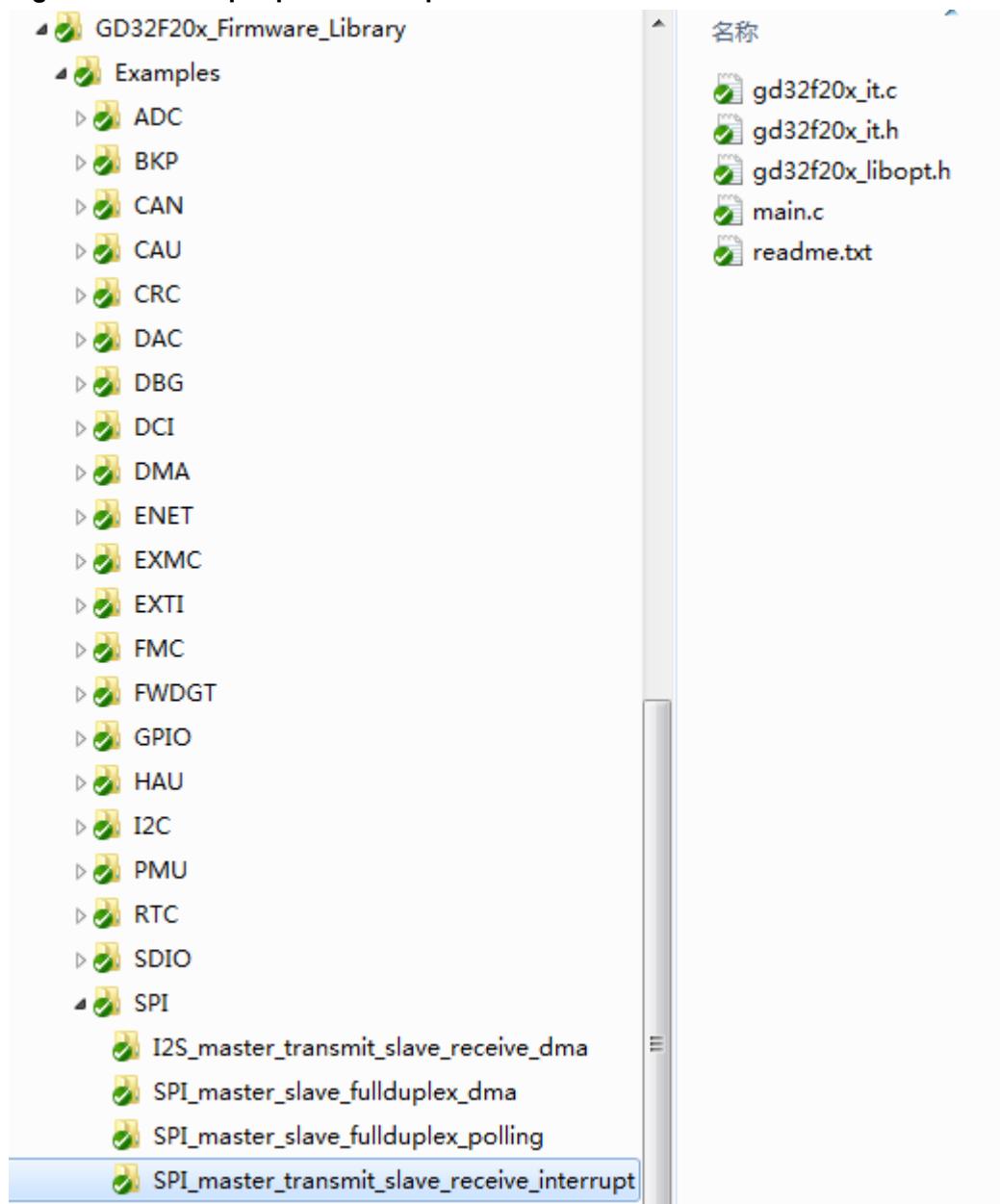
### 2.1.3. Template Folder

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 firmware examples, the steps are shown as below:

#### Select files

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

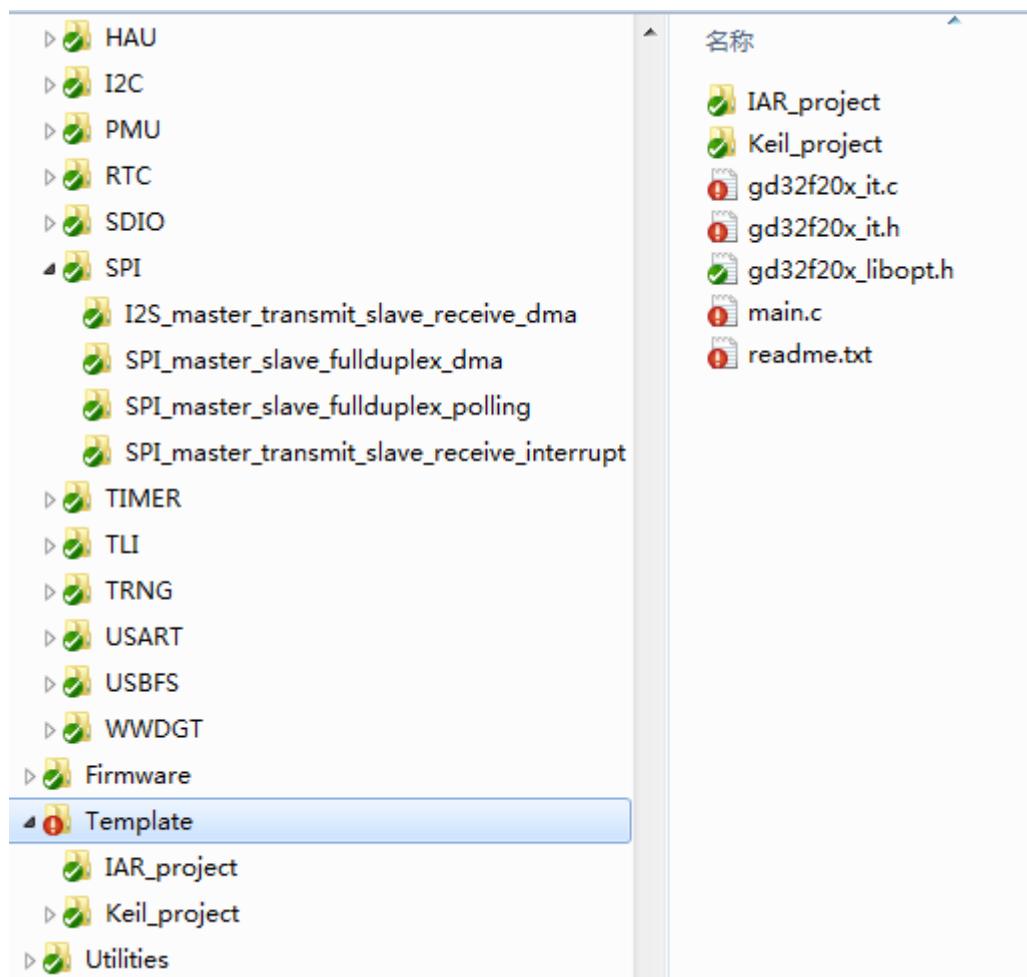
Figure 2-2. Select peripheral example files



### Copy files

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

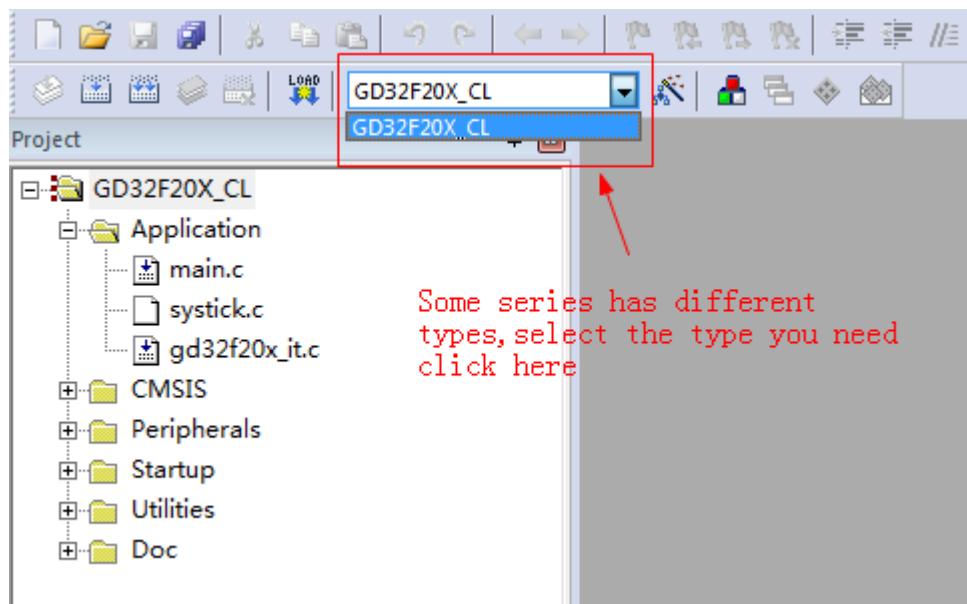
Figure 2-3. Copy the peripheral example files



### Open a project

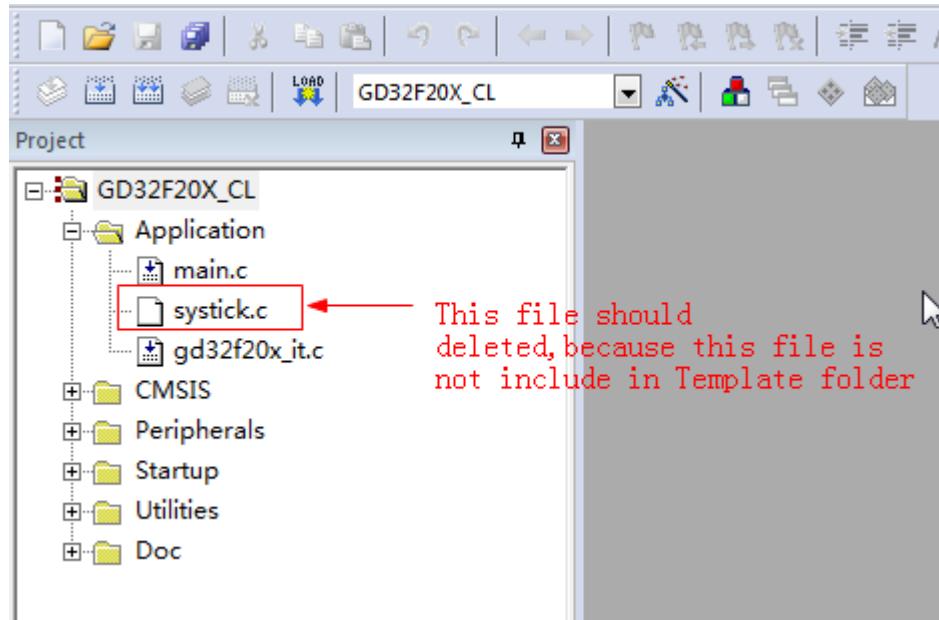
GD provides project in Keil and IAR, users can open project in different IDEs according to their need, such as “Keil\_project”, open \Template\Keil\_project\Project.uvproj, shown as below:

Figure 2-4. Open the project file



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

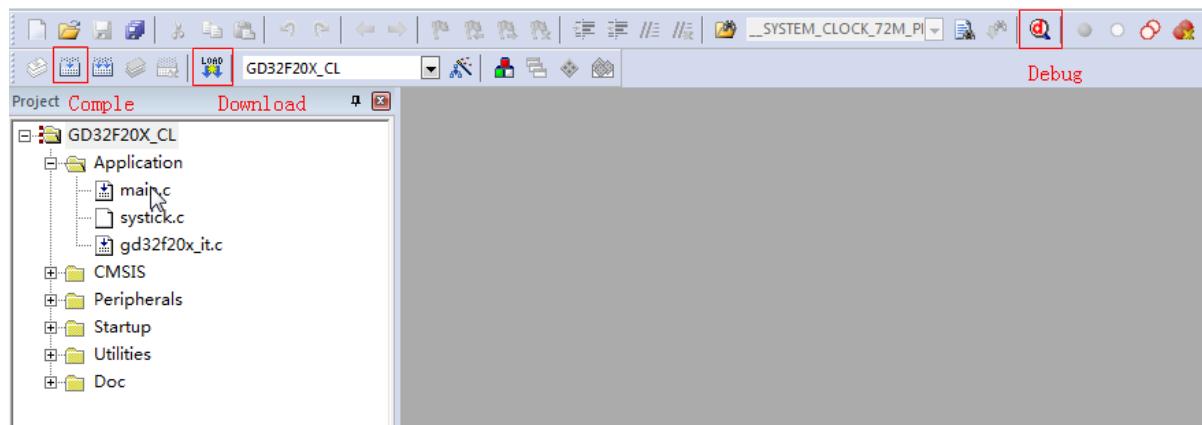
Figure 2-5. Configure project files



### Compile-Debug-Download

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

Figure 2-6. Compile-debug-download



## 2.1.4. Utilities Folder

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

- LCD\_Common and Third\_Party subfolders include files for USB tests;
- gd32f20x\_eval.h and gd32f20x\_lcd\_eval.h are related header files of the evaluation board about running the firmware examples;
- gd32f20x\_eval.c and gd32f20x\_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   |
|-------------------|--|
| gd32f20x_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.  |
| gd32f20x_it.h     | Header file, including all the prototypes of interrupt service routines.   |
| gd32f20x_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. |

|                |  |
|----------------|--|
| gd32f20x_xxx.h | The header file of peripheral PPP, including functions about peripheral PPP, and the variables used for functions. |
| gd32f20x_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 functin       |
| <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**

| Registers | Descriptions    |
|-----------|-----------------|
| ADC_STAT  | status register |

| Registers          | Descriptions                            |
|--------------------|---|
| 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 |
| ADC_WDHT           | watchdog high threshold register        |
| ADC_WDLT           | watchdog low threshold register         |
| ADC_RSQ0           | regular sequence register 0             |
| ADC_RSQ1           | regular sequence register 1             |
| ADC_RSQ2           | regular sequence register 2             |
| ADC_ISQ            | inserted sequence register              |
| ADC_IDATAx         | inserted data register x                |
| 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 mode(only for ADC0)                                   |
| 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_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                                    |

| Function name                      | Function description  |
|------------------------------------|---|
| 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 |
| adc_watchdog_single_channel_enable | configure ADC analog watchdog single channel                    |
| adc_watchdog_group_channel_enable  | configure ADC analog watchdog group channel                     |
| adc_watchdog_disable               | disable ADC analog watchdog                                     |
| adc_watchdog_threshold_config      | configure ADC analog watchdog threshold                         |
| adc_resolution_config              | configure ADC resolution  |
| adc_oversample_mode_config         | configure ADC oversample mode                                   |
| adc_oversample_mode_enable         | enable ADC oversample mode                                      |
| adc_oversample_mode_disable        | disable ADC oversample mode                                     |
| adc_flag_get                       | get the ADC flag bits   |
| adc_flag_clear                     | clear the ADC flag bits   |
| adc_interrupt_enable               | enable ADC interrupt  |
| adc_interrupt_disable              | disable ADC interrupt   |
| adc_interrupt_flag_get             | get the ADC interrupt bits                                      |
| adc_interrupt_flag_clear           | clear the ADC flag  |

### adc\_deinit

The description of adc\_deinit is shown as below:

Table 3-4. Function adc\_deinit

|                              |  |
|------------------------------|--|
| Function name                | adc_deinit   |
| Function prototype           | void adc_deinit(uint32_t adc_periph);              |
| Function descriptions        | reset ADCx peripheral                              |
| Precondition                 | -  |
| The called functions         | rcu_periph_reset_enable / rcu_periph_reset_disable |
| <b>Input parameter{in}</b>   |  |
| adc_periph                   | ADC peripheral                                     |
| ADCx                         | x=0,1,2  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

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>   | adc_mode_config  |
| <b>Function prototype</b>                                      | void adc_mode_config(uint32_t mode);                                     |
| <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   |
| <i>ADC_MODE_FREE</i>   | all the ADC work independently   |
| <i>ADC_DAUL_REGULAR_PARALLEL_INSERTED_D_PARALLEL</i>           | ADC0 and ADC1 work in combined regular parallel + inserted parallel mode |
| <i>ADC_DAUL_REGULAR_PARALLEL_INSERTED_D_ROTATION</i>           | ADC0 and ADC1 work in combined regular parallel + trigger rotation mode  |
| <i>ADC_DAUL_INSERTED_D_PARALLEL_REGULAR_AL_FOLLOWUP_FAST</i>   | ADC0 and ADC1 work in combined inserted parallel + follow-up fast mode   |
| <i>ADC_DAUL_INSERTED_D_PARALLEL_REGULAR_AL_FOLLOWUP_SLOW_W</i> | ADC0 and ADC1 work in combined inserted parallel + follow-up slow mode   |
| <i>ADC_DAUL_INSERTED_D_PARALLEL</i>                            | ADC0 and ADC1 work in inserted parallel mode only                        |
| <i>ADC_DAUL_REGULAR_PARALLEL</i>                               | ADC0 and ADC1 work in regular parallel mode only                         |
| <i>ADC_DAUL_REGULAR_FOLLOWUP_FAST</i>                          | ADC0 and ADC1 work in follow-up fast mode only                           |
| <i>ADC_DAUL_REGULAR_FOLLOWUP_SLOW</i>                          | ADC0 and ADC1 work in follow-up slow mode only                           |
| <i>ADC_DAUL_INSERTED_D_TRIGGER_ROTATION</i>                    | 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 new_value); |
| <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   |
| <b>ADCx</b>                      | x=0,1,2  |
| <b>Input parameter{in}</b>       |  |
| <b>function</b>                  | the function to config   |
| <b>ADC_SCAN_MODE</b>             | scan mode select   |
| <b>ADC_INSERTED_CHANNEL_AUTO</b> | inserted channel group convert automatically   |
| <b>ADC_CONTINUOUS_MODE</b>       | continuous mode select   |
| <b>Input parameter{in}</b>       |  |
| <b>newvalue</b>                  | control value  |
| <b>ENABLE</b>                    | enable function  |
| <b>DISABLE</b>                   | disable function   |
| <b>Output parameter{out}</b>     |  |
| -                                | -  |
| <b>Return value</b>              |  |
| -                                | -  |

Example:

```
/* enable 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                |
| <b>ADCx</b>                  | x=0,1,2                       |
| <b>Input parameter{in}</b>   |                               |
| <b>data_alignment</b>        | data alignment select         |
| <b>ADC_DATAALIGN_RIGHT</b>   | LSB alignment                 |
| <b>ADC_DATAALIGN_LEFT</b>    | 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                        |
| <b>ADCx</b>                  | x=0,1,2                               |
| <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                         |
| <b>ADCx</b>                  | x=0,1,2                                |
| <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                                    |
| <b>ADCx</b>                  | x=0,1,2   |
| <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\_dma\_mode\_enable**

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

**Table 3-13. 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                                 |
| <b>ADCx</b>                  | x=0,1,2  |
| <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 3-14. 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                                  |
| <b>ADCx</b>                  | x=0,1,2   |
| <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 3-15. 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>   |  |
| adc_periph                   | ADC peripheral   |
| ADCx                         | x=0,1,2  |
| <b>Input parameter{in}</b>   |  |
| adc_channel_group            | select the channel group   |
| ADC_REGULAR_CHANNEL          | regular channel group  |
| ADC_INSERTED_CHANNEL         | inserted channel group   |
| ADC_CHANNEL_DISABLE          | disable discontinuous mode of regular and inserted channel   |
| <b>Input parameter{in}</b>   |  |
| length                       | 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 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 3-16. 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  |
| <b>ADCx</b>                  | x=0,1,2   |
| <b>Input parameter{in}</b>   |   |
| <b>adc_channel_group</b>     | select the channel group  |
| <b>ADC_REGULAR_CHA_NNEL</b>  | regular channel group   |
| <b>ADC_INSERTED_CHA_NNEL</b> | inserted channel group  |
| <b>Input parameter{in}</b>   |   |
| <b>length</b>                | the length of the channel, regular channel 1...16, inserted channel 1...4 |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* configure the length of 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 3-17. 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   |
| <b>ADCx</b>                  | x=0,1,2  |
| <b>Input parameter{in}</b>   |  |
| <b>rank</b>                  | the regular group sequence rank, this parameter must be from 0 to 15   |
| <b>Input parameter{in}</b>   |  |
| <b>adc_channel</b>           | the selected ADC channel   |
| <b>ADC_CHANNEL_x</b>         | 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  |
| <b>ADC_SAMPLETIME_1</b>      | 1.5 cycles   |

|                                  |              |
|----------------------------------|--------------|
| <i>POINT5</i>                    |              |
| <i>ADC_SAMPLETIME_7 POINT5</i>   | 7.5 cycles   |
| <i>ADC_SAMPLETIME_1 3POINT5</i>  | 13.5 cycles  |
| <i>ADC_SAMPLETIME_2 8POINT5</i>  | 28.5 cycles  |
| <i>ADC_SAMPLETIME_4 1POINT5</i>  | 41.5 cycles  |
| <i>ADC_SAMPLETIME_5 5POINT5</i>  | 55.5 cycles  |
| <i>ADC_SAMPLETIME_7 1POINT5</i>  | 71.5 cycles  |
| <i>ADC_SAMPLETIME_2 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 3-18. Function `adc_inserted_channel_config`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | adc_inserted_channel_config   |
| <b>Function prototype</b>    | void adc_inserted_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time); |
| <b>Function descriptions</b> | configure ADC inserted channel  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>adc_periph</b>            | ADC peripheral  |
| <b>ADCx</b>                  | x=0,1,2   |
| <b>Input parameter{in}</b>   |   |
| <b>rank</b>                  | the inserted group sequencer rank, this parameter must be from 0 to 3   |
| <b>Input parameter{in}</b>   |   |
| <b>adc_channel</b>           | the selected ADC channel  |
| <b>ADC_CHANNEL_x</b>         | ADC Channelx (x=0...17)(x=16 and x=17 are only for ADC0)  |

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

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 3-19. 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>  | -  |
| Input parameter{in}          |  |
| <b>adc_periph</b>            | ADC peripheral   |
| <b>ADCx</b>                  | x=0,1,2  |
| Input parameter{in}          |  |
| <b>inserted_channel</b>      | insert channel select  |

|                                      |  |
|--------------------------------------|--|
| <code>ADC_INSERTED_CHA_NNEL_x</code> | inserted channel, x=0,1,2,3                            |
| <b>Input parameter{in}</b>           |  |
| <code>offset</code>                  | the offset data, this parameter must be from 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 3-20. Function `adc_external_trigger_source_config`**

|  |   |
|--|---|
| <b>Function name</b>                       | <code>adc_external_trigger_source_config</code>   |
| <b>Function prototype</b>                  | <code>void adc_external_trigger_source_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t external_trigger_source);</code> |
| <b>Function descriptions</b>               | configure ADC external trigger source   |
| <b>Precondition</b>                        | -   |
| <b>The called functions</b>                | -   |
| <b>Input parameter{in}</b>                 |   |
| <code>adc_periph</code>                    | ADC peripheral  |
| <code>ADCx</code>                          | x=0,1,2   |
| <b>Input parameter{in}</b>                 |   |
| <code>adc_channel_group</code>             | select the channel group  |
| <code>ADC_REGULAR_CHA_NNEL</code>          | regular channel group   |
| <code>ADC_INSERTED_CHA_NNEL</code>         | inserted channel group  |
| <b>Input parameter{in}</b>                 |   |
| <code>external_trigger_source</code>       | regular or inserted group trigger source  |
| <code>ADC0_1_EXTTRIG_REGULAR_T0_CH0</code> | TIMER0 CH0 event select for regular channel   |
| <code>ADC0_1_EXTTRIG_REGULAR_T0_CH1</code> | TIMER0 CH1 event select for regular channel   |
| <code>ADC0_1_EXTTRIG_REGULAR_T0_CH2</code> | TIMER0 CH2 event select for regular channel   |
| <code>ADC0_1_EXTTRIG_REGULAR_T1_CH1</code> | TIMER1 CH1 event select for regular channel   |

|   |   |
|---|---|
| <i>GULAR_T1_CH1</i>                         |   |
| <i>ADC0_1_EXTTRIG_RE<br/>GULAR_T2_TRGO</i>  | TIMER2 TRGO event select for regular channel    |
| <i>ADC0_1_EXTTRIG_RE<br/>GULAR_T3_CH3</i>   | TIMER3 CH3 event select for regular channel     |
| <i>ADC0_1_EXTTRIG_RE<br/>GULAR_T7_TRGO</i>  | TIMER7 TRGO event select for regular channel    |
| <i>ADC0_1_EXTTRIG_RE<br/>GULAR EXTI_11</i>  | external interrupt line 11 for regular channel  |
| <i>ADC2_EXTTRIG_REG<br/>ULAR_T2_CH0</i>     | TIMER2 CH0 event select for regular channel     |
| <i>ADC2_EXTTRIG_REG<br/>ULAR_T1_CH2</i>     | TIMER1 CH2 event select for regular channel     |
| <i>ADC2_EXTTRIG_REG<br/>ULAR_T0_CH2</i>     | TIMER0 CH2 event select for regular channel     |
| <i>ADC2_EXTTRIG_REG<br/>ULAR_T7_CH0</i>     | TIMER7 CH0 event select for regular channel     |
| <i>ADC2_EXTTRIG_REG<br/>ULAR_T7_TRGO</i>    | TIMER7 TRGO event select for regular channel    |
| <i>ADC2_EXTTRIG_REG<br/>ULAR_T4_CH0</i>     | TIMER4 CH0 event select for regular channel     |
| <i>ADC2_EXTTRIG_REG<br/>ULAR_T4_CH2</i>     | TIMER4 CH2 event select for regular channel     |
| <i>ADC0_1_2_EXTTRIG_<br/>REGULAR_NONE</i>   | software trigger for regular channel            |
| <i>ADC0_1_EXTTRIG_IN<br/>SERTED_T0_TRGO</i> | TIMER0 TRGO event select for inserted channel   |
| <i>ADC0_1_EXTTRIG_IN<br/>SERTED_T0_CH3</i>  | TIMER0 CH3 event select for inserted channel    |
| <i>ADC0_1_EXTTRIG_IN<br/>SERTED_T1_TRGO</i> | TIMER1 TRGO event select for inserted channel   |
| <i>ADC0_1_EXTTRIG_IN<br/>SERTED_T1_CH0</i>  | TIMER1 CH0 event select for inserted channel    |
| <i>ADC0_1_EXTTRIG_IN<br/>SERTED_T2_CH3</i>  | TIMER2 CH3 event select for inserted channel    |
| <i>ADC0_1_EXTTRIG_IN<br/>SERTED_T3_TRGO</i> | TIMER3 TRGO event select for inserted channel   |
| <i>ADC0_1_EXTTRIG_IN<br/>SERTED_EXTI_15</i> | external interrupt line 15 for inserted channel |
| <i>ADC0_1_EXTTRIG_IN<br/>SERTED_T7_CH3</i>  | TIMER7 CH3 event select for inserted channel    |
| <i>ADC2_EXTTRIG_INSE<br/>RTED_T0_TRGO</i>   | TIMER0 TRGO event select for inserted channel   |

|  |   |
|--|---|
| <i>ADC2_EXTTRIG_INSE<br/>RTED_T0_CH3</i>   | TIMER0 CH3 event select for inserted channel  |
| <i>ADC2_EXTTRIG_INSE<br/>RTED_T3_CH2</i>   | TIMER3 CH2 event select for inserted channel  |
| <i>ADC2_EXTTRIG_INSE<br/>RTED_T7_CH1</i>   | TIMER7 CH1 event select for inserted channel  |
| <i>ADC2_EXTTRIG_INSE<br/>RTED_T7_CH3</i>   | TIMER7 CH3 event select for inserted channel  |
| <i>ADC2_EXTTRIG_INSE<br/>RTED_T4_TRGO</i>  | TIMER4 TRGO event select for inserted channel |
| <i>ADC2_EXTTRIG_INSE<br/>RTED_T4_CH3</i>   | TIMER4 CH3 event select for inserted channel  |
| <i>ADC0_1_2_EXTTRIG_I<br/>NSERTED_NONE</i> | software trigger for inserted channel         |
| <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 3-21. 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  |
| <b>ADCx</b>                  | x=0,1,2   |
| <b>Input parameter{in}</b>   |   |
| <b>adc_channel_group</b>     | select the channel group  |
| <b>ADC_REGULAR_CHANNEL</b>   | regular channel group   |
| <b>ADC_INSERTED_CHANNEL</b>  | inserted channel group  |

|                              |                  |
|------------------------------|------------------|
| NNEL                         |                  |
| <b>Input parameter{in}</b>   |                  |
| <b>newvalue</b>              | control value    |
| <b>ENABLE</b>                | enable function  |
| <b>DISABLE</b>               | disable function |
| <b>Output parameter{out}</b> |                  |
| -                            | -                |
| <b>Return value</b>          |                  |
| -                            | -                |

Example:

```
/* enable 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 3-22. 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  |
| <b>ADCx</b>                      | x=0,1,2   |
| <b>Input parameter{in}</b>       |   |
| <b>adc_channel_group</b>         | select the channel group  |
| <b>ADC_REGULAR_CHA<br/>NNEL</b>  | regular channel group   |
| <b>ADC_INSERTED_CHA<br/>NNEL</b> | 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 3-23. 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>   |  |
| adc_periph                   | ADC peripheral                                       |
| ADCx                         | x=0,1,2  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| 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 3-24. Function adc\_inserted\_data\_read**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | adc_inserted_data_read  |
| <b>Function prototype</b>    | uint16_t adc_inserted_data_read(uint32_t adc_periph, uint8_t inserted_channel); |
| <b>Function descriptions</b> | read ADC inserted group data register   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| adc_periph                   | ADC peripheral  |
| ADCx                         | x=0,1,2   |
| <b>Input parameter{in}</b>   |   |
| inserted_channel             | insert channel select   |
| ADC_INSERTED_CHA_NNEL_x      | inserted Channelx, x=0,1,2,3  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |

| 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 3-25. 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);                |
| Function descriptions | read the last ADC0 and ADC1 conversion result data in sync mode |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |
| -                     | -   |
| Output parameter{out} |   |
| -                     | -   |
| Return value          |   |
| 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 3-26. Function adc\_watchdog\_single\_channel\_enable**

|                       |  |
|-----------------------|--|
| Function name         | adc_watchdog_single_channel_enable   |
| Function prototype    | void adc_watchdog_single_channel_enable(uint32_t adc_periph, uint8_t adc_channel); |
| Function descriptions | configure ADC analog watchdog single channel                                       |
| Precondition          | -  |
| The called functions  | -  |
| Input parameter{in}   |  |
| adc_periph            | ADC peripheral   |

|                              |  |
|------------------------------|--|
| ADCx                         | x=0,1,2  |
| <b>Input parameter{in}</b>   |  |
| adc_channel                  | the selected ADC channel                                 |
| ADC_CHANNEL_x                | 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 3-27. Function adc\_watchdog\_group\_channel\_enable**

|                               |   |
|-------------------------------|---|
| Function name                 | adc_watchdog_group_channel_enable   |
| Function prototype            | void adc_watchdog_group_channel_enable(uint32_t adc_periph, uint8_t adc_channel_group); |
| Function descriptions         | configure ADC analog watchdog group channel   |
| Precondition                  | -   |
| The called functions          | -   |
| <b>Input parameter{in}</b>    |   |
| adc_periph                    | ADC peripheral  |
| ADCx                          | x=0,1,2   |
| <b>Input parameter{in}</b>    |   |
| adc_channel_group             | the channel group use analog watchdog   |
| ADC_REGULAR_CHA_NNEL          | regular channel group   |
| ADC_INSERTED_CHA_NNEL         | inserted channel group  |
| ADC_REGULAR_INSE_RTED_CHANNEL | 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 3-28. 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                                  |
| <b>ADCx</b>                  | x=0,1,2   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable ADC0 analog watchdog */

adc_watchdog_disable(ADC0);
```

### adc\_watchdog\_threshold\_config

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

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

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | adc_watchdog_threshold_config   |
| <b>Function prototype</b>    | void adc_watchdog_threshold_config(uint32_t adc_periph, uint16_t low_threshold, uint16_t high_threshold); |
| <b>Function descriptions</b> | configure ADC analog watchdog threshold   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>adc_periph</b>            | ADC peripheral  |
| <b>ADCx</b>                  | x=0,1,2   |
| <b>Input parameter{in}</b>   |   |
| <b>low_threshold</b>         | analog watchdog low threshold, 0...4095   |
| <b>Input parameter{in}</b>   |   |
| <b>high_threshold</b>        | analog watchdog high threshold, 0...4095  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

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

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

### **adc\_resolution\_config**

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

**Table 3-30. 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</i>                  | x=0,1,2   |
| <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:

```
/* config ADC0 resolution */

adc_resolution_config(ADC0, ADC_RESOLUTION_10B);
```

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

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

**Table 3-31. 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, uint8_t mode, |

|                                     |  |
|-------------------------------------|--|
|                                     | uint16_t shift, uint8_t ratio);  |
| <b>Function descriptions</b>        | configure ADC oversample mode  |
| <b>Precondition</b>                 | -  |
| <b>The called functions</b>         | -  |
| <b>Input parameter{in}</b>          |  |
| <b>adc_periph</b>                   | ADC peripheral   |
| <b>ADCx</b>                         | x=0,1,2  |
| <b>Input parameter{in}</b>          |  |
| <b>mode</b>                         | ADC oversampling mode  |
| <b>ADC_OVERSAMPLING_ALL_CONVERT</b> | all oversampled conversions for a channel are done consecutively after a trigger |
| <b>ADC_OVERSAMPLING_ONE_CONVERT</b> | each oversampled conversion for a channel needs a trigger                        |
| <b>Input parameter{in}</b>          |  |
| <b>shift</b>                        | ADC oversampling shift   |
| <b>ADC_OVERSAMPLING_SHIFT_NONE</b>  | no oversampling shift  |
| <b>ADC_OVERSAMPLING_SHIFT_1B</b>    | 1-bit oversampling shift   |
| <b>ADC_OVERSAMPLING_SHIFT_2B</b>    | 2-bit oversampling shift   |
| <b>ADC_OVERSAMPLING_SHIFT_3B</b>    | 3-bit oversampling shift   |
| <b>ADC_OVERSAMPLING_SHIFT_4B</b>    | 4-bit oversampling shift   |
| <b>ADC_OVERSAMPLING_SHIFT_5B</b>    | 5-bit oversampling shift   |
| <b>ADC_OVERSAMPLING_SHIFT_6B</b>    | 6-bit oversampling shift   |
| <b>ADC_OVERSAMPLING_SHIFT_7B</b>    | 7-bit oversampling shift   |
| <b>ADC_OVERSAMPLING_SHIFT_8B</b>    | 8-bit oversampling shift   |
| <b>Input parameter{in}</b>          |  |
| <b>ratio</b>                        | ADC oversampling ratio   |
| <b>ADC_OVERSAMPLING_RATIO_MUL2</b>  | oversampling ratio multiple 2  |
| <b>ADC_OVERSAMPLING_RATIO_MUL4</b>  | oversampling ratio multiple 4  |
| <b>ADC_OVERSAMPLING_RATIO_MUL8</b>  | oversampling ratio multiple 8  |
| <b>ADC_OVERSAMPLING_RATIO_MUL16</b> | oversampling ratio multiple 16   |

|  |                                 |
|--|---------------------------------|
| <code>ADC_OVERSAMPLING_RATIO_MUL32</code>  | oversampling ratio multiple 32  |
| <code>ADC_OVERSAMPLING_RATIO_MUL64</code>  | oversampling ratio multiple 64  |
| <code>ADC_OVERSAMPLING_RATIO_MUL128</code> | oversampling ratio multiple 128 |
| <code>ADC_OVERSAMPLING_RATIO_MUL256</code> | oversampling ratio multiple 256 |
| <b>Output parameter{out}</b>               |                                 |
| -  | -                               |
| <b>Return value</b>                        |                                 |
| -  | -                               |

Example:

```
/* config ADC0 oversample mode */

adc_oversample_mode_config(ADC0, ADC_OVERSAMPLING_ALL_CONVERT,
                           ADC_OVERSAMPLING_SHIFT_2B, ADC_OVERSAMPLING_RATIO_MUL4);
```

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

The description of `adc_oversample_mode_enable` is shown as below:

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

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | <code>adc_oversample_mode_enable</code>                            |
| <b>Function prototype</b>    | <code>void adc_oversample_mode_enable(uint32_t adc_periph);</code> |
| <b>Function descriptions</b> | enable ADC oversample mode   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <code>adc_periph</code>      | ADC peripheral   |
| <code>ADCx</code>            | x=0,1,2  |
| <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 3-33. 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>   |  |
| adc_periph                   | ADC peripheral   |
| ADCx                         | x=0,1,2  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable ADC0 oversample mode */
adc_oversample_mode_disable(ADC0);
```

### adc\_flag\_get

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

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

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | adc_flag_get  |
| <b>Function prototype</b>    | FlagStatus adc_flag_get(uint32_t adc_periph , uint32_t adc_flag); |
| <b>Function descriptions</b> | get the ADC flag  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| adc_periph                   | ADC peripheral  |
| ADCx                         | x=0,1,2   |
| <b>Input parameter{in}</b>   |   |
| adc_flag                     | the adc flag  |
| ADC_FLAG_WDE                 | analog watchdog event flag  |
| ADC_FLAG_EOC                 | end of group conversion flag                                      |
| ADC_FLAG_EOIC                | end of inserted group conversion flag                             |
| ADC_FLAG_STIC                | start flag of inserted channel group                              |
| ADC_FLAG_STRC                | start flag of regular channel group                               |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| FlagStatus                   | 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   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| adc_periph                   | ADC peripheral   |
| ADCx                         | x=0,1,2  |
| <b>Input parameter{in}</b>   |  |
| adc_flag                     | the adc flag   |
| ADC_FLAG_WDE                 | analog watchdog event flag                                   |
| ADC_FLAG_EOC                 | end of group conversion flag                                 |
| ADC_FLAG_EOIC                | end of inserted group conversion flag                        |
| ADC_FLAG_STIC                | start flag of inserted channel group                         |
| ADC_FLAG_STRC                | start flag of regular channel group                          |
| <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\_interrupt\_enable**

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

**Table 3-36. 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>  | -   |

| Input parameter{in}   |  |
|-----------------------|--|
| <b>adc_periph</b>     | ADC peripheral                             |
| <b>ADCx</b>           | x=0,1,2                                    |
| Input parameter{in}   |  |
| <b>adc_interrupt</b>  | the adc interrupt                          |
| <b>ADC_INT_WDE</b>    | analog watchdog interrupt                  |
| <b>ADC_INT_EOC</b>    | end of group conversion interrupt          |
| <b>ADC_INT_EOIC</b>   | end of inserted group conversion interrupt |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

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 3-37. 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>  | -   |
| Input parameter{in}          |   |
| <b>adc_periph</b>            | ADC peripheral  |
| <b>ADCx</b>                  | x=0,1,2   |
| Input parameter{in}          |   |
| <b>adc_interrupt</b>         | the adc interrupt   |
| <b>ADC_INT_WDE</b>           | analog watchdog interrupt   |
| <b>ADC_INT_EOC</b>           | end of group conversion interrupt   |
| <b>ADC_INT_EOIC</b>          | end of inserted group conversion interrupt                                |
| Output parameter{out}        |   |
| -                            | -   |
| Return value                 |   |
| -                            | -   |

Example:

```
/* disable ADC0 interrupt */
adc_interrupt_disable(ADC0, ADC_INT_WDE);
```

### **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   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| adc_periph                   | ADC peripheral  |
| ADCx                         | x=0,1,2   |
| <b>Input parameter{in}</b>   |   |
| adc_interrupt                | the adc interrupt   |
| ADC_INT_WDE                  | analog watchdog interrupt   |
| ADC_INT_EOC                  | end of group conversion interrupt   |
| ADC_INT_EOIC                 | end of inserted group conversion interrupt                                      |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| FlagStatus                   | 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   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| adc_periph                   | ADC peripheral  |
| ADCx                         | x=0,1,2   |
| <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:

```
/* clear the ADC0 analog watchdog interrupt bits */
adc_interrupt_flag_clear(ADC0, ADC_INT_WDE);
```

### 3.3. BKP

The Backup registers are located in the Backup domain that remains powered-on by V<sub>BAT</sub> even if V<sub>DD</sub> 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 3-40. BKP Registers**

| Registers           | Descriptions                       |
|---------------------|------------------------------------|
| BKP_DATAx (x=0..41) | BKP data register x                |
| BKP_OCTL            | RTC signal output control register |
| BKP_TPCTL0          | tamper pin control register0       |
| BKP_TPCS            | tamper control and status register |
| BKP_TPCTL1          | tamper pin control register1       |

#### 3.3.2. Descriptions of Peripheral functions

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

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

| Function name  | Function description    |
|----------------|-------------------------|
| bkp_deinit     | reset BKP registers     |
| bkp_data_write | write BKP data register |

| Function name                       | Function description                      |
|-------------------------------------|---|
| bkp_data_read                       | read BKP 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                         |
| bkp_RTC_clock_output_select         | RTC clock output selection                |
| bkp_RTC_clock_calibration_direction | RTC clock calibration direction           |
| bkp_RTC_calibration_value_set       | set RTC clock calibration value           |
| bkp_tamper_detection_enable         | enable tamper pin detection               |
| bkp_tamper_detection_disable        | disable tamper pin detection              |
| bkp_tamper_active_level_set         | set tamper pin active level               |
| bkp_waveform_detect_config          | waveform detect configure                 |
| bkp_flag_get                        | get BKP flag                              |
| bkp_flag_clear                      | clear BKP flag                            |
| bkp_tamper_interrupt_enable         | enable tamper interrupt                   |
| bkp_tamper_interrupt_disable        | disable tamper interrupt                  |
| bkp_interrupt_flag_get              | get BKP interrupt flag                    |
| bkp_interrupt_flag_clear            | clear BKP interrupt flag                  |

### Enum bkp\_data\_register\_enum

Table 3-42. Enum bkp\_data\_register\_enum

| Member name | Function description |
|-------------|----------------------|
| BKP_DATA_0  | bkp data register 0  |
| BKP_DATA_1  | bkp data register 1  |
| BKP_DATA_2  | bkp data register 2  |
| BKP_DATA_3  | bkp data register 3  |
| BKP_DATA_4  | bkp data register 4  |
| BKP_DATA_5  | bkp data register 5  |
| BKP_DATA_6  | bkp data register 6  |
| BKP_DATA_7  | bkp data register 7  |
| BKP_DATA_8  | bkp data register 8  |
| BKP_DATA_9  | bkp data register 9  |
| BKP_DATA_10 | bkp data register 10 |
| BKP_DATA_11 | bkp data register 11 |
| BKP_DATA_12 | bkp data register 12 |
| BKP_DATA_13 | bkp data register 13 |
| BKP_DATA_14 | bkp data register 14 |
| BKP_DATA_15 | bkp data register 15 |
| BKP_DATA_16 | bkp data register 16 |
| BKP_DATA_17 | bkp data register 17 |

| Member name | Function description |
|-------------|----------------------|
| BKP_DATA_18 | bkp data register 18 |
| BKP_DATA_19 | bkp data register 19 |
| BKP_DATA_20 | bkp data register 20 |
| BKP_DATA_21 | bkp data register 21 |
| BKP_DATA_22 | bkp data register 22 |
| BKP_DATA_23 | bkp data register 23 |
| BKP_DATA_24 | bkp data register 24 |
| BKP_DATA_25 | bkp data register 25 |
| BKP_DATA_26 | bkp data register 26 |
| BKP_DATA_27 | bkp data register 27 |
| BKP_DATA_28 | bkp data register 28 |
| BKP_DATA_29 | bkp data register 29 |
| BKP_DATA_30 | bkp data register 30 |
| BKP_DATA_31 | bkp data register 31 |
| BKP_DATA_32 | bkp data register 32 |
| BKP_DATA_33 | bkp data register 33 |
| BKP_DATA_34 | bkp data register 34 |
| BKP_DATA_35 | bkp data register 35 |
| BKP_DATA_36 | bkp data register 36 |
| BKP_DATA_37 | bkp data register 37 |
| BKP_DATA_38 | bkp data register 38 |
| BKP_DATA_39 | bkp data register 39 |
| BKP_DATA_40 | bkp data register 40 |
| BKP_DATA_41 | bkp data register 41 |

### Enum bkp\_tamper\_enum

Table 3-43. Enum bkp\_tamper\_enum

| Member name | Function description |
|-------------|----------------------|
| TAMPER_0    | BKP tamper0          |
| TAMPER_1    | BKP tamper1          |

### 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 BKP registers                          |
| Precondition          | -  |
| The called functions  | rcu_bkp_reset_enable / rcu_bkp_reset_disable |
| Input parameter{in}   |  |

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

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**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | bkp_data_write  |
| <b>Function prototype</b>    | void bkp_data_write(bkp_data_register_enum register_number, uint16_t data); |
| <b>Function descriptions</b> | write BKP data register   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>register_number</b>       | refer to <a href="#">Table 3-42. Enum bkp_data_register_enum</a>            |
| <i>BKP_DATA_x(x = 0..41)</i> | bkp data register number x  |
| <b>Input parameter{in}</b>   |   |
| <b>data</b>                  | the data to be write in BKP data register                                   |
| <i>0x0000-0xFFFF</i>         | 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 BKP data register   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>register_number</b>       | refer to <a href="#">Table 3-42. Enum bkp_data_register enum</a> |
| <b>BKP_DATA_x(x = 0..41)</b> | bkp data register number x                                       |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>uint16_t</b>              | data of BKP data register(0x0000-0xFFFF)                         |

Example:

```
/* read BKP data register0 */

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                                |
| <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_SECOND_PULSE</i> | RTC second pulse is selected as the RTC output   |
| <b>Output parameter{out}</b>   |  |
| -                              | -  |
| <b>Return value</b>            |  |
| -                              | -  |

Example:

```
/* select RTC alarm pulse as the RTC 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> | RTC clock output selection                           |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>clocksel</b>              | RTC clock output selection                           |
| <i>RTC_CLOCK_DIV64</i>       | RTC clock div 64                                     |
| <i>RTC_CLOCK_DIV1</i>        | RTC clock  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* RTC clock output selection */

bkp_rtc_clock_output_select(RTC_CLOCK_DIV1);
```

### **bkp\_rtc\_clock\_calibration\_direction**

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

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

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | bkp_rtc_clock_calibration_direction                           |
| <b>Function prototype</b>    | void bkp_rtc_clock_calibration_direction(uint16_t direction); |
| <b>Function descriptions</b> | RTC clock calibration direction                               |
| <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 speed up */

bkp_rtc_clock_calibration_direction(RTC_CLOCK_SPEED_UP);
```

### 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                        |
| 0x00 - 0x7F                  | 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(bkp_tamper_enum tamperx); |
| <b>Function descriptions</b> | enable tamper pin detection                                |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>tamperx</b>               | refer to <a href="#">Table 3-43. Enum bkp_tamper_enum</a>  |
| TAMPER_0                     | BKP tamper0  |
| TAMPER_1                     | BKP tamper1  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable tamper0 pin detection */

bkp_tamper_detection_enable(TAMPER_0);
```

### 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(bkp_tamper_enum tamperx); |
| <b>Function descriptions</b> | disable tamper pin detection                                |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>tamperx</b>               | refer to <a href="#">Table 3-43. Enum bkp_tamper_enum</a>   |
| <i>TAMPER_0</i>              | BKP tamper0   |
| <i>TAMPER_1</i>              | BKP tamper1   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable tamper0 pin detection */
bkp_tamper_detection_disable(TAMPER_0);
```

### 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(bkp_tamper_enum tamperx, 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>tamperx</b>                | refer to <a href="#">Table 3-43. Enum bkp_tamper_enum</a>                  |
| <i>TAMPER_0</i>               | BKP tamper0  |
| <i>TAMPER_1</i>               | BKP tamper1  |
| <b>Input parameter{in}</b>    |  |
| <b>level</b>                  | tamper pin active level  |
| <i>TAMPER_PIN_ACTIVE_HIGH</i> | the tamper pin is active high  |
| <i>TAMPER_PIN_ACTIVE_LOW</i>  | the tamper pin is active low   |

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

Example:

```
/* set tamper0 pin active level high */
bkp_tamper_active_level_set(TAMPER_0, TAMPER_PIN_ACTIVE_HIGH);
```

### bkp\_waveform\_detect\_config

The description of bkp\_waveform\_detect\_config is shown as below:

**Table 3-58. Function bkp\_waveform\_detect\_config**

| Function name             | bkp_waveform_detect_config  |
|---------------------------|---|
| Function prototype        | void bkp_waveform_detect_config (uint16_t waveform_detect_mode,<br>ControlStatus newvalue); |
| Function descriptions     | waveform detect configure   |
| Precondition              | -   |
| The called functions      | -   |
| Input parameter{in}       |   |
| waveform_detect_mo<br>de  | waveform_detect_mode  |
| BKP_WAVEFORM_DE<br>TECT_1 | the first waveform detection  |
| BKP_WAVEFORM_DE<br>TECT_2 | the second waveform detection   |
| Input parameter{in}       |   |
| newvalue                  | control value   |
| ENABLE                    | enable function   |
| DISABLE                   | disable function  |
| Output parameter{out}     |   |
| -                         | -   |
| Return value              |   |
| -                         | -   |

Example:

```
/* enable the first waveform detect */
bkp_waveform_detect_config(BKP_WAVEFORM_DETECT_1, ENABLE);
```

### bkp\_flag\_get

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

**Table 3-59. Function bkp\_flag\_get**

|                                    |   |
|------------------------------------|---|
| <b>Function name</b>               | bkp_flag_get                            |
| <b>Function prototype</b>          | FlagStatus bkp_flag_get(uint16_t flag); |
| <b>Function descriptions</b>       | get BKP flag                            |
| <b>Precondition</b>                | -                                       |
| <b>The called functions</b>        | -                                       |
| <b>Input parameter{in}</b>         |   |
| <b>flag</b>                        | BKP flag                                |
| <i>BKP_FLAG_TAMPER0</i>            | tamper0 event flag                      |
| <i>BKP_FLAG_TAMPER1_WAVEDETECT</i> | tamper1/waveform detect event flag      |
| <b>Output parameter{out}</b>       |   |
| -                                  | -                                       |
| <b>Return value</b>                |   |
| <b>FlagStatus</b>                  | SET or RESET                            |

Example:

```
/* get BKP tamper0 event flag */

FlagStatus status;

status = bkp_flag_get(BKP_FLAG_TAMPER0);
```

### bkp\_flag\_clear

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

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

|                                    |                                     |
|------------------------------------|-------------------------------------|
| <b>Function name</b>               | bkp_flag_clear                      |
| <b>Function prototype</b>          | void bkp_flag_clear(uint16_t flag); |
| <b>Function descriptions</b>       | clear BKP flag                      |
| <b>Precondition</b>                | -                                   |
| <b>The called functions</b>        | -                                   |
| <b>Input parameter{in}</b>         |                                     |
| <b>flag</b>                        | BKP flag                            |
| <i>BKP_FLAG_TAMPER0</i>            | tamper0 event flag                  |
| <i>BKP_FLAG_TAMPER1_WAVEDETECT</i> | tamper1/waveform detect event flag  |
| <b>Output parameter{out}</b>       |                                     |
| -                                  | -                                   |
| <b>Return value</b>                |                                     |
| -                                  | -                                   |

Example:

```
/* clear BKP tamper0 event flag */
```

```
bkp_flag_clear(BKP_FLAG_TAMPER0);
```

### **bkp\_tamper\_interrupt\_enable**

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

**Table 3-61. Function bkp\_tamper\_interrupt\_enable**

|                                   |   |
|-----------------------------------|---|
| <b>Function name</b>              | bkp_tamper_interrupt_enable                               |
| <b>Function prototype</b>         | void bkp_tamper_interrupt_enable(uint16_t bkp_interrupt); |
| <b>Function descriptions</b>      | enable tamper interrupt                                   |
| <b>Precondition</b>               | -   |
| <b>The called functions</b>       | -   |
| <b>Input parameter{in}</b>        |   |
| <b>bkp_interrupt</b>              | the BKP interrupt   |
| <i>BKP_INT_TAMPER0</i>            | BKP tamper0 interrupt                                     |
| <i>BKP_INT_TAMPER1_WAVEDETECT</i> | BKP tamper1/waveform detect interrupt                     |
| <b>Output parameter{out}</b>      |   |
| -                                 | -   |
| <b>Return value</b>               |   |
| -                                 | -   |

Example:

```
/* enable tamper0 interrupt */

bkp_tamper_interrupt_enable(BKP_INT_TAMPER0);
```

### **bkp\_tamper\_interrupt\_disable**

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

**Table 3-62. Function bkp\_tamper\_interrupt\_disable**

|                                   |  |
|-----------------------------------|--|
| <b>Function name</b>              | bkp_tamper_interrupt_disable                               |
| <b>Function prototype</b>         | void bkp_tamper_interrupt_disable(uint16_t bkp_interrupt); |
| <b>Function descriptions</b>      | disable tamper interrupt                                   |
| <b>Precondition</b>               | -  |
| <b>The called functions</b>       | -  |
| <b>Input parameter{in}</b>        |  |
| <b>bkp_interrupt</b>              | the BKP interrupt  |
| <i>BKP_INT_TAMPER0</i>            | BKP tamper0 interrupt                                      |
| <i>BKP_INT_TAMPER1_WAVEDETECT</i> | BKP tamper1/waveform detect interrupt                      |
| <b>Output parameter{out}</b>      |  |
| -                                 | -  |
| <b>Return value</b>               |  |

Example:

```
/* disable tamper0 interrupt */
bkp_tamper_interrupt_disable(BKP_INT_TAMPER0);
```

### **bkp\_interrupt\_flag\_get**

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

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

|   |   |
|---|---|
| <b>Function name</b>                    | bkp_interrupt_flag_get                            |
| <b>Function prototype</b>               | FlagStatus bkp_interrupt_flag_get(uint16_t flag); |
| <b>Function descriptions</b>            | get BKP interrupt flag                            |
| <b>Precondition</b>                     | -   |
| <b>The called functions</b>             | -   |
| <b>Input parameter{in}</b>              |   |
| <b>flag</b>                             | BKP interrupt flag                                |
| <i>BKP_INT_FLAG_TAMP_ER0</i>            | tamper0 interrupt flag                            |
| <i>BKP_INT_FLAG_TAMP_ER1_WAVEDETECT</i> | tamper1/waveform detect interrupt flag            |
| <b>Output parameter{out}</b>            |   |
| -                                       | -   |
| <b>Return value</b>                     |   |
| <b>FlagStatus</b>                       | SET or RESET                                      |

Example:

```
/* get BKP tamper0 interrupt flag */
FlagStatus tamper0_status;
tamper0_status = bkp_interrupt_flag_get(BKP_INT_FLAG_TAMPER0);
```

### **bkp\_interrupt\_flag\_clear**

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

**Table 3-64. Function bkp\_interrupt\_flag\_clear**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | bkp_interrupt_flag_clear                      |
| <b>Function prototype</b>    | void bkp_interrupt_flag_clear(uint16_t flag); |
| <b>Function descriptions</b> | clear BKP interrupt flag                      |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |

|   |  |
|---|--|
| <b>flag</b>                                 | BKP interrupt flag                     |
| <i>BKP_INT_FLAG_TAMP<br/>ER0</i>            | tamper0 interrupt flag                 |
| <i>BKP_INT_FLAG_TAMP<br/>ER1_WAVEDETECT</i> | tamper1/waveform detect interrupt flag |
| <b>Output parameter{out}</b>                |  |
| -   | -                                      |
| <b>Return value</b>                         |  |
| -   | -                                      |

Example:

```
/* clear BKP tamper0 interrupt flag */
bkp_interrupt_flag_clear(BKP_INT_FLAG_TAMPER0);
```

## 3.4. CAN

CAN bus (for Controller Area Network) is a bus standard designed to allow microcontrollers and devices to communicate with each other without a host computer. The CAN registers are listed in chapter [3.4.1](#), the CAN firmware functions are introduced in chapter [3.4.2](#).

### 3.4.1. Descriptions of Peripheral registers

CAN registers are listed in the table shown as below:

**Table 3-65. CAN Registers**

| Registers           | Descriptions                             |
|---------------------|--|
| CAN_CTL             | Control register                         |
| CAN_STAT            | Status register                          |
| CAN_TSTAT           | Transmit status register                 |
| CAN_RFIFO0          | Receive message FIFO0 register           |
| CAN_RFIFO1          | Receive message FIFO1 register           |
| CAN_INTEN           | Interrupt enable register                |
| CAN_ERR             | Error register                           |
| CAN_BT              | Bit timing register                      |
| CAN_TMIx            | Transmit mailbox identifier register     |
| CAN_TMPx            | Transmit mailbox property register       |
| CAN_TMDATA0x        | Transmit mailbox data0 register          |
| CAN_TMDATA1x        | Transmit mailbox data1 register          |
| CAN_RFIFOMIx        | Receive FIFO mailbox identifier register |
| CAN_RFIFOMPx        | Receive FIFO mailbox property register   |
| CAN_RFIFODAT<br>A0x | Receive FIFO mailbox data0 register      |

| Registers            | Descriptions                        |
|----------------------|-------------------------------------|
| CAN_RFIFOMDAT<br>A1x | Receive FIFO mailbox data1 register |
| CAN_FCTL             | Filter control register             |
| CAN_FMCFG            | Filter mode configuration register  |
| CAN_FSCFG            | Filter scale configuration register |
| CAN_FAFIFO           | Filter associated FIFO register     |
| CAN_FW               | Filter working register             |
| CAN_FxDATAY          | Filter x data y register            |

### 3.4.2. Descriptions of Peripheral functions

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

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

| Function name                  | Function description              |
|--------------------------------|-----------------------------------|
| can_deinit                     | deinitialize CAN                  |
| can_struct_para_init           | initialize CAN struct             |
| can_init                       | initialize CAN                    |
| can_filter_init                | CAN filter initialize             |
| can1_filter_start_bank         | set CAN1 filter start bank number |
| can_debug_freeze_enable        | CAN debug freeze enable           |
| can_debug_freeze_disable       | CAN debug freeze disable          |
| can_time_trigger_mode_enable   | CAN time trigger mode enable      |
| can_time_trigger_mode_disable  | CAN time trigger mode disable     |
| can_message_transmit           | transmit CAN message              |
| can_transmit_states            | get CAN transmit state            |
| can_transmission_stop          | stop CAN transmission             |
| can_message_receive            | CAN receive message               |
| can_fifo_release               | CAN release fifo                  |
| can_receive_message_length_get | CAN receive message length        |
| can_working_mode_set           | CAN working mode                  |
| can_wakeup                     | CAN wakeup from sleep mode        |
| can_error_get                  | CAN get error                     |
| can_receive_error_number_get   | get CAN receive error number      |
| can_transmit_error_number_get  | get CAN transmit error number     |
| can_interrupt_enable           | CAN interrupt enable              |
| can_interrupt_disable          | CAN interrupt disable             |
| can_flag_get                   | CAN get flag state                |
| can_flag_clear                 | CAN clear flag state              |
| can_interrupt_flag_get         | CAN get interrupt flag state      |
| can_interrupt_flag_clear       | CAN clear interrupt flag state    |

### Structure can\_parameter\_struct

**Table 3-67. can\_parameter\_struct**

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

### Structure can\_trasnmit\_message\_struct

**Table 3-68. can\_trasnmit\_message\_struct**

| Member name | Function description                         |
|-------------|--|
| tx_sfid     | standard format frame identifier             |
| tx_efid     | extended format frame identifier             |
| tx_ff       | format of frame, standard or extended format |
| tx_ft       | type of frame, data or remote                |
| tx_dlen     | data length                                  |
| tx_data[8]  | transmit data                                |

### Structure can\_receive\_message\_struct

**Table 3-69. can\_receive\_message\_struct**

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

### Structure can\_filter\_parameter\_struct

**Table 3-70. can\_filter\_parameter\_struct**

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

### Enum can\_flag\_enum

**Table 3-71. can\_flag\_enum**

| Member name       | Function description                                 |
|-------------------|--|
| CAN_FLAG_RXL      | RX level   |
| CAN_FLAG_LASTRX   | last sample value of RX pin                          |
| CAN_FLAG_RS       | receiving state                                      |
| CAN_FLAG_TS       | transmitting state                                   |
| CAN_FLAG_SLPIF    | status change flag of entering sleep working mode    |
| CAN_FLAG_WUIF     | status change flag of wakeup from sleep working mode |
| CAN_FLAG_ERRIF    | error flag   |
| CAN_FLAG_SLPWS    | sleep working state                                  |
| CAN_FLAG_IWS      | initial working state                                |
| CAN_FLAG_TMLS2    | transmit mailbox 2 last sending in Tx FIFO           |
| CAN_FLAG_TMLS1    | transmit mailbox 1 last sending in Tx FIFO           |
| CAN_FLAG_TMLS0    | transmit mailbox 0 last sending in Tx FIFO           |
| CAN_FLAG_TME2     | transmit mailbox 2 empty                             |
| CAN_FLAG_TME1     | transmit mailbox 1 empty                             |
| CAN_FLAG_TME0     | transmit mailbox 0 empty                             |
| CAN_FLAG_MTE2     | mailbox 2 transmit error                             |
| CAN_FLAG_MTE1     | mailbox 1 transmit error                             |
| CAN_FLAG_MTE0     | mailbox 0 transmit error                             |
| CAN_FLAG_MAL2     | mailbox 2 arbitration lost                           |
| CAN_FLAG_MAL1     | mailbox 1 arbitration lost                           |
| CAN_FLAG_MAL0     | mailbox 0 arbitration lost                           |
| CAN_FLAG_MTFNERR2 | mailbox 2 transmit finished with no error            |
| CAN_FLAG_MTFNERR1 | mailbox 1 transmit finished with no error            |
| CAN_FLAG_MTFNERR0 | mailbox 0 transmit finished with no error            |
| CAN_FLAG_MTF2     | mailbox 2 transmit finished                          |

| Member name    | Function description        |
|----------------|-----------------------------|
| CAN_FLAG_MTF1  | mailbox 1 transmit finished |
| CAN_FLAG_MTF0  | mailbox 0 transmit finished |
| CAN_FLAG_RFO0  | receive FIFO0 overfull      |
| CAN_FLAG_RFF0  | receive FIFO0 full          |
| CAN_FLAG_RFO1  | receive FIFO1 overfull      |
| CAN_FLAG_RFF1  | receive FIFO1 full          |
| CAN_FLAG_BOERR | bus-off error               |
| CAN_FLAG_PERR  | passive error               |
| CAN_FLAG_WERR  | warning error               |

### Enum can\_interrupt\_flag\_enum

Table 3-72. can\_interrupt\_flag\_enum

| Member name        | Function description   |
|--------------------|--|
| CAN_INT_FLAG_SLPIF | status change interrupt flag of sleep working mode entering    |
| CAN_INT_FLAG_WUIF  | status change interrupt flag of wakeup from sleep working mode |
| CAN_INT_FLAG_ERRIF | error interrupt flag   |
| CAN_INT_FLAG_MTF2  | mailbox 2 transmit finished interrupt flag                     |
| CAN_INT_FLAG_MTF1  | mailbox 1 transmit finished interrupt flag                     |
| CAN_INT_FLAG_MTF0  | mailbox 0 transmit finished interrupt flag                     |
| CAN_INT_FLAG_RFO0  | receive FIFO0 overfull interrupt flag                          |
| CAN_INT_FLAG_RFF0  | receive FIFO0 full interrupt flag                              |
| CAN_INT_FLAG_RFL0  | receive FIFO0 not empty interrupt flag                         |
| CAN_INT_FLAG_RFO1  | receive FIFO1 overfull interrupt flag                          |
| CAN_INT_FLAG_RFF1  | receive FIFO1 full interrupt flag                              |
| CAN_INT_FLAG_RFL1  | receive FIFO1 not empty interrupt flag                         |
| CAN_INT_FLAG_ERRN  | error number interrupt flag                                    |
| CAN_INT_FLAG_BOERR | bus-off error interrupt flag                                   |
| CAN_INT_FLAG_PERR  | passive error interrupt flag                                   |
| CAN_INT_FLAG_WERR  | warning error interrupt flag                                   |

### Enum can\_error\_enum

Table 3-73. can\_error\_enum

| Member name             | Function description |
|-------------------------|----------------------|
| CAN_ERROR_NONE          | no error             |
| CAN_ERROR_FILL          | fill error           |
| CAN_ERROR_FORMATE       | format error         |
| CAN_ERROR_ACK           | ACK error            |
| CAN_ERROR_BITRECESSIVE  | bit recessive error  |
| CAN_ERROR_BITDOMINANTER | bit dominant error   |
| CAN_ERROR_CRC           | CRC error            |

| Member name           | Function description |
|-----------------------|----------------------|
| CAN_ERROR_SOFTWARECFG | software configure   |

### Enum can\_transmit\_state\_enum

Table 3-74. can\_transmit\_state\_enum

| Member name            | Function description                |
|------------------------|-------------------------------------|
| CAN_TRANSMIT_FAILED    | CAN transmitted failure             |
| CAN_TRANSMIT_OK        | CAN transmitted success             |
| CAN_TRANSMIT_PENDING   | CAN transmitted pending             |
| CAN_TRANSMIT_NOMAILBOX | no empty mailbox to be used for CAN |

### Enum can\_struct\_type\_enum

Table 3-75. can\_struct\_type\_enum

| Member name           | Function description            |
|-----------------------|---------------------------------|
| CAN_INIT_STRUCT       | CAN initilaze parameters struct |
| CAN_FILTER_STRUCT     | CAN filter parameters struct    |
| CAN_TX_MESSAGE_STRUCT | CAN transmit message struct     |
| CAN_RX_MESSAGE_STRUCT | CAN receive message struct      |

### can\_deinit

The description of can\_deinit is shown as below:

Table 3-76. Function can\_deinit

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | can_deinit  |
| <b>Function prototype</b>    | void can_deinit(uint32_t can_periph);             |
| <b>Function descriptions</b> | deinitialize CAN                                  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | rcu_periph_reset_enable/ rcu_periph_reset_disable |
| <b>Input parameter{in}</b>   |   |
| can_periph                   | CAN peripheral                                    |
| CANx(x=0, 1)                 | CAN peripheral selection                          |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* CAN0 deinitialize */

can_deinit(CAN0);
```

### can\_struct\_para\_init

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

**Table 3-77. Function can\_struct\_para\_init**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | can_struct_para_init   |
| <b>Function prototype</b>    | void can_struct_para_init(can_struct_type_enum type, void* p_struct) |
| <b>Function descriptions</b> | initialize CAN parameter struct with a default value                 |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>type</b>                  | Struct refer to <a href="#">Table 3-75. can_struct_type_enum</a>     |
| CAN_INIT_STRUCT              | CAN initilize parameters struct                                      |
| CAN_FILTER_STRUCT            | CAN filter parameters struct   |
| CAN_TX_MESSAGE_STRUCT        | CAN transmit message struct  |
| CAN_RX_MESSAGE_STRUCT        | CAN receive message struct   |
| <b>Output parameter{out}</b> |  |
| <b>p_struct</b>              | the pointer of the specific struct                                   |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* CAN parameter struct initialize */

can_parameter_struct    can_parameter;

can_struct_para_init(CAN_INIT_STRUCT, &can_parameter);
```

### can\_init

The description of can\_init is shown as below:

**Table 3-78. Function can\_init**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | can_init   |
| <b>Function prototype</b>    | ErrStatus can_init(uint32_t can_periph, can_parameter_struct* can_parameter_init); |
| <b>Function descriptions</b> | initialize CAN   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>can_periph</b>            | CAN peripheral   |
| CANx(x=0,1)                  | CAN peripheral selection   |

| Input parameter{in}   |  |
|-----------------------|--|
| can_parameter_init    | CAN parameter initialization stuct, the structure members can refer to members of the structure <a href="#">Table 3-67. can_parameter_struct</a> |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| ErrStatus             | SUCCESS / ERROR  |

Example:

```

/* CAN0 initialize */

can_parameter_struct can_parameter;

can_parameter.time_triggered = DISABLE;

can_parameter.auto_bus_off_recovery = DISABLE;

can_parameter.auto_wake_up = DISABLE;

can_parameter.auto_retrans = ENABLE;

can_parameter.rec_fifo_overwrite = DISABLE;

can_parameter.trans_fifo_order = DISABLE;

can_parameter.working_mode = CAN_NORMAL_MODE;

can_parameter.resync_jump_width = CAN_BT_SJW_1TQ;

can_parameter.time_segment_1 = CAN_BT_BS1_8TQ;

can_parameter.time_segment_2 = CAN_BT_BS2_3TQ;

can_parameter.prescaler = 5;

can_init(CAN0, &can_parameter);
  
```

### can\_filter\_init

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

**Table 3-79. Function can\_filter\_init**

| Function name             | can_filter_init  |
|---------------------------|--|
| Function prototype        | void can_filter_init(can_filter_parameter_struct* can_filter_parameter_init);  |
| Function descriptions     | initialize CAN filter  |
| Precondition              | -  |
| The called functions      | -  |
| Input parameter{in}       |  |
| can_filter_parameter_init | CAN filter initialization stuct, the structure members can refer to members of the structure <a href="#">Table 3-70. can_filter_parameter_struct</a> |
| Output parameter{out}     |  |

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

Example:

```

/* initialize CAN filter */

can_filter_parameter_struct can_filter;

can_filter.filter_number=0;

can_filter.filter_mode = CAN_FILTERMODE_MASK;

can_filter.filter_bits = CAN_FILTERBITS_32BIT;

can_filter.filter_list_high = 0x3000;

can_filter.filter_list_low = 0x0000;

can_filter.filter_mask_high = 0x3000;

can_filter.filter_mask_low = 0x0000;

can_filter.filter_fifo_number = CAN_FIFO0;

can_filter.filter_enable = ENABLE;

can_filter_init(&can_filter);
  
```

### can1\_filter\_start\_bank

The description of can1\_filter\_start\_bank is shown as below:

**Table 3-80. Function can1\_filter\_start\_bank**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | can1_filter_start_bank                           |
| <b>Function prototype</b>    | void can1_filter_start_bank(uint8_t start_bank); |
| <b>Function descriptions</b> | set CAN1 fliter start bank number                |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>start_bank</b>            | CAN1 start bank number                           |
| 1..27                        | start number                                     |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```

/* set CAN1 fliter start bank number 15 */
  
```

```
can1_filter_start_bank(15);
```

### **can\_debug\_freeze\_enable**

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

**Table 3-81. Function can\_debug\_freeze\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | can_debug_freeze_enable                            |
| <b>Function prototype</b>    | void can_debug_freeze_enable(uint32_t can_periph); |
| <b>Function descriptions</b> | enable CAN debug freeze                            |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | dbg_periph_enable                                  |
| <b>Input parameter{in}</b>   |  |
| can_periph                   | CAN peripheral                                     |
| CANx(x=0, 1)                 | CAN peripheral selection                           |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable CAN0 debug freeze */
can_debug_freeze_enable(CAN0);
```

### **can\_debug\_freeze\_disable**

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

**Table 3-82. Function can\_debug\_freeze\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | can_debug_freeze_disable                            |
| <b>Function prototype</b>    | void can_debug_freeze_disable(uint32_t can_periph); |
| <b>Function descriptions</b> | disable CAN debug freeze                            |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | dbg_periph_disable                                  |
| <b>Input parameter{in}</b>   |   |
| can_periph                   | CAN peripheral                                      |
| CANx(x=0, 1)                 | CAN peripheral selection                            |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable CAN0 debug freeze */
```

```
can_debug_freeze_disable(CAN0);
```

### **can\_time\_trigger\_mode\_enable**

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

**Table 3-83. Function can\_time\_trigger\_mode\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | can_time_trigger_mode_enable                            |
| <b>Function prototype</b>    | void can_time_trigger_mode_enable(uint32_t can_periph); |
| <b>Function descriptions</b> | enable CAN time trigger mode                            |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| can_periph                   | CAN peripheral  |
| CANx(x=0, 1)                 | CAN peripheral selection                                |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable CAN0 time trigger mode */
can_time_trigger_mode_enable(CAN0);
```

### **can\_time\_trigger\_mode\_disable**

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

**Table 3-84. Function can\_time\_trigger\_mode\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | can_time_trigger_mode_disable                            |
| <b>Function prototype</b>    | void can_time_trigger_mode_disable(uint32_t can_periph); |
| <b>Function descriptions</b> | disable CAN time trigger mode                            |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| can_periph                   | CAN peripheral   |
| CANx(x=0, 1)                 | CAN peripheral selection                                 |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable CAN0 time trigger mode */
```

```
can_time_trigger_mode_disable(CAN0);
```

### **can\_message\_transmit**

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

**Table 3-85. Function can\_message\_transmit**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | can_message_transmit  |
| <b>Function prototype</b>    | uint8_t can_message_transmit(uint32_t can_periph,<br>can_trasnmit_message_struct* transmit_message);  |
| <b>Function descriptions</b> | transmit CAN message  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| can_periph                   | CAN peripheral  |
| CANx(x=0,1)                  | CAN peripheral selection  |
| <b>Input parameter{in}</b>   |   |
| transmit_message             | CAN transmit message stuct, the structure members can refer to members of the structure <a href="#">Table 3-68. can_trasnmit_message_struct</a> |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| uint8_t                      | 0x00-0x03   |

Example:

```
/* CAN0 transmit message and return the mailbox number*/
uint8_t transmit_mailbox = 0;
can_trasnmit_message_struct transmit_message;
transmit_message.tx_sfid = 0x300 >> 1;
transmit_message.tx_efid = 0x00;
transmit_message.tx_ft = CAN_FT_DATA;
transmit_message.tx_ff = CAN_FF_STANDARD;
transmit_message.tx_dlen = 2;
transmit_message.tx_data[0] = 0x55;
transmit_message.tx_data[1] = 0xAA;
transmit_mailbox = can_message_transmit(CAN0, &transmit_message);
```

### **can\_transmit\_states**

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

**Table 3-86. Function can\_transmit\_states**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | can_transmit_states   |
| <b>Function prototype</b>    | can_transmit_state_enum can_transmit_states(uint32_t can_periph, uint8_t mailbox_number); |
| <b>Function descriptions</b> | get CAN transmit state  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| can_periph                   | CAN peripheral  |
| CANx(x=0,1)                  | CAN peripheral selection  |
| <b>Input parameter{in}</b>   |   |
| mailbox_number               | Mailbox number  |
| CAN_MAILBOXx(x=0,1,2)        | CAN_MAILBOXx  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| can_transmit_state_enum      | Refer to <a href="#">Table 3-74. can_transmit_state_enum</a>                              |

Example:

```
/* CAN0 mailbox0 transmit state */
can_transmit_state_enum transmit_state = CAN_TRANSMIT_FAILED;
transmit_state = can_transmit_states(CAN0, CAN_MAILBOX0);
```

### can\_transmission\_stop

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

**Table 3-87. Function can\_transmission\_stop**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | can_transmission_stop  |
| <b>Function prototype</b>    | void can_transmission_stop(uint32_t can_periph, uint8_t mailbox_number); |
| <b>Function descriptions</b> | stop CAN transmission  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| can_periph                   | CAN peripheral   |
| CANx(x=0,1)                  | CAN peripheral selection   |
| <b>Input parameter{in}</b>   |  |
| mailbox_number               | Mailbox number   |
| CAN_MAILBOXx(x=0,1,2)        | CAN_MAILBOXx   |
| <b>Output parameter{out}</b> |  |

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

Example:

```
/* stop CAN0 mailbox0 transmission */

can_transmission_stop(CAN0, CAN_MAILBOX0);
```

### can\_message\_receive

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

**Table 3-88. Function can\_message\_receive**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | can_message_receive  |
| <b>Function prototype</b>    | void can_message_receive(uint32_t can_periph, uint8_t fifo_number,<br>can_receive_message_struct* receive_message);                              |
| <b>Function descriptions</b> | CAN receive message  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| can_periph                   | CAN peripheral   |
| CANx(x=0,1)                  | CAN peripheral selection   |
| <b>Input parameter{in}</b>   |  |
| fifo_number                  | Fifo number  |
| CAN_FIFOx(x=0,1)             | CAN_FIFOx  |
| <b>Input parameter{in}</b>   |  |
| receive_message              | CAN message receive stuct, the structure members can refer to members<br>of the structure <a href="#">Table 3-69. can receive message struct</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* CAN0 FIFO0 receive message */

can_receive_message_struct receive_message;

can_message_receive(CAN0, CAN_FIFO0, &receive_message);
```

### can\_fifo\_release

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

Table 3-89. Function can\_fifo\_release

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | can_fifo_release   |
| <b>Function prototype</b>    | void can_fifo_release(uint32_t can_periph, uint8_t fifo_number); |
| <b>Function descriptions</b> | release FIFO   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| can_periph                   | CAN peripheral   |
| CANx(x=0,1)                  | CAN peripheral selection   |
| <b>Input parameter{in}</b>   |  |
| fifo_number                  | FIFO number  |
| CAN_FIFOx(x=0,1)             | CAN_FIFOx  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* CAN0 release FIFO0 */
can_fifo_release(CAN0, CAN_FIFO0);
```

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

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

Table 3-90. Function can\_receive\_message\_length\_get

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | can_receive_message_length_get  |
| <b>Function prototype</b>    | uint8_t can_receive_message_length_get(uint32_t can_periph, uint8_t fifo_number); |
| <b>Function descriptions</b> | CAN receive message length  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| can_periph                   | CAN peripheral  |
| CANx(x=0,1)                  | CAN peripheral selection  |
| <b>Input parameter{in}</b>   |   |
| fifo_number                  | FIFO number   |
| CAN_FIFOx(x=0,1)             | CAN_FIFOx   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| uint8_t                      | 0..3  |

Example:

```

/* CAN0 FIFO0 receive message length */

uint8_t frame_number = 0;

frame_number = can_receive_message_length_get(CAN0, CAN_FIFO0);

```

### **can\_working\_mode\_set**

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

**Table 3-91. Function can\_working\_mode\_set**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | can_working_mode_set   |
| <b>Function prototype</b>    | ErrStatus can_working_mode_set(uint32_t can_periph, uint8_t working_mode); |
| <b>Function descriptions</b> | set CAN working mode   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| can_periph                   | CAN peripheral   |
| CANx(x=0,1)                  | CAN peripheral selection   |
| <b>Input parameter{in}</b>   |  |
| can_working_mode             | Mode select  |
| CAN_MODE_INITIALIZE          | Initialize mode  |
| CAN_MODE_NORMAL              | Normal mode  |
| CAN_MODE_SLEEP               | Sleep mode   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>ErrStatus</b>             | SUCCESS / ERROR  |

Example:

```

/* set CAN0 working at initialize mode */

can_working_mode_set(CAN0, CAN_MODE_INITIALIZE);

```

### **can\_wakeup**

The description of can\_wakeup is shown as below:

**Table 3-92. Function can\_wakeup**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | can_wakeup                                 |
| <b>Function prototype</b>    | ErrStatus can_wakeup(uint32_t can_periph); |
| <b>Function descriptions</b> | wake up CAN                                |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |

| Input parameter{in}   |                          |
|-----------------------|--------------------------|
| can_periph            | CAN peripheral           |
| CANx(x=0,1)           | CAN peripheral selection |
| Output parameter{out} |                          |
| -                     | -                        |
| Return value          |                          |
| ErrStatus             | SUCCESS / ERROR          |

Example:

```
/* wake up CAN0 */

can_wakeup (CAN0);
```

### can\_error\_get

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

**Table 3-93. Function can\_error\_get**

| Function name         | can_error_get                                       |
|-----------------------|---|
| Function prototype    | can_error_enum can_error_get(uint32_t can_periph);  |
| Function descriptions | get CAN error type                                  |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |
| can_periph            | CAN peripheral                                      |
| CANx(x=0,1)           | CAN peripheral selection                            |
| Output parameter{out} |   |
| -                     | -   |
| Return value          |   |
| can_error_enum        | Refer to <a href="#">Table 3-73. can_error_enum</a> |

Example:

```
/* get CAN0 error type */

can_error_enum err_type;

err_type = can_error_get(CAN0);
```

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

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

**Table 3-94. Function can\_receive\_error\_number\_get**

|                       |  |
|-----------------------|--|
| Function name         | can_receive_error_number_get                               |
| Function prototype    | uint8_t can_receive_error_number_get(uint32_t can_periph); |
| Function descriptions | get CAN receive error number                               |

|                              |                          |
|------------------------------|--------------------------|
| <b>Precondition</b>          | -                        |
| <b>The called functions</b>  | -                        |
| <b>Input parameter{in}</b>   |                          |
| <b>can_periph</b>            | CAN peripheral           |
| <b>CANx(x=0, 1)</b>          | CAN peripheral selection |
| <b>Output parameter{out}</b> |                          |
| -                            | -                        |
| <b>Return value</b>          |                          |
| <b>uint8_t</b>               | 0..255                   |

Example:

```
/* get CAN0 receive error number */

uint8_t error_num;

error_num = can_receive_error_number_get(CAN0);
```

### **can\_transmit\_error\_number\_get**

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

**Table 3-95. Function can\_transmit\_error\_number\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | can_transmit_error_number_get                               |
| <b>Function prototype</b>    | uint8_t can_transmit_error_number_get(uint32_t can_periph); |
| <b>Function descriptions</b> | get CAN transmit error number                               |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>can_periph</b>            | CAN peripheral  |
| <b>CANx(x=0, 1)</b>          | CAN peripheral selection                                    |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>uint8_t</b>               | 0..255  |

Example:

```
/* get CAN0 transmit error number */

uint8_t error_num;

error_num = can_transmit_error_number_get(CAN0);
```

### **can\_flag\_get**

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

Table 3-96. Function can\_flag\_get

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | can_flag_get  |
| <b>Function prototype</b>    | FlagStatus can_flag_get(uint32_t can_periph, can_flag_enum flag); |
| <b>Function descriptions</b> | get CAN flag state  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| can_periph                   | CAN peripheral  |
| CANx(x=0,1)                  | CAN peripheral selection  |
| <b>Input parameter{in}</b>   |   |
| flag                         | CAN flags refer to <a href="#">Table 3-71. can_flag_enum</a>      |
| CAN_FLAG_RXL                 | RX level  |
| CAN_FLAG_LASTRX              | last sample value of RX pin                                       |
| CAN_FLAG_RS                  | receiving state   |
| CAN_FLAG_TS                  | transmitting state  |
| CAN_FLAG_SLPIF               | status change flag of entering sleep working mode                 |
| CAN_FLAG_WUIF                | status change flag of wakeup from sleep working mode              |
| CAN_FLAG_ERRIF               | error flag  |
| CAN_FLAG_SLPWS               | sleep working state   |
| CAN_FLAG_IWS                 | initial working state   |
| CAN_FLAG_TMLS2               | transmit mailbox 2 last sending in Tx FIFO                        |
| CAN_FLAG_TMLS1               | transmit mailbox 1 last sending in Tx FIFO                        |
| CAN_FLAG_TMLS0               | transmit mailbox 0 last sending in Tx FIFO                        |
| CAN_FLAG_TME2                | transmit mailbox 2 empty  |
| CAN_FLAG_TME1                | transmit mailbox 1 empty  |
| CAN_FLAG_TME0                | transmit mailbox 0 empty  |
| CAN_FLAG_MTE2                | mailbox 2 transmit error  |
| CAN_FLAG_MTE1                | mailbox 1 transmit error  |
| CAN_FLAG_MTE0                | mailbox 0 transmit error  |
| CAN_FLAG_MAL2                | mailbox 2 arbitration lost  |
| CAN_FLAG_MAL1                | mailbox 1 arbitration lost  |
| CAN_FLAG_MAL0                | mailbox 0 arbitration lost  |
| CAN_FLAG_MTFNER<br>R2        | mailbox 2 transmit finished with no error                         |
| CAN_FLAG_MTFNER<br>R1        | mailbox 1 transmit finished with no error                         |
| CAN_FLAG_MTFNER<br>R0        | mailbox 0 transmit finished with no error                         |
| CAN_FLAG_MTF2                | mailbox 2 transmit finished                                       |
| CAN_FLAG_MTF1                | mailbox 1 transmit finished                                       |
| CAN_FLAG_MTF0                | mailbox 0 transmit finished                                       |
| CAN_FLAG_RF00                | receive FIFO0 overfull  |

|                              |                        |
|------------------------------|------------------------|
| CAN_FLAG_RFF0                | receive FIFO0 full     |
| CAN_FLAG_RFO1                | receive FIFO1 overfull |
| CAN_FLAG_RFF1                | receive FIFO1 full     |
| CAN_FLAG_BOERR               | bus-off error          |
| CAN_FLAG_PERR                | passive error          |
| CAN_FLAG_WERR                | warning error          |
| <b>Output parameter{out}</b> |                        |
| -                            | -                      |
| <b>Return value</b>          |                        |
| FlagStatus                   | SET / RESET            |

Example:

```
/* get CAN0 mailbox 0 transmit finished flag */
can_flag_get(CAN0, CAN_FLAG_MTF0);
```

### can\_flag\_clear

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

**Table 3-97. Function can\_flag\_clear**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | can_flag_clear  |
| <b>Function prototype</b>    | void can_flag_clear(uint32_t can_periph, can_flag_enum flag); |
| <b>Function descriptions</b> | clear CAN flag state  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| can_periph                   | CAN peripheral  |
| CANx(x=0,1)                  | CAN peripheral selection                                      |
| <b>Input parameter{in}</b>   |   |
| flag                         | CAN flags refer to <a href="#">Table 3-71. can_flag_enum</a>  |
| CAN_FLAG_SLPIF               | status change flag of entering sleep working mode             |
| CAN_FLAG_WUIF                | status change flag of wakeup from sleep working mode          |
| CAN_FLAG_ERRIF               | error flag  |
| CAN_FLAG_MTE2                | mailbox 2 transmit error                                      |
| CAN_FLAG_MTE1                | mailbox 1 transmit error                                      |
| CAN_FLAG_MTE0                | mailbox 0 transmit error                                      |
| CAN_FLAG_MAL2                | mailbox 2 arbitration lost                                    |
| CAN_FLAG_MAL1                | mailbox 1 arbitration lost                                    |
| CAN_FLAG_MAL0                | mailbox 0 arbitration lost                                    |
| CAN_FLAG_MTFNER<br>R2        | mailbox 2 transmit finished with no error                     |
| CAN_FLAG_MTFNER<br>R1        | mailbox 1 transmit finished with no error                     |

|                              |   |
|------------------------------|---|
| <i>CAN_FLAG_MTFNER</i>       | mailbox 0 transmit finished with no error |
| <i>R0</i>                    |   |
| <i>CAN_FLAG_MTF2</i>         | mailbox 2 transmit finished               |
| <i>CAN_FLAG_MTF1</i>         | mailbox 1 transmit finished               |
| <i>CAN_FLAG_MTF0</i>         | mailbox 0 transmit finished               |
| <i>CAN_FLAG_RF00</i>         | receive FIFO0 overfull                    |
| <i>CAN_FLAG_RF00</i>         | receive FIFO0 full                        |
| <i>CAN_FLAG_RF01</i>         | receive FIFO1 overfull                    |
| <i>CAN_FLAG_RF01</i>         | receive FIFO1 full                        |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* clear CAN0 mailbox 0 transmit error flag */
can_flag_clear(CAN0, CAN_FLAG_MTE0);
```

### can\_interrupt\_enable

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

**Table 3-98. Function can\_interrupt\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | can_interrupt_enable  |
| <b>Function prototype</b>    | void can_interrupt_enable(uint32_t can_periph, uint32_t interrupt); |
| <b>Function descriptions</b> | enable CAN interrupt  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>can_periph</b>            | CAN peripheral  |
| <b>CANx(x=0,1)</b>           | CAN peripheral selection  |
| <b>Input parameter{in}</b>   |   |
| <b>interrupt</b>             | Interrupt type  |
| <b>CAN_INT_TME</b>           | transmit mailbox empty interrupt enable                             |
| <b>CAN_INT_RFNE0</b>         | receive FIFO0 not empty interrupt enable                            |
| <b>CAN_INT_RFF0</b>          | receive FIFO0 full interrupt enable                                 |
| <b>CAN_INT_RF00</b>          | receive FIFO0 overfull interrupt enable                             |
| <b>CAN_INT_RFNE1</b>         | receive FIFO1 not empty interrupt enable                            |
| <b>CAN_INT_RFF1</b>          | receive FIFO1 full interrupt enable                                 |
| <b>CAN_INT_RF01</b>          | receive FIFO1 overfull interrupt enable                             |
| <b>CAN_INT_WERR</b>          | warning error interrupt enable                                      |
| <b>CAN_INT_PERR</b>          | passive error interrupt enable                                      |
| <b>CAN_INT_BO</b>            | bus-off interrupt enable  |

|                              |                                |
|------------------------------|--------------------------------|
| <code>CAN_INT_ERRN</code>    | error number interrupt enable  |
| <code>CAN_INT_ERR</code>     | error interrupt enable         |
| <code>CAN_INT_WU</code>      | wakeup interrupt enable        |
| <code>CAN_INT_SLPW</code>    | sleep working interrupt enable |
| <b>Output parameter{out}</b> |                                |
| -                            | -                              |
| <b>Return value</b>          |                                |
| -                            | -                              |

Example:

```
/* CAN0 transmit mailbox empty interrupt */
can_interrupt_enable(CAN0, CAN_INT_TME);
```

### **can\_interrupt\_disable**

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

**Table 3-99. Function can\_interrupt\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | can_interrupt_disable  |
| <b>Function prototype</b>    | void can_interrupt_disable(uint32_t can_periph, uint32_t interrupt); |
| <b>Function descriptions</b> | disable CAN interrupt  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <code>can_periph</code>      | CAN peripheral   |
| <code>CANx(x=0,1)</code>     | CAN peripheral selection   |
| <b>Input parameter{in}</b>   |  |
| <code>interrupt</code>       | Interrupt type   |
| <code>CAN_INT_TME</code>     | transmit mailbox empty interrupt                                     |
| <code>CAN_INT_RFNE0</code>   | receive FIFO0 not empty interrupt                                    |
| <code>CAN_INT_RFF0</code>    | receive FIFO0 full interrupt   |
| <code>CAN_INT_RFO0</code>    | receive FIFO0 overfull interrupt                                     |
| <code>CAN_INT_RFNE1</code>   | receive FIFO1 not empty interrupt                                    |
| <code>CAN_INT_RFF1</code>    | receive FIFO1 full interrupt   |
| <code>CAN_INT_RFO1</code>    | receive FIFO1 overfull interrupt                                     |
| <code>CAN_INT_WERR</code>    | warning error interrupt  |
| <code>CAN_INT_PERR</code>    | passive error interrupt  |
| <code>CAN_INT_BO</code>      | bus-off interrupt  |
| <code>CAN_INT_ERRN</code>    | error number interrupt   |
| <code>CAN_INT_ERR</code>     | error interrupt  |
| <code>CAN_INT_WU</code>      | wakeup interrupt   |
| <code>CAN_INT_SLPW</code>    | sleep working interrupt  |
| <b>Output parameter{out}</b> |  |

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

Example:

```
/* CAN0 transmit mailbox empty interrupt disable */
can_interrupt_disable(CAN0, CAN_INT_TME);
```

### can\_interrupt\_flag\_get

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

**Table 3-100. Function can\_interrupt\_flag\_get**

|   |  |
|---|--|
| <b>Function name</b>                                  | can_interrupt_flag_get   |
| <b>Function prototype</b>                             | FlagStatus can_interrupt_flag_get(uint32_t can_periph,<br>can_interrupt_flag_enum flag); |
| <b>Function descriptions</b>                          | get CAN interrupt flag state   |
| <b>Precondition</b>                                   | -  |
| <b>The called functions</b>                           | -  |
| <b>Input parameter{in}</b>                            |  |
| can_periph  | CAN peripheral   |
| CANx(x=0,1)   | CAN peripheral selection   |
| <b>Input parameter{in}</b>                            |  |
| flag  | CAN interrupt flags refer to <a href="#">Table 3-72. can_interrupt_flag_enum</a>         |
| CAN_INT_FLAG_SLP <small>I</small><br><small>F</small> | status change interrupt flag of sleep working mode entering                              |
| CAN_INT_FLAG_WUI <small>F</small>                     | status change interrupt flag of wakeup from sleep working mode                           |
| CAN_INT_FLAG_ERR <small>I</small><br><small>F</small> | error interrupt flag   |
| CAN_INT_FLAG_MTF2                                     | mailbox 2 transmit finished interrupt flag   |
| CAN_INT_FLAG_MTF1                                     | mailbox 1 transmit finished interrupt flag   |
| CAN_INT_FLAG_MTF0                                     | mailbox 0 transmit finished interrupt flag   |
| CAN_INT_FLAG_RFO0                                     | receive FIFO0 overfull interrupt flag  |
| CAN_INT_FLAG_RFF0                                     | receive FIFO0 full interrupt flag  |
| CAN_INT_FLAG_RF01                                     | receive FIFO1 overfull interrupt flag  |
| CAN_INT_FLAG_RFF1                                     | receive FIFO1 full interrupt flag  |
| CAN_INT_FLAG_ERR<br><small>N</small>                  | error number interrupt flag  |
| CAN_INT_FLAG_BOE<br><small>RR</small>                 | bus-off error interrupt flag   |
| CAN_INT_FLAG_PER<br><small>R</small>                  | passive error interrupt flag   |
| CAN_INT_FLAG_WER                                      | warning error interrupt flag   |

|                              |             |
|------------------------------|-------------|
| R                            |             |
| <b>Output parameter{out}</b> |             |
| -                            | -           |
| <b>Return value</b>          |             |
| FlagStatus                   | SET / RESET |

Example:

```
/* get CAN0 mailbox 0 transmit finished interrupt flag */

FlagStatus Status;

Status = can_interrupt_flag_get(CAN0, CAN_INT_FLAG_MTF0);
```

### can\_interrupt\_flag\_clear

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

**Table 3-101. Function can\_interrupt\_flag\_clear**

|  |   |
|--|---|
| <b>Function name</b>                   | can_interrupt_flag_clear  |
| <b>Function prototype</b>              | void can_interrupt_flag_clear(uint32_t can_periph, can_interrupt_flag_enum flag); |
| <b>Function descriptions</b>           | clear CAN interrupt flag state  |
| <b>Precondition</b>                    | -   |
| <b>The called functions</b>            | -   |
| <b>Input parameter{in}</b>             |   |
| can_periph                             | CAN peripheral  |
| CANx(x=0,1)                            | CAN peripheral selection  |
| <b>Input parameter{in}</b>             |   |
| flag                                   | CAN interrupt flags refer to <a href="#">Table 3-72. can_interrupt_flag_enum</a>  |
| CAN_INT_FLAG_SLP <small>I</small><br>F | status change interrupt flag of sleep working mode entering                       |
| CAN_INT_FLAG_WUIF                      | status change interrupt flag of wakeup from sleep working mode                    |
| CAN_INT_FLAG_ERR <small>I</small><br>F | error interrupt flag  |
| CAN_INT_FLAG_MTF2                      | mailbox 2 transmit finished interrupt flag  |
| CAN_INT_FLAG_MTF1                      | mailbox 1 transmit finished interrupt flag  |
| CAN_INT_FLAG_MTF0                      | mailbox 0 transmit finished interrupt flag  |
| CAN_INT_FLAG_RF00                      | receive FIFO0 overfull interrupt flag   |
| CAN_INT_FLAG_RFF0                      | receive FIFO0 full interrupt flag   |
| CAN_INT_FLAG_RF01                      | receive FIFO1 overfull interrupt flag   |
| CAN_INT_FLAG_RFF1                      | receive FIFO1 full interrupt flag   |
| <b>Output parameter{out}</b>           |   |
| -                                      | -   |
| <b>Return value</b>                    |   |
| -                                      | -   |

Example:

```
/* clear CAN0 mailbox 0 transmit finished interrupt flag */
can_interrupt_flag_clear (CAN0, CAN_INT_FLAG_MTF0);
```

## 3.5. CAU

The Cryptographic Acceleration Unit supports acceleration of DES, Triple-DES or AES (128,192, or 256) algorithms. The CAU registers are listed in chapter [3.5.1](#) the CAU firmware functions are introduced in chapter [3.5.2](#)

### 3.5.1. Descriptions of Peripheral registers

CAU registers are listed in the table shown as below:

**Table 3-102. CAU Registers**

| Registers | Descriptions                   |
|-----------|--------------------------------|
| CAU_CTL   | control register               |
| CAU_STAT0 | status register 0              |
| CAU_DI    | data input register            |
| CAU_DO    | data output register           |
| CAU_DMAEN | DMA enable register            |
| CAU_INTEN | interrupt enable register      |
| CAU_STAT1 | status register 1              |
| CAU_INTF  | interrupt flag register        |
| CAU_KEY0H | key 0 high register            |
| CAU_KEY0L | key 0 low register             |
| CAU_KEY1H | key 1 high register            |
| CAU_KEY1L | key 1 low register             |
| CAU_KEY2H | key 2 high register            |
| CAU_KEY2L | key 2 low register             |
| CAU_KEY3H | key 3 high register            |
| CAU_KEY3L | key 3 low register             |
| CAU_IV0H  | initial vector 0 high register |
| CAU_IV0L  | initial vector 0 low register  |
| CAU_IV1H  | initial vector 1 high register |
| CAU_IV1L  | initial vector 1 low register  |

### 3.5.2. Descriptions of Peripheral functions

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

**Table 3-103. CAU firmware function**

| Function name          | Function description                                 |
|------------------------|--|
| cau_deinit             | reset the CAU peripheral                             |
| cau_enable             | enable the CAU peripheral                            |
| cau_disable            | disable the CAU peripheral                           |
| cau_dma_enable         | enable the CAU DMA interface                         |
| cau_dma_disable        | disable the CAU DMA interface                        |
| cau_init               | initialize the CAU peripheral                        |
| cau_aes_keysize_config | configure key size if used AES algorithm             |
| cau_key_init           | initialize the key parameters                        |
| cau_key_parameter_init | initialize the sturct cau_key_initpara               |
| cau_iv_init            | initialize the vectors parameters                    |
| cau_iv_parameter_init  | initialize the struct cau_iv_parameter_struct        |
| cau_fifo_flush         | flush the IN and OUT FIFOs                           |
| cau_enable_state_get   | return whether CAU peripheral is enabled or disabled |
| cau_data_write         | write data to the IN FIFO                            |
| cau_data_read          | return the last data entered into the output FIFO    |
| cau_aes_ecb            | encrypt and decrypt using AES in ECB mode            |
| cau_aes_cbc            | encrypt and decrypt using AES in CBC mode            |
| cau_aes_ctr            | encrypt and decrypt using AES in CTR mode            |
| cau_tdes_ecb           | encrypt and decrypt using TDES in ECB mode           |
| cau_tdes_cbc           | encrypt and decrypt using TDES in CBC mode           |
| cau_des_ecb            | encrypt and decrypt using DES in ECB mode            |
| cau_des_cbc            | encrypt and decrypt using DES in CBC mode            |
| cau_flag_get           | get the CAU flag status                              |
| cau_interrupt_enable   | enable the CAU interrupts                            |
| cau_interrupt_disable  | disable the CAU interrupts                           |
| cau_interrupt_flag_get | get the interrupt flag                               |

### Structure cau\_key\_parameter\_struct

**Table 3-104. Structure cau\_key\_parameter\_struct**

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

**Structure cau\_iv\_parameter\_struct**
**Table 3-105. Structure cau\_iv\_parameter\_struct**

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

**Enumeration cau\_text\_struct**
**Table 3-106. Enumeration cau\_text\_struct**

| Member name | Function description   |
|-------------|--|
| input       | pointer to the input buffer  |
| in_length   | length of the input buffer, must be a multiple of 8(DES and TDES) or 16(AES) |
| output      | pointer to the returned buffer   |

**cau\_deinit**

The description of cau\_deinit is shown as below:

**Table 3-107. Function cau\_deinit**

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

Example:

```
/* reset the CAU peripheral */

cau_deinit();
```

**cau\_enable**

The description of cau\_enable is shown as below:

**Table 3-108. Function cau\_enable**

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

Example:

```
/* enable the CAU peripheral */

cau_enable();
```

### cau\_disable

The description of cau\_disable is shown as below:

**Table 3-109. Function cau\_disable**

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

Example:

```
/* disable the CAU peripheral */

cau_disable();
```

### cau\_dma\_enable

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

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

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

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

Example:

```
/* enable the CAU DMA interface */

cau_dma_enable(CAU_DMA_INFIFO);
```

### cau\_dma\_disable

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

**Table 3-111. Function cau\_dma\_disable**

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

Example:

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

cau_dma_disable(CAU_DMA_INFIFO);
```

### cau\_init

The description of cau\_init is shown as below:

Table 3-112. Function cau\_init

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | cau_init   |
| <b>Function prototype</b>    | void cau_init(uint32_t algo_dir, uint32_t algo_mode, uint32_t swapping); |
| <b>Function descriptions</b> | initialize the CAU peripheral  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>algo_dir</b>              | algorithm direction  |
| <b>CAU_ENCRYPT</b>           | encrypt  |
| <b>CAU_DECRYPT</b>           | decrypt  |
| <b>Input parameter{in}</b>   |  |
| <b>algo_mode</b>             | algorithm mode selection   |
| <b>CAU_MODE_TDES_ECB</b>     | TDES-ECB (3DES Electronic codebook)                                      |
| <b>CAU_MODE_TDES_CBC</b>     | TDES-CBC (3DES Cipher block chaining)                                    |
| <b>CAU_MODE DES ECB</b>      | DES-ECB (simple DES Electronic codebook)                                 |
| <b>CAU_MODE DES CBC</b>      | DES-CBC (simple DES Cipher block chaining)                               |
| <b>CAU_MODE AES ECB</b>      | AES-ECB (AES Electronic codebook)  |
| <b>CAU_MODE AES CBC</b>      | AES-CBC (AES Cipher block chaining)                                      |
| <b>CAU_MODE AES CTR</b>      | AES-CTR (AES counter mode)   |
| <b>CAU_MODE AES KE</b>       | AES decryption key preparation mode                                      |
| <b>Input parameter{in}</b>   |  |
| <b>swapping</b>              | data swapping selection  |
| <b>CAU_SWAPPING_32BIT</b>    | no swapping  |
| <b>CAU_SWAPPING_16BIT</b>    | half-word swapping   |
| <b>CAU_SWAPPING_8BIT</b>     | bytes swapping   |
| <b>CAU_SWAPPING_1BIT</b>     | bit swapping   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

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

---

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

### **cau\_aes\_keysize\_config**

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

**Table 3-113. Function cau\_aes\_keysize\_config**

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

Example:

```
/* configure key size if used AES algorithm */
cau_aes_keysize_config(CAU_KEYSIZE_128BIT);
```

### **cau\_key\_init**

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

**Table 3-114. Function cau\_key\_init**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | cau_key_init   |
| <b>Function prototype</b>    | void cau_key_init(cau_key_parameter_struct* key_initpara);   |
| <b>Function descriptions</b> | initialize the key parameters  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>key_initpara</b>          | the key parameters, refer to structure <a href="#">Table 3-104. Structure cau_key_parameter_struct</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```

/* initialize the key parameters */

cau_key_parameter_struct key_initpara;

uint32_t key_128[4] = {0x2b7e1516U, 0x28aed2a6U, 0xabf71588U, 0x09cf4f3cU};

key_initpara.key_2_high = key_128[0];

key_initpara.key_2_low = key_128[1];

key_initpara.key_3_high = key_128[2];

key_initpara.key_3_low = key_128[3];

cau_key_init(&key_initpara);

```

### **cau\_key\_parameter\_init**

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

**Table 3-115. Function cau\_key\_parameter\_init**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | cau_key_parameter_init   |
| <b>Function prototype</b>    | void cau_key_parameter_init(cau_key_parameter_struct* key_initpara);                                   |
| <b>Function descriptions</b> | initialize the sturct cau_key_initpara   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| key_initpara                 | the key parameters, refer to structure <a href="#">Table 3-104. Structure cau_key_parameter_struct</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```

/* initialize the sturct cau_key_initpara */

cau_key_parameter_struct key_initpara;

cau_key_parameter_init(&key_initpara);

```

### **cau\_iv\_init**

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

**Table 3-116. Function cau\_iv\_init**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | cau_iv_init   |
| <b>Function prototype</b>    | void cau_iv_init(cau_iv_parameter_struct* iv_initpara); |
| <b>Function descriptions</b> | initialize the vectors parameters                       |

|                              |  |
|------------------------------|--|
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>iv_initpara</b>           | the vectors parameter, refer to structure <a href="#">Table 3-105. Structure cau_iv_parameter_struct</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* initialize the vectors parameters */

cau_iv_parameter_struct iv_initpara;

uint32_t vectors[4] = {0x00010203U, 0x04050607U, 0x08090A0BU, 0x0C0D0E0FU};

iv_initpara.iv_0_high = vectors[0];
iv_initpara.iv_0_low = vectors[1];
iv_initpara.iv_1_high = vectors[2];
iv_initpara.iv_1_low = vectors[3];
cau_iv_init(&iv_initpara);
```

### **cau\_iv\_parameter\_init**

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

**Table 3-117. Function cau\_iv\_parameter\_init**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | cau_iv_parameter_init  |
| <b>Function prototype</b>    | void cau_iv_parameter_init(cau_iv_parameter_struct* iv_initpara);  |
| <b>Function descriptions</b> | initialize the vectors parameters  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| <b>iv_initpara</b>           | the vectors parameter, refer to structure <a href="#">Table 3-105. Structure cau_iv_parameter_struct</a> |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* initialize the vectors parameters */
```

```

cau_iv_parameter_struct iv_initpara;
cau_iv_parameter_init(&iv_initpara);
  
```

### **cau\_fifo\_flush**

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

**Table 3-118. Function cau\_fifo\_flush**

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

Example:

```

/* flush the IN and OUT FIFOs */
cau_fifo_flush();
  
```

### **cau\_enable\_state\_get**

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

**Table 3-119. Function cau\_enable\_state\_get**

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

Example:

```

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

```

ControlStatus state;
state = cau_enable_state_get();

```

### **cau\_data\_write**

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

**Table 3-120. Function cau\_data\_write**

|                              |                                     |
|------------------------------|-------------------------------------|
| <b>Function name</b>         | cau_data_write                      |
| <b>Function prototype</b>    | void cau_data_write(uint32_t data); |
| <b>Function descriptions</b> | write data to the IN FIFO           |
| <b>Precondition</b>          | -                                   |
| <b>The called functions</b>  | -                                   |
| <b>Input parameter{in}</b>   |                                     |
| <b>data</b>                  | data to write(0x0 - 0xFFFFFFFF)     |
| <b>Output parameter{out}</b> |                                     |
| -                            | -                                   |
| <b>Return value</b>          |                                     |
| -                            | -                                   |

Example:

```

/* write data to the IN FIFO */
cau_data_write(0x10);

```

### **cau\_data\_read**

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

**Table 3-121. Function cau\_data\_read**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | cau_data_read                                     |
| <b>Function prototype</b>    | uint32_t cau_data_read(void);                     |
| <b>Function descriptions</b> | return the last data entered into the output FIFO |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>uint32_t</b>              | 0x0 – 0xFFFFFFFF                                  |

Example:

```

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

```

```

  uint32_t data;
  data = cau_data_read();

```

### **cau\_aes\_ecb**

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

**Table 3-122. Function cau\_aes\_ecb**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | cau_aes_ecb   |
| <b>Function prototype</b>    | ErrStatus cau_aes_ecb(uint32_t algo_dir, uint8_t *key, uint16_t keysize,<br>cau_text_struct *text); |
| <b>Function descriptions</b> | encrypt and decrypt using AES in ECB mode   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| algo_dir                     | algorithm direction   |
| CAU_ENCRYPT                  | encrypt   |
| CAU_DECRYPT                  | decrypt   |
| <b>Input parameter{in}</b>   |   |
| key                          | key used for AES algorithm  |
| <b>Input parameter{in}</b>   |   |
| keysize                      | length of the key, must be a 128, 192 or 256  |
| <b>Input parameter{in}</b>   |   |
| text                         | the text parameter, refer to structure <a href="#">Table 3-106. Enumeration cau_text_struct</a>     |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| ErrStatus                    | SUCCESS or ERROR  |

Example:

```

/* encrypt using AES in ECB mode */

ErrStatus status;

cau_text_struct text;

uint8_t plaintext[64] = {

  0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
  0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,
  0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,
  0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,
  0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
}

```

```

0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,
0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U
};

uint8_t encrypt_result[64];

text.input = plaintext;
text.in_length = 64;
text.output = encrypt_result;
status = cau_aes_ecb(CAU_ENCRYPT ,CAU_KEYSIZE_128BIT, 128, &text);

```

### **cau\_aes\_cbc**

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

**Table 3-123. Function cau\_aes\_cbc**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | cau_aes_cbc   |
| <b>Function prototype</b>    | ErrStatus cau_aes_cbc(uint32_t algo_dir, uint8_t *key, uint16_t keysize,<br>uint8_t iv[16], cau_text_struct *text); |
| <b>Function descriptions</b> | encrypt and decrypt using AES in CBC mode   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>algo_dir</b>              | algorithm direction   |
| <b>CAU_ENCRYPT</b>           | encrypt   |
| <b>CAU_DECRYPT</b>           | decrypt   |
| <b>Input parameter{in}</b>   |   |
| <b>key</b>                   | key used for AES algorithm  |
| <b>Input parameter{in}</b>   |   |
| <b>keysize</b>               | length of the key, must be a 128, 192 or 256  |
| <b>Input parameter{in}</b>   |   |
| <b>iv</b>                    | initialization vectors used for AES algorithm   |
| <b>Input parameter{in}</b>   |   |
| <b>text</b>                  | the text parameter, refer to structure <a href="#">Table 3-106. Enumeration cau_text_struct</a>                     |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>ErrStatus</b>             | SUCCESS or ERROR  |

Example:

```

/* encrypt using AES in CBC mode */

ErrStatus status;

cau_text_struct text;

uint8_t plaintext[64] = {

    0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
    0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,
    0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,
    0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,
    0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
    0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
    0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,
    0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U

};

uint8_t encrypt_result[64];

text.input = plaintext;
text.in_length = 64;
text.output = encrypt_result;
uint8_t iv[16] = {0x00}

status = cau_aes_cbc(CAU_ENCRYPT ,CAU_KEYSIZE_128BIT, 128, iv, &text);

```

### **cau\_aes\_ctr**

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

**Table 3-124. Function cau\_aes\_ctr**

| Function name                | cau_aes_ctr   |
|------------------------------|---|
| <b>Function prototype</b>    | ErrStatus cau_aes_ctr(uint32_t algo_dir, uint8_t *key, uint16_t keysize,<br>uint8_t iv[16], cau_text_struct *text); |
| <b>Function descriptions</b> | encrypt and decrypt using AES in CTR mode   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>algo_dir</b>              | algorithm direction   |
| <b>CAU_ENCRYPT</b>           | encrypt   |
| <b>CAU_DECRYPT</b>           | decrypt   |

| Input parameter{in}   |   |
|-----------------------|---|
| key                   | key used for AES algorithm  |
| Input parameter{in}   |   |
| keysize               | length of the key, must be a 128, 192 or 256  |
| Input parameter{in}   |   |
| iv                    | initialization vectors used for AES algorithm   |
| Input parameter{in}   |   |
| text                  | the text parameter, refer to structure <a href="#">Table 3-106. Enumeration cau_text_struct</a> |
| Output parameter{out} |   |
| -                     | -   |
| Return value          |   |
| ErrStatus             | SUCCESS or ERROR  |

Example:

```

/* encrypt using AES in CTR mode */

ErrStatus status;

cau_text_struct text;

uint8_t plaintext[64] = {
    0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
    0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,
    0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,
    0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,
    0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
    0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
    0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,
    0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U
};

uint8_t encrypt_result[64];

text.input = plaintext;
text.in_length = 64;
text.output = encrypt_result;
uint8_t iv[16] = {0x00};

status = cau_aes_ctr(CAU_ENCRYPT, CAU_KEYSIZE_128BIT, 128, iv, &text);

```

### cau\_tdes\_ecb

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

**Table 3-125. Function cau\_tdes\_ecb**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | cau_tdes_ecb  |
| <b>Function prototype</b>    | ErrStatus cau_tdes_ecb(uint32_t algo_dir, uint8_t key[24], cau_text_struct *text);              |
| <b>Function descriptions</b> | encrypt and decrypt using TDES in ECB mode  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| algo_dir                     | algorithm direction   |
| CAU_ENCRYPT                  | encrypt   |
| CAU_DECRYPT                  | decrypt   |
| <b>Input parameter{in}</b>   |   |
| key                          | key used for TDES algorithm   |
| <b>Input parameter{in}</b>   |   |
| text                         | the text parameter, refer to structure <a href="#">Table 3-106. Enumeration cau_text_struct</a> |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| ErrStatus                    | SUCCESS or ERROR  |

Example:

```
/* encrypt using TDES in ECB mode */

ErrStatus status;

cau_text_struct text;

uint8_t plaintext[64] = {

    0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
    0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,
    0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,
    0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,
    0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
    0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
    0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,
    0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U
}
```

```

};

uint8_t encrypt_result[64];

text.input = plaintext;

text.in_length = 64;

text.output = encrypt_result;

uint8_t key[24] = {0x01};

status = cau_tdes_ecb(CAU_ENCRYPT, key, &text);

```

### **cau\_tdes\_cbc**

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

**Table 3-126. Function cau\_tdes\_cbc**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | cau_tdes_cbc  |
| <b>Function prototype</b>    | ErrStatus cau_tdes_cbc(uint32_t algo_dir, uint8_t key[24], uint8_t iv[8], cau_text_struct *text); |
| <b>Function descriptions</b> | encrypt and decrypt using TDES in CBC mode  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>algo_dir</b>              | algorithm direction   |
| <b>CAU_ENCRYPT</b>           | encrypt   |
| <b>CAU_DECRYPT</b>           | decrypt   |
| <b>Input parameter{in}</b>   |   |
| <b>key</b>                   | key used for TDES algorithm   |
| <b>Input parameter{in}</b>   |   |
| <b>iv</b>                    | initialization vectors used for TDES algorithm  |
| <b>Input parameter{in}</b>   |   |
| <b>text</b>                  | the text parameter, refer to structure <a href="#">Table 3-106. Enumeration cau_text_struct</a>   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>ErrStatus</b>             | SUCCESS or ERROR  |

Example:

```

/* encrypt using TDES in CBC mode */

ErrStatus status;

cau_text_struct text;

uint8_t plaintext[64] = {

```

```

0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,
0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,
0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,
0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,
0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U
};

uint8_t encrypt_result[64];

text.input = plaintext;

text.in_length = 64;

text.output = encrypt_result;

uint8_t key[24] = {0x01};

uint8_t iv[8] = {0x00};

status = cau_tdes_cbc(CAU_ENCRYPT, key, iv, &text);

```

### **cau\_des\_ecb**

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

**Table 3-127. Function cau\_des\_ecb**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | cau_des_ecb   |
| <b>Function prototype</b>    | ErrStatus cau_des_ecb(uint32_t algo_dir, uint8_t *key, uint16_t keysize,<br>cau_text_struct *text); |
| <b>Function descriptions</b> | encrypt and decrypt using DES in ECB mode   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>algo_dir</b>              | algorithm direction   |
| <b>CAU_ENCRYPT</b>           | encrypt   |
| <b>CAU_DECRYPT</b>           | decrypt   |
| <b>Input parameter{in}</b>   |   |
| <b>key</b>                   | key used for DES algorithm  |
| <b>Input parameter{in}</b>   |   |
| <b>keysize</b>               | length of the key, must be a 128, 192 or 256  |
| <b>Input parameter{in}</b>   |   |

|                              |   |
|------------------------------|---|
| <b>text</b>                  | the text parameter, refer to structure <a href="#">Table 3-106. Enumeration cau_text_struct</a> |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>ErrStatus</b>             | SUCCESS or ERROR  |

Example:

```

/* encrypt using DES in ECB mode */

ErrStatus status;

cau_text_struct text;

uint8_t plaintext[64] = {

  0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
  0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,
  0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,
  0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,
  0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
  0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
  0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,
  0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U

};

uint8_t encrypt_result[64];

text.input = plaintext;
text.in_length = 64;
text.output = encrypt_result;
uint8_t key[24] = {0x01};
status = cau_des_ecb(CAU_ENCRYPT, key, &text);

```

### cau\_des\_cbc

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

**Table 3-128. Function cau\_des\_cbc**

|                           |  |
|---------------------------|--|
| <b>Function name</b>      | cau_des_cbc  |
| <b>Function prototype</b> | ErrStatus cau_des_cbc(uint32_t algo_dir, uint8_t key[24], uint8_t iv[8], |

|                              |   |
|------------------------------|---|
|                              | cau_text_struct *text);   |
| <b>Function descriptions</b> | encrypt and decrypt using DES in CBC mode   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>algo_dir</b>              | algorithm direction   |
| <b>CAU_ENCRYPT</b>           | encrypt   |
| <b>CAU_DECRYPT</b>           | decrypt   |
| <b>Input parameter{in}</b>   |   |
| <b>key</b>                   | key used for DES algorithm  |
| <b>Input parameter{in}</b>   |   |
| <b>iv</b>                    | initialization vectors used for DES algorithm   |
| <b>Input parameter{in}</b>   |   |
| <b>text</b>                  | the text parameter, refer to structure <a href="#">Table 3-106. Enumeration cau_text_struct</a> |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>ErrStatus</b>             | SUCCESS or ERROR  |

Example:

```

/* encrypt using DES in CBC mode */

ErrStatus status;

cau_text_struct text;

uint8_t plaintext[64] = {
    0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
    0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,
    0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,
    0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,
    0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
    0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
    0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,
    0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U
};

uint8_t encrypt_result[64];

text.input = plaintext;
  
```

```

text.in_length = 64;

text.output = encrypt_result;

uint8_t key[24] = {0x01};

uint8_t iv[8] = {0x00};

status = cau_des_cbc(CAU_ENCRYPT, key, iv, &text);
  
```

### **cau\_flag\_get**

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

**Table 3-129. Function cau\_flag\_get**

|                                       |   |
|---------------------------------------|---|
| <b>Function name</b>                  | cau_flag_get                            |
| <b>Function prototype</b>             | FlagStatus cau_flag_get(uint32_t flag); |
| <b>Function descriptions</b>          | get the CAU flag status                 |
| <b>Precondition</b>                   | -                                       |
| <b>The called functions</b>           | -                                       |
| <b>Input parameter{in}</b>            |   |
| <b>flag</b>                           | CAU flag status                         |
| <b>CAU_FLAG_INFIFO_E<br/>MPTY</b>     | input FIFO empty                        |
| <b>CAU_FLAG_INFIFO_N<br/>O_FULL:</b>  | input FIFO is not full                  |
| <b>CAU_FLAG_OUTFIFO<br/>_NO_EMPTY</b> | output FIFO not empty                   |
| <b>CAU_FLAG_OUTFIFO<br/>_FULL</b>     | output FIFO is full                     |
| <b>CAU_FLAG_BUSY</b>                  | the CAU core is busy                    |
| <b>CAU_FLAG_INFIFO</b>                | input FIFO flag status                  |
| <b>CAU_FLAG_OUTFIFO</b>               | output FIFO flag status                 |
| <b>Output parameter{out}</b>          |   |
| -                                     | -                                       |
| <b>Return value</b>                   |   |
| <b>FlagStatus</b>                     | SET or RESET                            |

Example:

```

/* get the CAU flag status */

FlagStatus status;

status = cau_flag_get(CAU_FLAG_INFIFO_EMPTY);
  
```

### **cau\_interrupt\_enable**

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

**Table 3-130. Function cau\_interrupt\_enable**

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

Example:

```
/* enable CAU interrupt */
cau_interrupt_enable(CAU_INT_INFIFO);
```

### **cau\_interrupt\_disable**

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

**Table 3-131. Function cau\_interrupt\_disable**

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

Example:

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

### **cau\_interrupt\_flag\_get**

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

**Table 3-132. Function cau\_interrupt\_flag\_get**

|                                 |   |
|---------------------------------|---|
| <b>Function name</b>            | cau_interrupt_flag_get                                |
| <b>Function prototype</b>       | FlagStatus cau_interrupt_flag_get(uint32_t int_flag); |
| <b>Function descriptions</b>    | get the interrupt flag                                |
| <b>Precondition</b>             | -   |
| <b>The called functions</b>     | -   |
| <b>Input parameter{in}</b>      |   |
| <b>int_flag</b>                 | CAU interrupt flag                                    |
| <b>CAU_INT_FLAG_INFIFO</b><br>0 | input FIFO interrupt                                  |
| <b>CAU_INT_FLAG_OUTFIFO</b>     | output FIFO interrupt                                 |
| <b>Output parameter{out}</b>    |   |
| -                               | -   |
| <b>Return value</b>             |   |
| <b>FlagStatus</b>               | SET or RESET  |

Example:

```
/* get the CAU interrupt flag status */
FlagStatus status;
status = cau_interrupt_flag_get(CAU_INT_FLAG_INFIFO);
```

## **3.6. CRC**

A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. The CRC registers are listed in chapter [3.6.1](#), the CRC firmware functions are introduced in chapter [3.6.2](#).

### **3.6.1. Descriptions of Peripheral registers**

CRC registers are listed in the table shown as below:

**Table 3-133. CRC Registers**

| <b>Registers</b> | <b>Descriptions</b>    |
|------------------|------------------------|
| CRC_DATA         | CRC data register      |
| CRC_FDATA        | CRC free data register |
| CRC_CTL          | CRC control register   |

### 3.6.2. Descriptions of Peripheral functions

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

**Table 3-134. CRC firmware function**

| Function name                | Function description   |
|------------------------------|--|
| crc_deinit                   | deinit CRC calculation unit                                      |
| crc_data_register_reset      | reset data register to the value of initializaiton 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 3-135. Function crc\_deinit**

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

Example:

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

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

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

**Table 3-136. Function crc\_data\_register\_reset**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | crc_data_register_reset  |
| <b>Function prototype</b>    | void crc_data_register_reset(void);                              |
| <b>Function descriptions</b> | reset data register to the value of initializaiton data register |
| <b>Precondition</b>          | -  |

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

Example:

```
/* reset crc data register */

crc_data_register_reset();
```

### crc\_data\_register\_read

The description of `crc_data_register_read` is shown as below:

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

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

Example:

```
/* read crc data register */

uint32_t crc_value = 0;

crc_value = crc_data_register_read();
```

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

The description of `crc_free_data_register_read` is shown as below:

**Table 3-138. Function `crc_free_data_register_read`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | crc_free_data_register_read                |
| <b>Function prototype</b>    | uint8_t crc_free_data_register_read(void); |
| <b>Function descriptions</b> | read the value of the free data register   |
| <b>Precondition</b>          | -  |

|                              |  |
|------------------------------|--|
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>uint8_t</b>               | 8-bit value of the free data register (0-0xFF) |

Example:

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

uint8_t crc_value = 0;

crc_value = crc_free_data_register_read();
```

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

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

**Table 3-139. 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>   |   |
| free_data                    | specify 8-bit data                                    |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* write the free data register */

crc_free_data_register_write(0x11);
```

### crc\_single\_data\_calculate

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

**Table 3-140. 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>   |   |
| sdata                        | specify 32-bit data                       |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| uint32_t                     | 32-bit CRC calculate value (0-0xFFFFFFFF) |

Example:

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

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

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

```
rcu_periph_clock_enable(RCU_CRC);
```

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

### crc\_block\_data\_calculate

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

**Table 3-141. Function crc\_block\_data\_calculate**

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

Example:

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

```
#define BUFFER_SIZE      6
```

```
uint32_t valcrc = 0;
```

```
static const uint32_t data_buffer[BUFFER_SIZE] = {
```

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

---

```

rcu_periph_clock_enable(RCU_CRC);

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

```

## 3.7. DAC

The Digital-to-analog converter converts 12-bit digital data to a voltage on the external pins. The DAC registers are listed in chapter [3.7.1](#), the DAC firmware functions are introduced in chapter [3.7.2](#).

### 3.7.1. Descriptions of Peripheral registers

DAC registers are listed in the table shown as below:

**Table 3-142. DAC Registers**

| Registers  | Descriptions   |
|------------|--|
| DAC_CTL    | DAC control register   |
| DAC_SWT    | DAC software trigger register                                  |
| DAC0_R12DH | DAC0 12-bit right-aligned data holding register                |
| DAC0_L12DH | DAC0 12-bit left-aligned data holding register                 |
| DAC0_R8DH  | DAC0 8-bit right-aligned data holding register                 |
| DAC1_R12DH | DAC1 12-bit right-aligned data holding register                |
| DAC1_L12DH | DAC1 12-bit left-aligned data holding register                 |
| DAC1_R8DH  | DAC1 8-bit right-aligned data holding register                 |
| DACC_R12DH | DAC concurrent mode 12-bit right-aligned data holding register |
| DACC_L12DH | DAC concurrent mode 12-bit left-aligned data holding register  |
| DACC_R8DH  | DAC concurrent mode 8-bit right-aligned data holding register  |
| DAC0_DO    | DAC0 data output register                                      |
| DAC1_DO    | DAC1 data output register                                      |

### 3.7.2. Descriptions of Peripheral functions

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

**Table 3-143. 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            |
| dac_dma_disable           | disable DAC DMA           |
| 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                                |
|---|---|
| dac_data_set                            | set DAC data holding register value                 |
| dac_trigger_enable                      | enable DAC trigger                                  |
| dac_trigger_disable                     | disable DAC trigger                                 |
| dac_trigger_source_config               | configure DAC trigger source                        |
| dac_software_trigger_enable             | enable DAC software trigger                         |
| dac_software_trigger_disable            | disable DAC software trigger                        |
| dac_wave_mode_config                    | configure DAC wave mode                             |
| dac_wave_bit_width_config               | configure DAC wave bit width                        |
| dac_lfsr_noise_config                   | configure DAC LFSR noise mode                       |
| dac_triangle_noise_config               | configure DAC triangle noise mode                   |
| dac_concurrent_enable                   | enable DAC concurrent mode                          |
| dac_concurrent_disable                  | disable DAC concurrent mode                         |
| dac_concurrent_software_trigger_enable  | enable DAC concurrent software trigger              |
| dac_concurrent_software_trigger_disable | disable DAC concurrent software trigger             |
| dac_concurrent_output_buffer_enable     | enable DAC concurrent buffer function               |
| dac_concurrent_output_buffer_disable    | disable DAC concurrent buffer function              |
| dac_concurrent_data_set                 | set DAC concurrent mode data holding register value |

### **dac\_deinit**

The description of dac\_deinit is shown as below:

**Table 3-144. Function dac\_deinit**

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

Example:

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

### **dac\_enable**

The description of `dac_enable` is shown as below:

**Table 3-145. Function `dac_enable`**

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

Example:

```
/* enable DAC */

dac_enable(DAC0);
```

### **dac\_disable**

The description of `dac_disable` is shown as below:

**Table 3-146. Function `dac_disable`**

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

Example:

```
/* disable DAC */

dac_disable(DAC0);
```

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

The description of `dac_dma_enable` is shown as below:

**Table 3-147. Function `dac_dma_enable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | dac_dma_enable                           |
| <b>Function prototype</b>    | void dac_dma_enable(uint32_t dac_periph) |
| <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>            | peripheral                               |
| <b>DACx</b>                  | peripheral selection(x = 0,1)            |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable DAC DMA function */

dac_dma_enable(DAC0);
```

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

The description of `dac_dma_disable` is shown as below:

**Table 3-148. Function `dac_dma_disable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | dac_dma_disable                           |
| <b>Function prototype</b>    | void dac_dma_disable(uint32_t dac_periph) |
| <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>            | peripheral                                |
| <b>DACx</b>                  | peripheral selection(x = 0,1)             |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable DAC DMA function */

dac_dma_disable(DAC0);
```

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

The description of `dac_output_buffer_enable` is shown as below:

**Table 3-149. Function `dac_output_buffer_enable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | dac_output_buffer_enable                           |
| <b>Function prototype</b>    | void dac_output_buffer_enable(uint32_t dac_periph) |
| <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>            | peripheral   |
| <b>DACx</b>                  | peripheral selection(x = 0,1)                      |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable DAC output buffer */
dac_output_buffer_enable(DAC0);
```

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

The description of `dac_output_buffer_disable` is shown as below:

**Table 3-150. Function `dac_output_buffer_disable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | dac_output_buffer_disable                           |
| <b>Function prototype</b>    | void dac_output_buffer_disable(uint32_t dac_periph) |
| <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>            | peripheral  |
| <b>DACx</b>                  | peripheral selection(x = 0,1)                       |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable DAC output buffer */
dac_output_buffer_disable(DAC0);
```

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

The description of `dac_output_value_get` is shown as below:

**Table 3-151. Function `dac_output_value_get`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | <code>dac_output_value_get</code>                               |
| <b>Function prototype</b>    | <code>uint16_t dac_output_value_get(uint32_t dac_periph)</code> |
| <b>Function descriptions</b> | get DAC output value  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <code>dac_periph</code>      | peripheral  |
| <code>DACx</code>            | peripheral selection(x = 0,1)                                   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <code>uint16_t</code>        | DAC output data (0x0000 – 0x07FF)-                              |

Example:

```
/* get DAC output value */

uint16_t data = 0U;

data = dac_output_value_get(DAC0);
```

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

The description of `dac_data_set` is shown as below:

**Table 3-152. 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, uint32_t dac_align, uint16_t data)</code> |
| <b>Function descriptions</b> | set the DAC specified data holding register value                                      |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <code>dac_periph</code>      | peripheral   |
| <code>DACx</code>            | peripheral selection(x =0,1)   |
| <b>Input parameter{in}</b>   |  |
| <code>dac_align</code>       | DAC align mode   |
| <code>DAC_ALIGN_8B_R</code>  | data right 8b alignment  |
| <code>DAC_ALIGN_12B_R</code> | data right 12b alignment   |
| <code>DAC_ALIGN_12B_L</code> | data left 12b alignment  |
| <b>Input parameter{in}</b>   |  |
| <code>data</code>            | The data sending to holding register   |

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

Example:

```
/* set the DAC specified data holding register value */
dac_data_set(DAC0,DAC_ALIGN_8B_R,0x00FF);
```

### dac\_trigger\_enable

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

**Table 3-153. Function dac\_trigger\_enable**

| Function name         | dac_trigger_enable                           |
|-----------------------|--|
| Function prototype    | void dac_trigger_enable(uint32_t dac_periph) |
| Function descriptions | enable DAC trigger                           |
| Precondition          | -  |
| The called functions  | -  |
| Input parameter{in}   |  |
| dac_periph            | peripheral                                   |
| DACx                  | peripheral selection(x = 0,1)                |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

Example:

```
/* enable DAC trigger */
dac_trigger_enable(DAC0);
```

### dac\_trigger\_disable

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

**Table 3-154. Function dac\_trigger\_disable**

| Function name         | dac_trigger_disable                           |
|-----------------------|---|
| Function prototype    | void dac_trigger_disable(uint32_t dac_periph) |
| Function descriptions | disable DAC trigger                           |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |
| dac_periph            | peripheral                                    |
| DACx                  | peripheral selection(x = 0,1)                 |

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

Example:

```
/* disable DAC trigger */
dac_trigger_disable(DAC0);
```

### dac\_trigger\_source\_config

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

**Table 3-155. Function dac\_trigger\_source\_config**

| Function name           | dac_trigger_source_config  |
|-------------------------|--|
| Function prototype      | void dac_trigger_source_config(uint32_t dac_periph,uint32_t triggersource) |
| Function descriptions   | set DAC trigger source   |
| Precondition            | -  |
| The called functions    | -  |
| Input parameter{in}     |  |
| dac_periph              | peripheral   |
| DACx                    | peripheral selection(x = 0,1)  |
| Input parameter{in}     |  |
| triggersource           | external triggers of DAC   |
| DAC_TRIGGER_T1_T<br>RGO | TIMER1 TRGO  |
| DAC_TRIGGER_T2_T<br>RGO | TIMER2 TRGO  |
| DAC_TRIGGER_T3_T<br>RGO | TIMER3 TRGO  |
| DAC_TRIGGER_T4_T<br>RGO | TIMER4 TRGO  |
| DAC_TRIGGER_T5_T<br>RGO | TIMER5 TRGO  |
| DAC_TRIGGER_T6_T<br>RGO | TIMER6 TRGO  |
| DAC_TRIGGER_EXTI_9      | EXTI interrupt line9 event   |
| DAC_TRIGGER_SOFT_WARE   | software trigger   |
| Output parameter{out}   |  |
| -                       | -  |
| Return value            |  |

|   |   |
|---|---|
| - | - |
|---|---|

Example:

```
/* set DAC trigger source*/
dac_trigger_source_config(DAC0,DAC_TRIGGER_T1_TRGO);
```

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

The description of `dac_software_trigger_enable` is shown as below:

**Table 3-156. Function `dac_software_trigger_enable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | dac_software_trigger_enable                           |
| <b>Function prototype</b>    | void dac_software_trigger_enable(uint32_t dac_periph) |
| <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>            | peripheral  |
| DACx                         | peripheral selection(x = 0,1)                         |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable DAC software trigger */
dac_software_trigger_enable(DAC0);
```

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

The description of `dac_software_trigger_disable` is shown as below:

**Table 3-157. Function `dac_software_trigger_disable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | dac_software_trigger_disable                           |
| <b>Function prototype</b>    | void dac_software_trigger_disable(uint32_t dac_periph) |
| <b>Function descriptions</b> | disable DAC software trigger                           |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>dac_periph</b>            | peripheral   |
| DACx                         | peripheral selection(x = 0,1)                          |
| <b>Output parameter{out}</b> |  |
| -                            | -  |

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

Example:

```
/* disable DAC software trigger */
dac_software_trigger_disable(DAC0);
```

### dac\_wave\_mode\_config

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

**Table 3-158. 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, 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>              | peripheral   |
| <i>DACx</i>                    | peripheral selection(x = 0,1)                                      |
| <b>Input parameter{in}</b>     |  |
| <b>wave_mode</b>               | wave_mode  |
| <i>DAC_WAVE_DISABLE</i>        | wave disable   |
| <i>DAC_WAVE_MODE_L_FSR</i>     | LFSR noise mode  |
| <i>DAC_WAVE_MODE_T_RIANGLE</i> | triangle noise mode  |
| <b>Output parameter{out}</b>   |  |
| -                              | -  |
| <b>Return value</b>            |  |
| -                              | -  |

Example:

```
/* configure DAC wave mode */
dac_wave_mode_config(DAC0, DAC_WAVE_DISABLE);
```

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

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

**Table 3-159. Function dac\_wave\_bit\_width\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | dac_wave_bit_width_config   |
| <b>Function prototype</b>    | void dac_wave_bit_width_config(uint32_t dac_periph, uint32_t bit_width) |
| <b>Function descriptions</b> | configure DAC wave bit width  |

|                              |  |
|------------------------------|--|
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>dac_periph</b>            | peripheral                                   |
| <i>DACx</i>                  | peripheral selection(x = 0,1)                |
| <b>Input parameter{in}</b>   |  |
| <b>bit_width</b>             | noise wave bit width                         |
| <i>DAC_WAVE_BIT_WID TH_x</i> | bit width of the wave signal is x(x = 1..12) |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

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

dac_wave_bit_width_config(DAC0,DAC_WAVE_BIT_WIDTH_1);
```

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

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

**Table 3-160. Function dac\_lfsr\_noise\_config**

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

Example:

```
/* configure DAC LFSR noise mode */

dac_lfsr_noise_config(DAC0,DAC_LFSR_BIT0);
```

### dac\_triangle\_noise\_config

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

**Table 3-161. Function dac\_triangle\_noise\_config**

|                                 |   |
|---------------------------------|---|
| <b>Function name</b>            | dac_triangle_noise_config   |
| <b>Function prototype</b>       | void dac_triangle_noise_config(uint32_t dac_periph, 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>               | peripheral  |
| <i>DACx</i>                     | peripheral selection(x = 0,1)   |
| <b>Input parameter{in}</b>      |   |
| <b>amplitude</b>                | the amplitude of triangle noise   |
| <i>DAC_TRIANGLE_AMPLITUDE_X</i> | triangle amplitude is x(x = 2 <sup>n</sup> -1(n = 1..12))               |
| <b>Output parameter{out}</b>    |   |
| -                               | -   |
| <b>Return value</b>             |   |
| -                               | -   |

Example:

```
/* configure DAC triangle noise mode */
dac_triangle_noise_config(DAC0,DAC_TRIANGLE_AMPLITUDE_1);
```

### dac\_concurrent\_enable

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

**Table 3-162. Function dac\_concurrent\_enable**

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

Example:

---

```
/* enable DAC concurrent mode */
```

```
dac_concurrent_enable();
```

### **dac\_concurrent\_disable**

The description of `dac_concurrent_disable` is shown as below:

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

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

Example:

```
/* disable DAC concurrent mode */
```

```
dac_concurrent_disable();
```

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

The description of `dac_concurrent_software_trigger_enable` is shown as below:

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

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | dac_concurrent_software_trigger_enable              |
| <b>Function prototype</b>    | void dac_concurrent_software_trigger_enable (void); |
| <b>Function descriptions</b> | enable DAC concurrent software trigger function     |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable DAC concurrent software trigger function */
```

```
dac_concurrent_software_trigger_enable();
```

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

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

**Table 3-165. Function dac\_concurrent\_software\_trigger\_disable**

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

Example:

```
/* disable DAC concurrent software trigger function */

dac_concurrent_software_trigger_disable();
```

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

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

**Table 3-166. Function dac\_concurrent\_output\_buffer\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | dac_concurrent_output_buffer_enable             |
| <b>Function prototype</b>    | void dac_concurrent_output_buffer_enable(void); |
| <b>Function descriptions</b> | enable DAC concurrent buffer function           |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable DAC concurrent buffer function */

dac_concurrent_output_buffer_enable();
```

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

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

**Table 3-167. Function dac\_concurrent\_output\_buffer\_disable**

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

Example:

```
/* disable DAC concurrent buffer function */

dac_concurrent_output_buffer_disable();
```

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

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

**Table 3-168. Function dac\_concurrent\_data\_set**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | dac_concurrent_data_set  |
| <b>Function prototype</b>    | void dac_concurrent_data_set(uint32_t dac_align, uint16_t data0, uint16_t data1) |
| <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_align</b>             | DAC align mode   |
| <i>DAC_ALIGN_8B_R</i>        | data right 8b alignment  |
| <i>DAC_ALIGN_12B_R</i>       | data right 12b alignment   |
| <i>DAC_ALIGN_12B_L</i>       | data left 12b alignment  |
| <b>Input parameter{in}</b>   |  |
| <b>data0</b>                 | The data sending to holding register of DAC0                                     |
| <b>Input parameter{in}</b>   |  |
| <b>data1</b>                 | The data sending to holding register of DAC1                                     |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |

|   |   |
|---|---|
| - | - |
|---|---|

Example:

```
/* set DAC concurrent mode data holding register value */
dac_concurrent_data_set(DAC_ALIGN_8B_R,0x00FF,0x00FF);
```

## 3.8. DBG

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

### 3.8.1. Descriptions of Peripheral registers

DBG registers are listed in the table shown as below:

**Table 3-169. DBG Registers**

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

### 3.8.2. Descriptions of Peripheral functions

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

**Table 3-170. DBG firmware function**

| Function name          | Function description                                      |
|------------------------|---|
| dbg_id_get             | read DBG_ID 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                              |
| dbg_trace_pin_mode_set | set trace pin mode  |

#### Enum dbg\_periph\_enum

**Table 3-171. Enum dbg\_periph\_enum**

| Member name     | Function description                    |
|-----------------|---|
| DBG_FWDGT_HOLD  | debug FWDGT kept when core is halted    |
| DBG_WWDGT_HOLD  | debug WWDGT kept when core is halted    |
| DBG_TIMER0_HOLD | hold TIMER0 counter when core is halted |
| DBG_TIMER1_HOLD | hold TIMER1 counter when core is halted |

| Member name      | Function description                     |
|------------------|--|
| DBG_TIMER2_HOLD  | hold TIMER2 counter when core is halted  |
| DBG_TIMER3_HOLD  | hold TIMER3 counter when core is halted  |
| DBG_CAN0_HOLD    | debug CAN0 kept 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_TIMER7_HOLD  | hold TIMER7 counter 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_CAN1_HOLD    | debug CAN1 kept 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_I2C2_HOLD    | hold I2C2 smbus when core is halted      |

### dbg\_id\_get

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

**Table 3-172. 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 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 3-173. Function `dbg_low_power_enable`**

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

Example:

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

### **dbg\_low\_power\_disable**

The description of `dbg_low_power_disable` is shown as below:

**Table 3-174. Function `dbg_low_power_disable`**

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

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

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 3-175. Function dbg\_periph\_enable**

|                       |   |
|-----------------------|---|
| Function name         | dbg_periph_enable   |
| Function prototype    | void dbg_periph_enable(dbg_periph_enum dbg_periph);                   |
| Function descriptions | enable peripheral behavior when the mcu is in debug mode              |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |
| dbg_periph            | peripheral refer to <a href="#">Table 3-171. 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_CANx_HOLD         | hold CANx counter when core is halted (x=0, 1)                        |
| DBG_I2Cx_HOLD         | hold I2Cx smbus when core is halted (x=0, 1, 2)                       |
| DBG_TIMERx_HOLD       | hold TIMERx counter when core is halted (x=0~13)                      |
| 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 3-176. 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 |

|                              |   |
|------------------------------|---|
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>dbg_periph</b>            | peripheral refer to <a href="#">Table 3-171. Enum dbg_periph enum</a> |
| <i>DBG_FWDGT_HOLD</i>        | debug FWDGT kept when core is halted                                  |
| <i>DBG_WWDGT_HOLD</i>        | debug WWDGT kept when core is halted                                  |
| <i>DBG_CANx_HOLD</i>         | hold CANx counter when core is halted (x=0, 1)                        |
| <i>DBG_I2Cx_HOLD</i>         | hold I2Cx smbus when core is halted (x=0, 1, 2)                       |
| <i>DBG_TIMERx_HOLD</i>       | hold TIMERx counter when core is halted (x=0~13)                      |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

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

dbg_periph_disable(DBG_TIMER0_HOLD);
```

### **dbg\_trace\_pin\_enable**

The description of `dbg_trace_pin_enable` is shown as below:

**Table 3-177. 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 3-178. 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();
```

### **dbg\_trace\_pin\_mode\_set**

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

**Table 3-179. Function dbg\_trace\_pin\_mode\_set**

|                                   |  |
|-----------------------------------|--|
| <b>Function name</b>              | dbg_trace_pin_mode_set                                     |
| <b>Function prototype</b>         | void dbg_trace_pin_mode_set(uint32_t trace_mode);          |
| <b>Function descriptions</b>      | trace pin mode selection                                   |
| <b>Precondition</b>               | -  |
| <b>The called functions</b>       | -  |
| <b>Input parameter{in}</b>        |  |
| <b>trace_mode</b>                 | trace pin mode selection                                   |
| <b>TRACE_MODE_ASYNC</b>           | trace pin used for synchronization mode                    |
| <b>TRACE_MODE_SYNC_DATASIZE_1</b> | trace pin used for synchronization mode and data size is 1 |
| <b>TRACE_MODE_SYNC_DATASIZE_2</b> | trace pin used for synchronization mode and data size is 2 |
| <b>TRACE_MODE_SYNC_DATASIZE_4</b> | trace pin used for synchronization mode and data size is 4 |
| <b>Output parameter{out}</b>      |  |
| -                                 | -  |
| <b>Return value</b>               |  |
| -                                 | -  |

Example:

---

```

/* trace pin mode selection */

dbg_trace_pin_mode_set(TRACE_MODE_ASYNC);

```

## 3.9. DCI

The DCI is a parallel interface able to capture video or picture from a camera via Digital Camera Interface. The DCI registers are listed in chapter [3.9.1](#), the DCI firmware functions are introduced in chapter [3.9.2](#).

### 3.9.1. Descriptions of Peripheral registers

DCI registers are listed in the table shown as below:

**Table 3-180. DCI Registers**

| Registers  | Descriptions                                |
|------------|---|
| DCI_CTL    | DCI control register                        |
| DCI_STAT0  | DCI status register 0                       |
| DCI_STAT1  | DCI status register 1                       |
| DCI_INTEN  | DCI interrupt enable register               |
| DCI_INTF   | DCI interrupt flag register                 |
| DCI_INTC   | DCI interrupt clear register                |
| DCI_SC     | DCI synchronization codes register          |
| DCI_SCUMSK | DCI synchronization codes unmask register   |
| DCI_CWSPOS | DCI cropping window start position register |
| DCI_CWSZ   | DCI cropping window size register           |
| DCI_DATA   | DCI data register                           |

### 3.9.2. Descriptions of Peripheral functions

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

**Table 3-181. DCI firmware function**

| Function name           | Function description            |
|-------------------------|---------------------------------|
| dci_deinit              | DCI deinit                      |
| dci_init                | initialize DCI registers        |
| dci_enable              | enable DCI function             |
| dci_disable             | disble DCI function             |
| dci_capture_enable      | enable DCI capture              |
| dci_capture_disable     | disble DCI capture              |
| dci_jpeg_enable         | enable DCI jpeg mode            |
| dci_jpeg_disable        | disble DCI jpeg mode            |
| dci_crop_window_enable  | enable cropping window function |
| dci_crop_window_disable | disble cropping window function |

| Function name                | Function description   |
|------------------------------|--|
| dci_crop_window_config       | config DCI cropping window                                   |
| dci_embedded_sync_enable     | enable embedded synchronous mode                             |
| dci_embedded_sync_disable    | disble embedded synchronous mode                             |
| dci_sync_codes_config        | config synchronous codes in embedded synchronous mode        |
| dci_sync_codes_unmask_config | config synchronous codes unmask in embedded synchronous mode |
| dci_data_read                | read DCI data register                                       |
| dci_flag_get                 | get specified flag   |
| dci_interrupt_enable         | enable specified DCI interrupt                               |
| dci_interrupt_disable        | disable specified DCI interrupt                              |
| dci_interrupt_flag_get       | get specified interrupt flag                                 |
| dci_interrupt_flag_clear     | clear specified interrupt flag                               |

### Structure dci\_parameter\_struct

Table 3-182. Structure dci\_parameter\_struct

| Member name      | Function description   |
|------------------|--|
| capture_mode     | DCI capture mode: DCI_CAPTURE_MODE_CONTINUOUS / DCI_CAPTURE_MODE_SNAPSHOT  |
| clock_polarity   | clock polarity selection: DCI_CK_POLARITY_FALLING / DCI_CK_POLARITY_RISING   |
| hsync_polarity   | horizontal polarity selection: DCI_HSYNC_POLARITY_LOW / DCI_HSYNC_POLARITY_HIGH  |
| vsync_polarity   | vertical polarity selection: DCI_VSYNC_POLARITY_LOW / DCI_VSYNC_POLARITY_HIGH  |
| frame_rate       | frame capture rate: DCI_FRAME_RATE_ALL / DCI_FRAME_RATE_1_2 / DCI_FRAME_RATE_1_4   |
| interface_format | digital camera interface format:<br>DCI_INTERFACE_FORMAT_8BITS / DCI_INTERFACE_FORMAT_10BITS / DCI_INTERFACE_FORMAT_12BITS / DCI_INTERFACE_FORMAT_14BITS |

### dci\_deinit

The description of dci\_deinit is shown as below:

Table 3-183. Function dci\_deinit

|                       |                        |
|-----------------------|------------------------|
| Function name         | dci_deinit             |
| Function prototype    | void dci_deinit(void); |
| Function descriptions | DCI deinit             |
| Precondition          | -                      |
| The called functions  | -                      |

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

Example:

```
/* DCI deinit */

dci_deinit();
```

### dci\_init

The description of dci\_init is shown as below:

**Table 3-184. Function dci\_init**

| Function name         | dci_init   |
|-----------------------|--|
| Function prototype    | void dci_init(dci_parameter_struct* dci_struct);   |
| Function descriptions | initialize DCI registers   |
| Precondition          | -  |
| The called functions  | -  |
| Input parameter{in}   |  |
| dci_struct            | DCI parameter initialization stuct, refer to <a href="#">Table 3-182. Structure dci_parameter_struct</a> |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

Example:

```
/* initialize DCI registers */

dci_parameter_struct  dci_struct;

dci_struct.capture_mode = DCI_CAPTURE_MODE_CONTINUOUS;
dci_struct.clock_polarity = DCI_CK_POLARITY_RISING;
dci_struct.hsync_polarity = DCI_HSYNC_POLARITY_LOW;
dci_struct.vsync_polarity = DCI_VSYNC_POLARITY_LOW;
dci_struct.frame_rate = DCI_FRAME_RATE_ALL;
dci_struct.interface_format = DCI_INTERFACE_FORMAT_8BITS;
dci_init(&dci_struct);
```

### dci\_enable

The description of dci\_enable is shown as below:

**Table 3-185. Function dci\_enable**

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

Example:

```
/* enable DCI function */

dci_enable();
```

### dci\_disable

The description of dci\_disable is shown as below:

**Table 3-186. Function dci\_disable**

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

Example:

```
/* disable DCI function */

dci_disable();
```

### dci\_capture\_enable

The description of dci\_capture\_enable is shown as below:

**Table 3-187. Function dci\_capture\_enable**

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

Example:

```
/* enable DCI capture */

dci_capture_enable();
```

### dci\_capture\_disable

The description of dci\_capture\_disable is shown as below:

**Table 3-188. Function dci\_capture\_disable**

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

Example:

```
/* disable DCI capture */

dci_capture_disable();
```

### dci\_jpeg\_enable

The description of dci\_jpeg\_enable is shown as below:

**Table 3-189. Function dci\_jpeg\_enable**

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

Example:

```
/* enable DCI jpeg mode */
dci_jpeg_enable();
```

### dci\_jpeg\_disable

The description of dci\_jpeg\_disable is shown as below:

**Table 3-190. Function dci\_jpeg\_disable**

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

Example:

```
/* disable DCI jpeg mode */
dci_jpeg_disable();
```

### dci\_crop\_window\_enable

The description of dci\_crop\_window\_enable is shown as below:

**Table 3-191. Function dci\_crop\_window\_enable**

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

Example:

```
/* enable cropping window function */
dci_crop_window_enable();
```

### dci\_crop\_window\_disable

The description of dci\_crop\_window\_disable is shown as below:

**Table 3-192. Function dci\_crop\_window\_disable**

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

Example:

```
/* disable cropping window function */
dci_crop_window_disable();
```

### dci\_crop\_window\_config

The description of dci\_crop\_window\_config is shown as below:

**Table 3-193. Function dci\_crop\_window\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | dci_crop_window_config  |
| <b>Function prototype</b>    | void dci_crop_window_config(uint16_t start_x, uint16_t start_y, uint16_t size_width, uint16_t size_height); |
| <b>Function descriptions</b> | config DCI cropping window  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| start_x                      | window horizontal start position  |
| <b>Input parameter{in}</b>   |   |
| start_y                      | window vertical start position  |
| <b>Input parameter{in}</b>   |   |
| size_width                   | window horizontal size  |
| <b>Input parameter{in}</b>   |   |
| size_height                  | window vertical size  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* config DCI cropping window */
dci_crop_window_config(0x800, 0x600, 0x100, 0x100);
```

### dci\_embedded\_sync\_enable

The description of dci\_embedded\_sync\_enable is shown as below:

**Table 3-194. Function dci\_embedded\_sync\_enable**

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

Example:

```
/* enable embedded synchronous mode */
dci_embedded_sync_enable();
```

### **dci\_embedded\_sync\_disable**

The description of dci\_embedded\_sync\_disable is shown as below:

**Table 3-195. Function dci\_embedded\_sync\_disable**

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

Example:

```
/* disable embedded synchronous mode */
dci_embedded_sync_disable();
```

### **dci\_sync\_codes\_config**

The description of dci\_sync\_codes\_config is shown as below:

**Table 3-196. Function dci\_sync\_codes\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | dci_sync_codes_config   |
| <b>Function prototype</b>    | void dci_sync_codes_config(uint8_t frame_start, uint8_t line_start, uint8_t line_end, uint8_t frame_end); |
| <b>Function descriptions</b> | config synchronous codes in embedded synchronous mode   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| frame_start                  | frame start code in embedded synchronous mode   |
| <b>Input parameter{in}</b>   |   |
| line_start                   | line start code in embedded synchronous mode  |
| <b>Input parameter{in}</b>   |   |
| line_end                     | line end code in embedded synchronous mode  |
| <b>Input parameter{in}</b>   |   |
| frame_end                    | frame end code in embedded synchronous mode   |

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

Example:

```
/* config synchronous codes in embedded synchronous mode */
dci_sync_codes_config(0x10, 0x10, 0x20, 0x20);
```

### dci\_sync\_codes\_unmask\_config

The description of dci\_sync\_codes\_unmask\_config is shown as below:

**Table 3-197. Function dci\_sync\_codes\_unmask\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | dci_sync_codes_unmask_config  |
| <b>Function prototype</b>    | void dci_sync_codes_unmask_config(uint8_t frame_start, uint8_t line_start,<br>uint8_t line_end, uint8_t frame_end); |
| <b>Function descriptions</b> | config synchronous codes unmask in embedded synchronous mode  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| frame_start                  | frame start code in embedded synchronous mode   |
| <b>Input parameter{in}</b>   |   |
| line_start                   | line start code in embedded synchronous mode  |
| <b>Input parameter{in}</b>   |   |
| line_end                     | line end code in embedded synchronous mode  |
| <b>Input parameter{in}</b>   |   |
| frame_end                    | frame end code in embedded synchronous mode   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* config synchronous codes unmask in embedded synchronous mode */
dci_sync_codes_unmask_config(0x10, 0x10, 0x20, 0x20);
```

### dci\_data\_read

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

**Table 3-198. Function dci\_data\_read**

|                              |                               |
|------------------------------|-------------------------------|
| <b>Function name</b>         | dci_data_read                 |
| <b>Function prototype</b>    | uint32_t dci_data_read(void); |
| <b>Function descriptions</b> | read DCI data register        |

|                              |                   |
|------------------------------|-------------------|
| <b>Precondition</b>          | -                 |
| <b>The called functions</b>  | -                 |
| <b>Input parameter{in}</b>   |                   |
| -                            | -                 |
| <b>Output parameter{out}</b> |                   |
| -                            | -                 |
| <b>Return value</b>          |                   |
| <b>uint32_t</b>              | 0x00 – 0xFFFFFFFF |

Example:

```
/* read DCI data register */

uint32_t data = 0;

data = dci_data_read();
```

### dci\_flag\_get

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

**Table 3-199. Function dci\_flag\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | dci_flag_get                            |
| <b>Function prototype</b>    | FlagStatus dci_flag_get(uint32_t flag); |
| <b>Function descriptions</b> | get specified flag                      |
| <b>Precondition</b>          | -                                       |
| <b>The called functions</b>  | -                                       |
| <b>Input parameter{in}</b>   |   |
| <b>flag</b>                  | DCI flag                                |
| <i>DCI_FLAG_HS</i>           | HS line status                          |
| <i>DCI_FLAG_VS</i>           | VS line status                          |
| <i>DCI_FLAG_FV</i>           | FIFO valid                              |
| <i>DCI_FLAG_EF</i>           | end of frame flag                       |
| <i>DCI_FLAG_OVR</i>          | FIFO overrun flag                       |
| <i>DCI_FLAG_ESE</i>          | embedded synchronous error flag         |
| <i>DCI_FLAG_VSYNC</i>        | vsync flag                              |
| <i>DCI_FLAG_EL</i>           | end of line flag                        |
| <b>Output parameter{out}</b> |   |
| -                            | -                                       |
| <b>Return value</b>          |   |
| FlagStatus                   | SET or RESET                            |

Example:

```
/* get specified flag */

FlagStatus status = RESET;
```

```
status = dci_flag_get (DCI_FLAG_HS);
```

### dci\_interrupt\_enable

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

**Table 3-200. Function dci\_interrupt\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | dci_interrupt_enable                           |
| <b>Function prototype</b>    | void dci_interrupt_enable(uint32_t interrupt); |
| <b>Function descriptions</b> | enable specified DCI interrupt                 |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>interrupt</b>             | DCI interrupt                                  |
| <i>DCI_INT_EF</i>            | end of frame interrupt                         |
| <i>DCI_INT_OVR</i>           | FIFO overrun interrupt                         |
| <i>DCI_INT_ESE</i>           | embedded synchronous error interrupt           |
| <i>DCI_INT_VSYNC</i>         | vsync interrupt                                |
| <i>DCI_INT_EL</i>            | end of line interrupt                          |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable specified DCI interrupt */
dci_interrupt_enable(DCI_INT_EF);
```

### dci\_interrupt\_disable

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

**Table 3-201. Function dci\_interrupt\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | dci_interrupt_disable                           |
| <b>Function prototype</b>    | void dci_interrupt_disable(uint32_t interrupt); |
| <b>Function descriptions</b> | disable specified DCI interrupt                 |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>interrupt</b>             | DCI interrupt                                   |
| <i>DCI_INT_EF</i>            | end of frame interrupt                          |
| <i>DCI_INT_OVR</i>           | FIFO overrun interrupt                          |
| <i>DCI_INT_ESE</i>           | embedded synchronous error interrupt            |
| <i>DCI_INT_VSYNC</i>         | vsync interrupt                                 |

|                              |                       |
|------------------------------|-----------------------|
| <i>DCI_INT_EL</i>            | end of line interrupt |
| <b>Output parameter{out}</b> |                       |
| -                            | -                     |
| <b>Return value</b>          |                       |
| -                            | -                     |

Example:

```
/* disable specified DCI interrupt */
dci_interrupt_disable(DCI_INT_EF);
```

### dci\_interrupt\_flag\_get

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

**Table 3-202. Function dci\_interrupt\_flag\_get**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | dci_interrupt_flag_get                                 |
| <b>Function prototype</b>    | FlagStatus dci_interrupt_flag_get(uint32_t interrupt); |
| <b>Function descriptions</b> | get specified interrupt flag                           |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <i>interrupt</i>             | DCI interrupt  |
| <i>DCI_INT_FLAG_EF</i>       | end of frame interrupt flag                            |
| <i>DCI_INT_FLAG_OVR</i>      | FIFO overrun interrupt flag                            |
| <i>DCI_INT_FLAG_ESE</i>      | embedded synchronous error interrupt flag              |
| <i>DCI_INT_FLAG_VSYN_C</i>   | vsync interrupt flag                                   |
| <i>DCI_INT_FLAG_EL</i>       | end of line interrupt flag                             |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| FlagStatus                   | SET or RESET   |

Example:

```
/* get specified interrupt flag */
FlagStatus status = RESET;
status = dci_interrupt_flag_get(DCI_INT_FLAG_EF);
```

### dci\_interrupt\_flag\_clear

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

**Table 3-203. Function dci\_interrupt\_flag\_clear**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | dci_interrupt_flag_clear                           |
| <b>Function prototype</b>    | void dci_interrupt_flag_clear(uint32_t interrupt); |
| <b>Function descriptions</b> | clear specified interrupt flag                     |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <i>interrupt</i>             | DCI interrupt                                      |
| <i>DCI_INT_FLAG_EF</i>       | end of frame interrupt flag                        |
| <i>DCI_INT_FLAG_OVR</i>      | FIFO overrun interrupt flag                        |
| <i>DCI_INT_FLAG_ESE</i>      | embedded synchronous error interrupt flag          |
| <i>DCI_INT_FLAG_VSYN_C</i>   | vsync interrupt flag                               |
| <i>DCI_INT_FLAG_EL</i>       | end of line interrupt flag                         |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/*clear specified interrupt flag */
dci_interrupt_flag_clear(DCI_INT_FLAG_EF);
```

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

### 3.10.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

**Table 3-204. 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    |

| Registers                | Descriptions                               |
|--------------------------|--|
| DMA_CHxPADDR<br>(x=0..6) | channel x peripheral base address register |
| DMA_CHxMADDR<br>(x=0..6) | channel x memory base address register     |
| DMA_ACFG                 | additional configuration register          |

### 3.10.2. Descriptions of Peripheral functions

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

**Table 3-205. DMA firmware function**

| Function name                       | Function description  |
|-------------------------------------|---|
| dma_deinit                          | deinitialize DMA a channel registers                            |
| dma_struct_para_init                | initialize the parameters of DMA struct with the default values |
| dma_init                            | initialize DMA channel  |
| dma_circulation_enable              | enable DMA circulation mode                                     |
| dma_circulation_disable             | disable DMA circulation mode                                    |
| dma_memory_to_memory_enable         | enable memory to memory mode                                    |
| dma_memory_to_memory_disable        | disable memory to memory mode                                   |
| dma_channel_enable                  | enable DMA channel  |
| dma_channel_disable                 | disable DMA channel   |
| dma_periph_address_config           | set DMA peripheral base address                                 |
| dma_memory_address_config           | set DMA memory base address                                     |
| dma_transfer_number_config          | set the number of remaining data to be transferred by the DMA   |
| dma_transfer_number_get             | get the number of remaining data to be transferred by the DMA   |
| dma_priority_config                 | configure priority level of DMA channel                         |
| dma_memory_width_config             | configure transfer data size of memory                          |
| dma_periph_width_config             | configure transfer data size of peripheral                      |
| dma_memory_increase_enable          | enable next address increasement algorithm of memory            |
| dma_memory_increase_disable         | disable next address increasement algorithm of memory           |
| dma_periph_increase_enable          | enable next address increasement algorithm of peripheral        |
| dma_periph_increase_disable         | disable next address increasement algorithm of peripheral       |
| dma_transfer_direction_config       | configure the direction of data transfer on the channel         |
| dma_flag_get                        | check DMA flag is set or not                                    |
| dma_flag_clear                      | clear DMA a channel flag  |
| dma_interrupt_flag_get              | check DMA flag and interrupt enable bit is set or not           |
| dma_interrupt_flag_clear            | clear DMA a channel flag  |
| dma_interrupt_enable                | enable DMA interrupt  |
| dma_interrupt_disable               | disable DMA interrupt   |
| dma_1_channel_5_fulldata_transfer_e | enable the DMA1 channel 5 Full_Data transfer mode               |

| Function name                                 | Function description                               |
|---|--|
| nable   |  |
| dma_1_channel_5_fulldata_transfer_d<br>isable | disable the DMA1 channel 5 Full_Data transfer mode |

### Structure dma\_parameter\_struct

Table 3-206. Structure dma\_parameter\_struct

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

### dma\_deinit

The description of dma\_deinit is shown as below:

Table 3-207. Function dma\_deinit

|                                    |  |
|------------------------------------|--|
| Function name                      | dma_deinit   |
| Function prototype                 | void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx); |
| Function descriptions              | deinitialize DMA a channel registers                             |
| Precondition                       | -  |
| The called functions               | -  |
| Input parameter{in}                |  |
| dma_periph                         | DMA peripheral   |
| DMAx(x=0,1)                        | DMA peripheral selection   |
| Input parameter{in}                |  |
| channelx                           | DMA channel  |
| DMA_CHx(DMA0:x=0..6; DMA1: x=0..6) | DMA channel selection  |
| Output parameter{out}              |  |
| -                                  | -  |
| Return value                       |  |
| -                                  | -  |

Example:

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

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

The description of `dma_struct_para_init` is shown as below:

**Table 3-208. Function `dma_struct_para_init`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | dma_struct_para_init  |
| <b>Function prototype</b>    | <code>void dma_struct_para_init(dma_parameter_struct* init_struct);</code>  |
| <b>Function descriptions</b> | Initialize the parameters of DMA struct with the default values   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| <b>Init_struct</b>           | Structure for initialization, the structure members can refer to <a href="#">Table 3-206.</a><br><a href="#">Structure dma_parameter_struct</a> |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* initialize the parameters of DMA structure with the default values */

dma_parameter_struct init_struct;

dma_struct_para_init (&init_struct);
```

### **dma\_init**

The description of `dma_init` is shown as below:

**Table 3-209. Function `dma_init`**

|   |   |
|---|---|
| <b>Function name</b>                    | dma_init  |
| <b>Function prototype</b>               | <code>void dma_init(uint32_t dma_periph, dma_channel_enum channelx,<br/>dma_parameter_struct init_struct);</code>                               |
| <b>Function descriptions</b>            | initialize 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=..6; DMA1: x=..6)</i> | DMA channel selection   |
| <b>Input parameter{in}</b>              |   |
| <b>init_struct</b>                      | Structure for initialization, the structure members can refer to <a href="#">Table 3-206.</a><br><a href="#">Structure dma_parameter_struct</a> |

| <u>Structure dma_parameter_struct</u> |   |
|---------------------------------------|---|
| Output parameter{out}                 |   |
| -                                     | - |
| <b>Return value</b>                   |   |
| -                                     | - |

Example:

```

/* initialize DMA0 channel0 */

dma_parameter_struct dma_init_struct;

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

```

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

The description of `dma_circulation_enable` is shown as below:

**Table 3-210. 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>  | -  |
| <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=..)</i>    | DMA channel selection  |

|                              |   |
|------------------------------|---|
| 6; DMA1: x=0..6)             |   |
| <b>Output parameter{out}</b> |   |
| -                            | - |
| <b>Return value</b>          |   |
| -                            | - |

Example:

```
/* DMA0 channel0 mode configuration */

dma_circulation_enable(DMA0, DMA_CH0);
```

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

The description of `dma_circulation_disable` is shown as below:

**Table 3-211. 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  |
| <b>DMAx(x=0,1)</b>                        | DMA peripheral selection  |
| <b>Input parameter{in}</b>                |   |
| <b>channelx</b>                           | DMA channel   |
| <b>DMA_CHx(DMA0:x=0..6; DMA1: x=0..6)</b> | DMA channel selection   |
| <b>Output parameter{out}</b>              |   |
| -   | -   |
| <b>Return value</b>                       |   |
| -   | -   |

Example:

```
/* DMA0 channel0 mode configuration */

dma_circulation_disable(DMA0, DMA_CH0);
```

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

The description of `dma_memory_to_memory_enable` is shown as below:

**Table 3-212. 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, |

|   |                              |
|---|------------------------------|
|   | 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..6)</i> | DMA channel selection        |
| <b>Output parameter{out}</b>              |                              |
| -   | -                            |
| <b>Return value</b>                       |                              |
| -   | -                            |

Example:

```
/* DMA0 channel0 mode configuration */

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 3-213. 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..6)</i> | DMA channel selection   |
| <b>Output parameter{out}</b>              |   |
| -   | -   |
| <b>Return value</b>                       |   |
| -   | -   |

Example:

---

```

/* DMA0 channel0 mode configuration */

dma_memory_to_memory_enable(DMA0, DMA_CH0);

```

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

The description of `dma_channel_enable` is shown as below:

**Table 3-214. 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..6)</i> | DMA channel selection  |
| <b>Output parameter{out}</b>              |  |
| -   | -  |
| <b>Return value</b>                       |  |
| -   | -  |

Example:

```

/* enable DMA0 channel0 transfer */

dma_channel_enable(DMA0, DMA_CH0);

```

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

The description of `dma_channel_disable` is shown as below:

**Table 3-215. 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  |

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

Example:

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

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

The description of `dma_periph_address_config` is shown as below:

**Table 3-216. 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,<br>dma_channel_enum channelx, uint32_t address); |
| <b>Function descriptions</b>              | set DMA peripheral base address  |
| <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..6)</i> | DMA channel selection  |
| Input parameter{in}                       |  |
| <b>address</b>                            | peripheral base address  |
| Output parameter{out}                     |  |
| -   | -  |
| Return value                              |  |
| -   | -  |

Example:

```
/* configure DMA0 channel0 address */
#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 3-217. 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..6)</i> | DMA channel selection  |
| <b>Input parameter{in}</b>                |  |
| <b>address</b>                            | memory base address  |
| <b>Output parameter{out}</b>              |  |
| -   | -  |
| <b>Return value</b>                       |  |
| -   | -  |

Example:

```
/* configure DMA0 channel0 memory address */
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 3-218. 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..6)</i> | DMA channel selection    |
| <b>Input parameter{in}</b>                |                          |
| <b>number</b>                             | data transfer number     |
| <i>0x0000-0xffff</i>                      | number                   |
| <b>Output parameter{out}</b>              |                          |
| -   | -                        |
| <b>Return value</b>                       |                          |
| -   | -                        |

Example:

```
/* configure DMA0 channel0 transfer number */

#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 3-219. Function `dma_transfer_number_get`**

|   |   |
|---|---|
| <b>Function name</b>                      | <code>dma_transfer_number_get</code>  |
| <b>Function prototype</b>                 | <code>uint32_t dma_transfer_number_get(uint32_t dma_periph,<br/>                                     dma_channel_enum channelx);</code> |
| <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..6)</i> | DMA channel selection   |
| <b>Output parameter{out}</b>              |   |
| -   | -   |
| <b>Return value</b>                       |   |
| <b>uint32_t</b>                           | <i>0x0000-0xffff</i>  |

Example:

```
/* get DMA0 channel0 transfer number */
```

```

uint32_t number = 0;
number = dma_transfer_number_get(DMA0, DMA_CH0);
  
```

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

The description of `dma_priority_config` is shown as below:

**Table 3-220. 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..6)</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:

```

/* configure DMA0 channel0 priority */
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 3-221. 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..6)</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:

```
/* configure DMA0 channel0 memory width */
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 3-222. 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  |

| Input parameter{in}                       |   |
|---|---|
| <b>channelx</b>                           | DMA channel                                 |
| <i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..6)</i> | DMA channel selection                       |
| Input parameter{in}                       |   |
| <b>pwidth</b>                             | transfer data width of peripheral           |
| <i>DMA_PERIPHERAL_WIDTH_8BIT</i>          | transfer data width of peripheral is 8-bit  |
| <i>DMA_PERIPHERAL_WIDTH_16BIT</i>         | transfer data width of peripheral is 16-bit |
| <i>DMA_PERIPHERAL_WIDTH_32BIT</i>         | transfer data width of peripheral is 32-bit |
| Output parameter{out}                     |   |
| -   | -   |
| Return value                              |   |
| -   | -   |

Example:

```
/* configure DMA0 channel0 peripheral width */
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 3-223. 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>               | -   |
| 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..6)</i> | DMA channel selection   |
| Output parameter{out}                     |   |
| -   | -   |
| Return value                              |   |
| -   | -   |

Example:

```
/* enable DMA0 channel0 memory increase */

dma_memory_increase_enable(DMA0, DMA_CH0);
```

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

The description of `dma_memory_increase_disable` is shown as below:

**Table 3-224. 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   |
| <b>DMAx(x=0,1)</b>                        | DMA peripheral selection   |
| <b>Input parameter{in}</b>                |  |
| <b>channelx</b>                           | DMA channel  |
| <b>DMA_CHx(DMA0:x=0..6; DMA1: x=0..6)</b> | DMA channel selection  |
| <b>Output parameter{out}</b>              |  |
| -   | -  |
| <b>Return value</b>                       |  |
| -   | -  |

Example:

```
/* disable DMA0 channel0 memory increase */

dma_memory_increase_disable(DMA0, DMA_CH0);
```

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

The description of `dma_periph_increase_enable` is shown as below:

**Table 3-225. 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..6)</i> | DMA channel selection    |
| <b>Output parameter{out}</b>              |                          |
| -   | -                        |
| <b>Return value</b>                       |                          |
| -   | -                        |

Example:

```
/* enable DMA0 channel0 peripheral increase */

dma_periph_increase_enable(DMA0, DMA_CH0);
```

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

The description of `dma_periph_increase_disable` is shown as below:

**Table 3-226. Function `dma_periph_increase_disable`**

|   |  |
|---|--|
| <b>Function name</b>                      | <code>dma_periph_increase_disable</code>   |
| <b>Function prototype</b>                 | <code>void dma_periph_increase_disable(uint32_t dma_periph,<br/>dma_channel_enum channelx);</code> |
| <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..6)</i> | DMA channel selection  |
| <b>Output parameter{out}</b>              |  |
| -   | -  |
| <b>Return value</b>                       |  |
| -   | -  |

Example:

```
/* disable DMA0 channel0 peripheral increase */

dma_periph_increase_disable(DMA0, DMA_CH0);
```

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

The description of `dma_transfer_direction_config` is shown as below:

**Table 3-227. 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..6)</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:

```
/* config DMA0 channel0 peripheral to memory */
dma_transfer_direction_config(DMA0, DMA_CH0, DMA_PERIPHERAL_TO_MEMORY);
```

### **dma\_1\_channel\_5\_fulldata\_transfer\_enable**

The description of `dma_1_channel_5_fulldata_transfer_enable` is shown as below:

**Table 3-228. Function `dma_1_channel_5_fulldata_transfer_enable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | dma_1_channel_5_fulldata_transfer_enable             |
| <b>Function prototype</b>    | void dma_1_channel_5_fulldata_transfer_enable(void); |
| <b>Function descriptions</b> | enable the DMA1 channel 5 Full_Data transfer mode    |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |

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

Example:

```
/* enable the DMA1 channel 5 Full_Data transfer mode */

dma_1_channel_5_fulldata_transfer_enable();
```

### **dma\_1\_channel\_5\_fulldata\_transfer\_disable**

The description of dma\_1\_channel\_5\_fulldata\_transfer\_disable is shown as below:

**Table 3-229. Function dma\_1\_channel\_5\_fulldata\_transfer\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | dma_1_channel_5_fulldata_transfer_disable             |
| <b>Function prototype</b>    | void dma_1_channel_5_fulldata_transfer_disable(void); |
| <b>Function descriptions</b> | disable the DMA1 channel 5 Full_Data transfer mode    |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable the DMA1 channel 5 Full_Data transfer mode */

dma_1_channel_5_fulldata_transfer_disable();
```

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

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

**Table 3-230. 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..6)</i> | DMA channel selection                |
| <b>Input parameter{in}</b>                |                                      |
| <b>flag</b>                               | specify get which flag               |
| <i>DMA_FLAG_G</i>                         | global interrupt flag of channel     |
| <i>DMA_FLAG_FTF</i>                       | full transfer finish flag of channel |
| <i>DMA_FLAG_HTF</i>                       | half transfer finish flag of channel |
| <i>DMA_FLAG_ERR</i>                       | error flag of channel                |
| <b>Output parameter{out}</b>              |                                      |
| -   | -                                    |
| <b>Return value</b>                       |                                      |
| <b>FlagStatus</b>                         | SET or RESET                         |

Example:

```
/* get DMA0 channel0 flag */
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 3-231. Function `dma_flag_clear`**

|   |  |
|---|--|
| <b>Function name</b>                      | <code>dma_flag_clear</code>  |
| <b>Function prototype</b>                 | <code>void dma_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);</code> |
| <b>Function descriptions</b>              | clear DMA a channel flag   |
| <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..6)</i> | DMA channel selection  |
| <b>Input parameter{in}</b>                |  |
| <b>flag</b>                               | specify get which flag   |
| <i>DMA_FLAG_G</i>                         | global interrupt flag of channel   |
| <i>DMA_FLAG_FTF</i>                       | full transfer finish flag of channel   |

|                              |                                      |
|------------------------------|--------------------------------------|
| DMA_FLAG_HTF                 | half transfer finish flag of channel |
| DMA_FLAG_ERR                 | error flag of channel                |
| <b>Output parameter{out}</b> |                                      |
| -                            | -                                    |
| <b>Return value</b>          |                                      |
| -                            | -                                    |

Example:

```
/* clear DMA0 channel0 flag */

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

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

The description of `dma_interrupt_enable` is shown as below:

**Table 3-232. 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>               | -   |
| <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..6)</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_enable(DMA0, DMA_CH0, DMA_INT_FTF);
```

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

The description of `dma_interrupt_disable` is shown as below:

**Table 3-233. Function `dma_interrupt_disable`**

|   |   |
|---|---|
| <b>Function name</b>                      | dma_interrupt_disable   |
| <b>Function prototype</b>                 | <code>void dma_interrupt_disable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);</code> |
| <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..6)</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);
```

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

The description of `dma_interrupt_flag_get` is shown as below:

**Table 3-234. Function `dma_interrupt_flag_get`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | dma_interrupt_flag_get   |
| <b>Function prototype</b>    | <code>FlagStatus dma_interrupt_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);</code> |
| <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..6)</i> | DMA channel selection                          |
| <b>Input parameter{in}</b>                |  |
| <b>flag</b>                               | specify get which flag                         |
| <i>DMA_INT_FLAG_G</i>                     | global interrupt flag of channel               |
| <i>DMA_INT_FLAG_FTF</i>                   | full transfer finish interrupt flag of channel |
| <i>DMA_INT_FLAG_HTF</i>                   | half transfer finish interrupt flag of channel |
| <i>DMA_INT_FLAG_ERR</i>                   | error interrupt flag of channel                |
| <b>Output parameter{out}</b>              |  |
| -   | -  |
| <b>Return value</b>                       |  |
| <b>FlagStatus</b>                         | SET or RESET                                   |

Example:

```
/* get DMA0 channel3 interrupt flag */
FlagStatus int_flag_status = RESET;
int_flag_status = dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_G);
```

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

The description of `dma_interrupt_flag_clear` is shown as below:

**Table 3-235. 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 DMA a channel flag  |
| <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..6)</i> | DMA channel selection   |
| <b>Input parameter{in}</b>                |   |
| <b>flag</b>                               | specify get which flag  |
| <i>DMA_INT_FLAG_G</i>                     | global interrupt flag of channel  |

|                              |  |
|------------------------------|--|
| DMA_INT_FLAG_FTF             | full transfer finish interrupt flag of channel |
| DMA_INT_FLAG_HTF             | half transfer finish interrupt flag of channel |
| DMA_INT_FLAG_ERR             | error interrupt flag of channel                |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* clear DMA0 channel3 interrupt flag */

dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_G);
```

## 3.11. ENET

There is a media access controller (MAC) designed in Ethernet module to support 10/100Mbps interface speed. For more efficient data transfer between Ethernet and memory, a DMA controller is designed in this module. The support interface protocol for Ethernet is media independent interface (MII) and reduced media independent interface (RMII). The ENET registers are listed in chapter [3.11.1](#), the ENET firmware functions are introduced in chapter [3.11.2](#).

### 3.11.1. Descriptions of Peripheral registers

ENET registers are listed in the table shown as below:

**Table 3-236. ENET Registers**

| Registers         | Descriptions                            |
|-------------------|---|
| ENET_MAC_CFG      | MAC configuration register              |
| ENET_MAC_FRMF     | MAC frame filter register               |
| ENET_MAC_HLH      | MAC hash list high register             |
| ENET_MAC_HLL      | MAC hash list low register              |
| ENET_MAC_PHY_CTL  | MAC PHY control register                |
| ENET_MAC_PHY_DATA | MAC PHY data register                   |
| ENET_MAC_FCTL     | MAC flow control register               |
| ENET_MAC_FCTH     | MAC flow control threshold register     |
| ENET_MAC_VLT      | MAC VLAN tag register                   |
| ENET_MAC_RWFF     | MAC remote wakeup frame filter register |
| ENET_MAC_WUM      | MAC wakeup management register          |
| ENET_MAC_INTF     | MAC interrupt flag register             |
| ENET_MAC_INTMS    | MAC interrupt mask register             |

| Registers            | Descriptions  |
|----------------------|---|
| K                    |   |
| ENET_MAC_ADDR<br>0H  | MAC address 0 high register   |
| ENET_MAC_ADDR<br>0L  | MAC address 0 low register  |
| ENET_MAC_ADDR<br>1H  | MAC address 1 high register   |
| ENET_MAC_ADDR<br>1L  | MAC address 1 low register  |
| ENET_MAC_ADDT<br>2H  | MAC address 2 high register   |
| ENET_MAC_ADDR<br>2L  | MAC address 2 low register  |
| ENET_MAC_ADDR<br>3H  | MAC address 3 high register   |
| ENET_MAC_ADDR<br>3L  | MAC address 3 low register  |
| ENET_MSC_CTL         | MSC control register  |
| ENET_MSC_RINTF       | MSC receive interrupt flag register   |
| ENET_MSC_TINTF       | MSC transmit interrupt flag register  |
| ENET_MSC_RINT<br>MSK | MSC receive interrupt mask register   |
| ENET_MSC_TINTM<br>SK | MSC transmit interrupt mask register  |
| ENET_MSC_SCCN<br>T   | MSC transmitted good frames after a single collision counter register           |
| ENET_MSC_MSCC<br>NT  | MSC transmitted good frames after more than a single collision counter register |
| ENET_MSC_TGFC<br>NT  | MSC transmitted good frames counter register                                    |
| ENET_MSC_RFCE<br>CNT | MSC received frames with CRC error counter register                             |
| ENET_MSC_RFAE<br>CNT | MSC received frames with alignment error counter register                       |
| ENET_MSC_RGUF<br>CNT | MSC received good unicast frames counter register                               |
| ENET_PTP_TSCTL       | PTP time stamp control register   |
| ENET_PTP_SSINC       | PTP subsecond increment register  |
| ENET_PTP_TSH         | PTP time stamp high register  |
| ENET_PTP_TSL         | PTP time stamp low register   |
| ENET_PTP_TSUH        | PTP time stamp update high register   |
| ENET_PTP_TSUL        | PTP time stamp update low register  |

| Registers             | Descriptions  |
|-----------------------|---|
| ENET_PTP_TSADD<br>END | PTP time stamp addend register                        |
| ENET_PTP_ETH          | PTP expected time high register                       |
| ENET_PTP_ETL          | PTP expected time low register                        |
| ENET_DMA_BCTL         | DMA bus control register                              |
| ENET_DMA_TPEN         | DMA transmit poll enable register                     |
| ENET_DMA_RPEN         | DMA receive poll enable register                      |
| ENET_DMA_RDTA<br>DDR  | DMA receive descriptor table address register         |
| ENET_DMA_TDTA<br>DDR  | DMA transmit descriptor table address register        |
| ENET_DMA_STAT         | DMA status register                                   |
| ENET_DMA_CTL          | DMA control register                                  |
| ENET_DMA_INTEN        | DMA interrupt enable register                         |
| ENET_DMA_MFBO<br>CNT  | DMA missed frame and buffer overflow counter register |
| ENET_DMA_CTDA<br>DDR  | DMA current transmit descriptor address register      |
| ENET_DMA_CRDA<br>DDR  | DMA current receive descriptor address register       |
| ENET_DMA_CTBA<br>DDR  | DMA current transmit buffer address register          |
| ENET_DMA_CRBA<br>DDR  | DMA current receive buffer address register           |

### 3.11.2. Descriptions of Peripheral functions

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

**Table 3-237. ENET firmware function**

| Function name               | Function description   |
|-----------------------------|--|
| main function               |  |
| enet_deinit                 | deinitialize the ENET, and reset structure parameters for ENET initialization                |
| enet_initpara_config        | configure the parameters which are usually less cared for initialization                     |
| enet_init                   | initialize ENET peripheral with generally concerned parameters and the less cared parameters |
| enet_software_reset         | reset all core internal registers located in CLK_TX and CLK_RX                               |
| enet_rxframe_size_get       | check receive frame valid and return frame size  |
| enet_descriptors_chain_init | initialize the dma tx/rx descriptors's parameters in chain mode                              |

| <b>Function name</b>              | <b>Function description</b>   |
|-----------------------------------|---|
| enet_descriptors_ring_init        | initialize the dma tx/rx descriptors's parameters in ring mode                          |
| enet_frame_receive                | handle current received frame data to application buffer                                |
| enet_frame_transmit               | handle application buffer data to transmit it   |
| enet_transmit_checksum_config     | configure the transmit IP frame checksum offload calculation and insertion              |
| enet_enable                       | ENET Tx and Rx function enable (include MAC and DMA module)                             |
| enet_disable                      | ENET Tx and Rx function disable (include MAC and DMA module)                            |
| enet_mac_address_set              | configure MAC address   |
| enet_mac_address_get              | get MAC address   |
| enet_flag_get                     | get the ENET MAC/MSC/PTP/DMA status flag  |
| enet_flag_clear                   | clear the ENET DMA status flag  |
| enet_interrupt_enable             | enable ENET MAC/MSC/DMA interrupt   |
| enet_interrupt_disable            | disable ENET MAC/MSC/DMA interrupt  |
| enet_interrupt_flag_get           | get ENET MAC/MSC/DMA interrupt flag   |
| enet_interrupt_flag_clear         | clear ENET DMA interrupt flag   |
| MAC function                      |   |
| enet_tx_enable                    | ENET Tx function enable (include MAC and DMA module)                                    |
| enet_tx_disable                   | ENET Tx function disable (include MAC and DMA module)                                   |
| enet_rx_enable                    | ENET Rx function enable (include MAC and DMA module)                                    |
| enet_rx_disable                   | ENET Rx function disable (include MAC and DMA module)                                   |
| enet_registers_get                | put registers value into the application buffer   |
| enet_address_filter_enable        | enable the MAC address filter   |
| enet_address_filter_disable       | disable the MAC address filter  |
| enet_address_filter_config        | configure the MAC address filter  |
| enet_phy_config                   | PHY interface configuration (configure SMI clock and reset PHY chip)                    |
| enet_phy_write_read               | write to/read from a PHY register   |
| enet_phyloopback_enable           | enable the loopback function of phy chip  |
| enet_phyloopback_disable          | disable the loopback function of phy chip   |
| enet_forward_feature_enable       | enable ENET forward feature   |
| enet_forward_feature_disable      | disable ENET forward feature  |
| enet_fliter_feature_enable        | enable ENET fliter feature  |
| enet_fliter_feature_disable       | disable ENET fliter feature   |
| flow control function             |   |
| enet_pauseframe_generate          | generate the pause frame, ENET will send pause frame after enable transmit flow control |
| enet_pauseframe_detect_config     | configure the pause frame detect type   |
| enet_pauseframe_config            | configure the pause frame parameters  |
| enet_flowcontrol_threshold_config | configure the threshold of the flow control(deactive and active)                        |

| Function name                          | Function description   |
|--|--|
|  | threshold)   |
| enet_flowcontrol_feature_enable        | enable ENET flow control feature   |
| enet_flowcontrol_feature_disable       | disable ENET flow control feature  |
|  | DMA function   |
| enet_dmaprocess_state_get              | get the dma transmit/receive process state   |
| enet_dmaprocess_resume                 | poll the dma transmission/reception enable   |
| enet_rxprocess_check_recovery          | check and recover the Rx process   |
| enet_txfifo_flush                      | flush the ENET transmit fifo, and wait until the flush operation completes                                     |
| enet_current_desc_address_get          | get the transmit/receive address of current descriptor, or current buffer, or descriptor table                 |
| enet_desc_information_get              | get the Tx or Rx descriptor information  |
| enet_missed_frame_counter_get          | get the number of missed frames during receiving   |
|  | descriptor function  |
| enet_desc_flag_get                     | get the bit flag of ENET dma descriptor  |
| enet_desc_flag_set                     | set the bit flag of ENET dma tx descriptor   |
| enet_desc_flag_clear                   | clear the bit flag of ENET dma tx descriptor   |
| enet_desc_receive_complete_bit_enable  | when receiving the completed, set RS bit in ENET_DMA_STAT register will immediately set                        |
| enet_desc_receive_complete_bit_disable | when receiving the completed, set RS bit in ENET_DMA_STAT register will be set after a configurable delay time |
| enet_rxframe_drop                      | drop current receive frame   |
| enet_dma_feature_enable                | enable DMA feature   |
| enet_dma_feature_disable               | disable DMA feature  |
| enet_ptp_normal_descriptors_chain_init | initialize the dma Tx/Rx descriptors's parameters in normal chain mode with ptp function                       |
| enet_ptp_normal_descriptors_ring_init  | initialize the dma Tx/Rx descriptors's parameters in normal ring mode with ptp function                        |
| enet_ptpframe_receive_normal_mode      | receive a packet data with timestamp values to application buffer, when the DMA is in normal mode              |
| enet_ptpframe_transmit_normal_mode     | send data with timestamp values in application buffer as a transmit packet, when the DMA is in normal mode     |
|  | WUM function   |
| enet_wum_filter_register_pointer_reset | wakeup frame filter register pointer reset   |
| enet_wum_filter_config                 | set the remote wakeup frame registers  |
| enet_wum_feature_enable                | enable wakeup management features  |
| enet_wum_feature_disable               | disable wakeup management features   |
|  | MSC function   |
| enet_msc_counters_reset                | reset the MAC statistics counters  |
| enet_msc_feature_enable                | enable the MAC statistics counter features   |

| Function name                            | Function description  |
|--|---|
| enet_msc_feature_disable                 | disable the MAC statistics counter features                                 |
| enet_msc_counters_get                    | get MAC statistics counter  |
| PTP function                             |   |
| enet_ptp_subsecond_2_nanosecond          | change subsecond to nanosecond  |
| enet_ptp_nanosecond_2_subsecond          | change nanosecond to subsecond  |
| enet_ptp_feature_enable                  | enable the PTP features   |
| enet_ptp_feature_disable                 | disable the PTP features  |
| enet_ptp_timestamp_function_config       | configure the PTP timestamp function  |
| enet_ptp_subsecond_increment_config      | configure the PTP system time subsecond increment value                     |
| enet_ptp_timestamp_addend_config         | adjusting the PTP clock frequency only in fine update mode                  |
| enet_ptp_timestamp_update_config         | initializing or adding/subtracting to second of the PTP system time         |
| enet_ptp_expected_time_config            | configure the PTP expected target time                                      |
| enet_ptp_system_time_get                 | get the PTP current system time   |
| enet_ptp_start                           | configure and start PTP timestamp counter                                   |
| enet_ptp_finecorrection_adjfreq          | adjust frequency in fine method by configure addend register                |
| enet_ptp_coarsecorrection_systime_update | update system time in coarse method   |
| enet_ptp_finecorrection_settime          | set system time in fine method  |
| enet_ptp_flag_get                        | get the ptpt flag status  |
| Other                                    |   |
| enet_initpara_reset                      | reset the ENET initpara struct, call it before using enet_initpara_config() |

### Structure enet\_descriptors\_struct

Table 3-238. Structure enet\_descriptors\_struct

| member name            | Function description                                      |
|------------------------|---|
| status                 | status  |
| control_buffer_size    | control and buffer1, buffer2 lengths                      |
| buffer1_addr           | buffer1 address pointer/timestamp low                     |
| buffer2_next_desc_addr | buffer2 or next descriptor address pointer/timestamp high |

### Structure enet\_ptp\_systime\_struct

Table 3-239. Structure enet\_ptp\_systime\_struct

| member name | Function description      |
|-------------|---------------------------|
| second      | second of system time     |
| nanosecond  | nanosecond of system time |
| sign        | sign of system time       |

**Enum enet\_flag\_enum**
**Table 3-240. Enum enet\_flag\_enum**

| member name               | Function description                               |
|---------------------------|--|
| ENET_MAC_FLAG_MPKR        | magic packet received flag                         |
| ENET_MAC_FLAG_WUFR        | wakeup frame received flag                         |
| ENET_MAC_FLAG_FLOWCONTROL | flow control status flag                           |
| ENET_MAC_FLAG_WUM         | WUM status flag                                    |
| ENET_MAC_FLAG_MSC         | MSC status flag                                    |
| ENET_MAC_FLAG_MSCR        | MSC receive status flag                            |
| ENET_MAC_FLAG_MSCT        | MSC transmit status flag                           |
| ENET_MAC_FLAG_TMST        | timestamp trigger status flag                      |
| ENET_PTP_FLAG_TSSCO       | timestamp second counter overflow flag             |
| ENET_PTP_FLAG_TTM         | target time match flag                             |
| ENET_MSC_FLAG_RFCE        | received frames CRC error flag                     |
| ENET_MSC_FLAG_RFAE        | received frames alignment error flag               |
| ENET_MSC_FLAG_RGUF        | received good unicast frames flag                  |
| ENET_MSC_FLAG_TGFSC       | transmitted good frames single collision flag      |
| ENET_MSC_FLAG_TGFMSC      | transmitted good frames more single collision flag |
| ENET_MSC_FLAG_TGF         | transmitted good frames flag                       |
| ENET_DMA_FLAG_TS          | transmit status flag                               |
| ENET_DMA_FLAG_TPS         | transmit process stopped status flag               |
| ENET_DMA_FLAG_TBU         | transmit buffer unavailable status flag            |
| ENET_DMA_FLAG_JT          | transmit jabber timeout status flag                |

| member name                      | Function description                            |
|----------------------------------|---|
| TJT                              |   |
| ENET_DMA_FLAG_RO                 | receive overflow status flag                    |
| ENET_DMA_FLAG_TU                 | transmit underflow status flag                  |
| ENET_DMA_FLAG_RS                 | receive status flag                             |
| ENET_DMA_FLAG_RBU                | receive buffer unavailable status flag          |
| ENET_DMA_FLAG_RPS                | receive process stopped status flag             |
| ENET_DMA_FLAG_RWT                | receive watchdog timeout status flag            |
| ENET_DMA_FLAG_ET                 | early transmit status flag                      |
| ENET_DMA_FLAG_FBE                | fatal bus error status flag                     |
| ENET_DMA_FLAG_ER                 | early receive status flag                       |
| ENET_DMA_FLAG_AI                 | abnormal interrupt summary flag                 |
| ENET_DMA_FLAG_NI                 | normal interrupt summary flag                   |
| ENET_DMA_FLAG_EB_DMA_ERROR       | error during data transfer by RxDMA/TxDMA flag  |
| ENET_DMA_FLAG_EB_TRANSFER_E_RROR | error during write/read transfer flag           |
| ENET_DMA_FLAG_EB_ACCESS_ERR_OR   | error during data buffer/descriptor access flag |
| ENET_DMA_FLAG_MSC                | MSC status flag                                 |
| ENET_DMA_FLAG_WUM                | WUM status flag                                 |
| ENET_DMA_FLAG_TST                | timestamp trigger status flag                   |

**Enum enet\_flag\_clear\_enum**
**Table 3-241. Enum enet\_flag\_clear\_enum**

| member name           | Function description                          |
|-----------------------|---|
| ENET_DMA_FLAG_TS_CLR  | transmit status flag clear                    |
| ENET_DMA_FLAG_TPS_CLR | transmit process stopped status flag clear    |
| ENET_DMA_FLAG_TBU_CLR | transmit buffer unavailable status flag clear |
| ENET_DMA_FLAG_TJT_CLR | transmit jabber timeout status flag clear     |
| ENET_DMA_FLAG_RO_CLR  | receive overflow status flag clear            |
| ENET_DMA_FLAG_TU_CLR  | transmit underflow status flag clear          |
| ENET_DMA_FLAG_RS_CLR  | receive status flag clear                     |
| ENET_DMA_FLAG_RBU_CLR | receive buffer unavailable status flag clear  |
| ENET_DMA_FLAG_RPS_CLR | receive process stopped status flag clear     |
| ENET_DMA_FLAG_RWT_CLR | receive watchdog timeout status flag clear    |
| ENET_DMA_FLAG_ET_CLR  | early transmit status flag clear              |
| ENET_DMA_FLAG_FBE_CLR | fatal bus error status flag clear             |
| ENET_DMA_FLAG_ER_CLR  | early receive status flag clear               |
| ENET_DMA_FLAG_AI_CLR  | abnormal interrupt summary flag clear         |
| ENET_DMA_FLAG_NI_CLR  | normal interrupt summary flag clear           |

**Enum enet\_int\_enum**
**Table 3-242. Enum enet\_int\_enum**

| member name         | Function description                    |
|---------------------|---|
| ENET_MAC_INT_WUMIM  | WUM interrupt mask                      |
| ENET_MAC_INT_TMSTIM | timestamp trigger interrupt mask        |
| ENET_MSC_INT_R      | received frame CRC error interrupt mask |

| <b>member name</b>            | <b>Function description</b>                                  |
|-------------------------------|--|
| <i>FCEIM</i>                  |  |
| <i>ENET_MSC_INT_R_FAEIM</i>   | received frames alignment error interrupt mask               |
| <i>ENET_MSC_INT_R_GUFIM</i>   | received good unicast frames interrupt mask                  |
| <i>ENET_MSC_INT_T_GFSCIM</i>  | transmitted good frames single collision interrupt mask      |
| <i>ENET_MSC_INT_T_GFMSCIM</i> | transmitted good frames more single collision interrupt mask |
| <i>ENET_MSC_INT_T_GFIM</i>    | transmitted good frames interrupt mask                       |
| <i>ENET_DMA_INT_TI_E</i>      | transmit interrupt enable                                    |
| <i>ENET_DMA_INT_T_PSIE</i>    | transmit process stopped interrupt enable                    |
| <i>ENET_DMA_INT_T_BUIE</i>    | transmit buffer unavailable interrupt enable                 |
| <i>ENET_DMA_INT_T_JTIE</i>    | transmit jabber timeout interrupt enable                     |
| <i>ENET_DMA_INT_R_OIE</i>     | receive overflow interrupt enable                            |
| <i>ENET_DMA_INT_T_UIE</i>     | transmit underflow interrupt enable                          |
| <i>ENET_DMA_INT_RI_E</i>      | receive interrupt enable                                     |
| <i>ENET_DMA_INT_R_BUIE</i>    | receive buffer unavailable interrupt enable                  |
| <i>ENET_DMA_INT_R_PSIE</i>    | receive process stopped interrupt enable                     |
| <i>ENET_DMA_INT_R_WTIE</i>    | receive watchdog timeout interrupt enable                    |
| <i>ENET_DMA_INT_E_TIE</i>     | early transmit interrupt enable                              |
| <i>ENET_DMA_INT_F_BEIE</i>    | fatal bus error interrupt enable                             |
| <i>ENET_DMA_INT_E_RIE</i>     | early receive interrupt enable                               |
| <i>ENET_DMA_INT_AI_E</i>      | abnormal interrupt summary enable                            |
| <i>ENET_DMA_INT_NI_E</i>      | normal interrupt summary enable                              |

**Enum enet\_int\_flag\_enum**
**Table 3-243. Enum enet\_int\_flag\_enum**

| member name                      | Function description                               |
|----------------------------------|--|
| <i>ENET_MAC_INT_F_LAG_WUM</i>    | WUM status flag                                    |
| <i>ENET_MAC_INT_F_LAG_MSC</i>    | MSC status flag                                    |
| <i>ENET_MAC_INT_F_LAG_MSCR</i>   | MSC receive status flag                            |
| <i>ENET_MAC_INT_F_LAG_MSCT</i>   | MSC transmit status flag                           |
| <i>ENET_MAC_INT_F_LAG_TMST</i>   | time stamp trigger status flag                     |
| <i>ENET_MSC_INT_F_LAG_RFCE</i>   | received frames CRC error flag                     |
| <i>ENET_MSC_INT_F_LAG_RFAE</i>   | received frames alignment error flag               |
| <i>ENET_MSC_INT_F_LAG_RGUF</i>   | received good unicast frames flag                  |
| <i>ENET_MSC_INT_F_LAG_TGFSC</i>  | transmitted good frames single collision flag      |
| <i>ENET_MSC_INT_F_LAG_TGFMSC</i> | transmitted good frames more single collision flag |
| <i>ENET_MSC_INT_F_LAG_TGF</i>    | transmitted good frames flag                       |
| <i>ENET_DMA_INT_F_LAG_TS</i>     | transmit status flag                               |
| <i>ENET_DMA_INT_F_LAG_TPS</i>    | transmit process stopped status flag               |
| <i>ENET_DMA_INT_F_LAG_TBU</i>    | transmit buffer unavailable status flag            |
| <i>ENET_DMA_INT_F_LAG_TJT</i>    | transmit jabber timeout status flag                |
| <i>ENET_DMA_INT_F_LAG_RO</i>     | receive overflow status flag                       |
| <i>ENET_DMA_INT_F_LAG_TU</i>     | transmit underflow status flag                     |
| <i>ENET_DMA_INT_F_LAG_RS</i>     | receive status flag                                |
| <i>ENET_DMA_INT_F_LAG_RBU</i>    | receive buffer unavailable status flag             |
| <i>ENET_DMA_INT_F</i>            | receive process stopped status flag                |

| member name                             | Function description                 |
|---|--------------------------------------|
| <i>LAG_RPS</i>                          |                                      |
| <i>ENET_DMA_INT_F</i><br><i>LAG_RWT</i> | receive watchdog timeout status flag |
| <i>ENET_DMA_INT_F</i><br><i>LAG_ET</i>  | early transmit status flag           |
| <i>ENET_DMA_INT_F</i><br><i>LAG_FBE</i> | fatal bus error status flag          |
| <i>ENET_DMA_INT_F</i><br><i>LAG_ER</i>  | early receive status flag            |
| <i>ENET_DMA_INT_F</i><br><i>LAG_AI</i>  | abnormal interrupt summary flag      |
| <i>ENET_DMA_INT_F</i><br><i>LAG_NI</i>  | normal interrupt summary flag        |
| <i>ENET_DMA_INT_F</i><br><i>LAG_MSC</i> | MSC status flag                      |
| <i>ENET_DMA_INT_F</i><br><i>LAG_WUM</i> | WUM status flag                      |
| <i>ENET_DMA_INT_F</i><br><i>LAG_TST</i> | timestamp trigger status flag        |

### Enum enet\_int\_flag\_clear\_enum

Table 3-244. Enum enet\_int\_flag\_clear\_enum

| member name                                 | Function description                    |
|---|---|
| <i>ENET_DMA_INT_F</i><br><i>LAG_TS_CLR</i>  | transmit status flag                    |
| <i>ENET_DMA_INT_F</i><br><i>LAG_TPS_CLR</i> | transmit process stopped status flag    |
| <i>ENET_DMA_INT_F</i><br><i>LAG_TBU_CLR</i> | transmit buffer unavailable status flag |
| <i>ENET_DMA_INT_F</i><br><i>LAG_TJT_CLR</i> | transmit jabber timeout status flag     |
| <i>ENET_DMA_INT_F</i><br><i>LAG_RO_CLR</i>  | receive overflow status flag            |
| <i>ENET_DMA_INT_F</i><br><i>LAG_TU_CLR</i>  | transmit underflow status flag          |
| <i>ENET_DMA_INT_F</i><br><i>LAG_RS_CLR</i>  | receive status flag                     |
| <i>ENET_DMA_INT_F</i><br><i>LAG_RBU_CLR</i> | receive buffer unavailable status flag  |
| <i>ENET_DMA_INT_F</i><br><i>LAG_RPS_CLR</i> | receive process stopped status flag     |

| member name                                 | Function description                 |
|---|--------------------------------------|
| <i>ENET_DMA_INT_F</i><br><i>LAG_RWT_CLR</i> | receive watchdog timeout status flag |
| <i>ENET_DMA_INT_F</i><br><i>LAG_ET_CLR</i>  | early transmit status flag           |
| <i>ENET_DMA_INT_F</i><br><i>LAG_FBE_CLR</i> | fatal bus error status flag          |
| <i>ENET_DMA_INT_F</i><br><i>LAG_ER_CLR</i>  | early receive status flag            |
| <i>ENET_DMA_INT_F</i><br><i>LAG_AI_CLR</i>  | abnormal interrupt summary flag      |
| <i>ENET_DMA_INT_F</i><br><i>LAG_NI_CLR</i>  | normal interrupt summary flag        |

### Enum enet\_desc\_reg\_enum

Table 3-245. Enum enet\_desc\_reg\_enum

| member name                              | Function description   |
|--|--|
| <i>ENET_RX_DESC_T</i><br><i>ABLE</i>     | the start address of the receive descriptor table  |
| <i>ENET_RX_CURRE</i><br><i>NT_DESC</i>   | the start descriptor address of the current receive descriptor read by the RxDMA controller  |
| <i>ENET_RX_CURRE</i><br><i>NT_BUFFER</i> | the current receive buffer address being read by the RxDMA controller                        |
| <i>ENET_TX_DESC_T</i><br><i>ABLE</i>     | the start address of the transmit descriptor table   |
| <i>ENET_TX_CURRE</i><br><i>NT_DESC</i>   | the start descriptor address of the current transmit descriptor read by the TxDMA controller |
| <i>ENET_TX_CURRE</i><br><i>NT_BUFFER</i> | the current transmit buffer address being read by the TxDMA controller                       |

### Enum enet\_msc\_counter\_enum

Table 3-246. Enum enet\_msc\_counter\_enum

| member name                           | Function description   |
|---------------------------------------|--|
| <i>ENET_MSC_TX_S</i><br><i>CCNT</i>   | MSC transmitted good frames after a single collision counter           |
| <i>ENET_MSC_TX_M</i><br><i>SCCNT</i>  | MSC transmitted good frames after more than a single collision counter |
| <i>ENET_MSC_TX_T</i><br><i>GFCNT</i>  | MSC transmitted good frames counter                                    |
| <i>ENET_MSC_RX_R</i><br><i>FCECNT</i> | MSC received frames with CRC error counter                             |
| <i>ENET_MSC_RX_R</i>                  | MSC received frames with alignment error counter                       |

| member name                           | Function description                     |
|---------------------------------------|--|
| <i>FAECNT</i>                         |  |
| <i>ENET_MSC_RX_R</i><br><i>GUFCNT</i> | MSC received good unicast frames counter |

### Enum enet\_option\_enum

Table 3-247. Enum enet\_option\_enum

| member name                   | Function description  |
|-------------------------------|---|
| <i>FORWARD_OPTION</i>         | choose to configure the frame forward related parameters      |
| <i>DMABUS_OPTION</i>          | choose to configure the DMA bus mode related parameters       |
| <i>DMA_MAXBURST_OPTION</i>    | choose to configure the DMA max burst related parameters      |
| <i>DMA_ARBITRATION_OPTION</i> | choose to configure the DMA arbitration related parameters    |
| <i>STORE_OPTION</i>           | choose to configure the store forward mode related parameters |
| <i>DMA_OPTION</i>             | choose to configure the DMA related parameters                |
| <i>VLAN_OPTION</i>            | choose to configure vlan related parameters                   |
| <i>FLOWCTL_OPTION</i>         | choose to configure flow control related parameters           |
| <i>HASHH_OPTION</i>           | choose to configure hash high                                 |
| <i>HASL_OPTION</i>            | choose to configure hash low                                  |
| <i>FILTER_OPTION</i>          | choose to configure frame filter related parameters           |
| <i>HALFDUPLEX_OPTION</i>      | choose to configure halfduplex mode related parameters        |
| <i>TIMER_OPTION</i>           | choose to configure time counter related parameters           |
| <i>INTERFRAMEGAP_OPTION</i>   | choose to configure the inter frame gap related parameters    |

### Enum enet\_mediemode\_enum

Table 3-248. Enum enet\_mediemode\_enum

| member name                  | Function description            |
|------------------------------|---------------------------------|
| <i>ENET_AUTO_NEGOTIATION</i> | PHY auto negotiation            |
| <i>ENET_100M_FULLDUPLEX</i>  | 100Mbit/s, full-duplex          |
| <i>ENET_100M_HALFDUPLEX</i>  | 100Mbit/s, half-duplex          |
| <i>ENET_10M_FULLDUPLEX</i>   | 10Mbit/s, full-duplex           |
| <i>ENET_10M_HALFDUPLEX</i>   | 10Mbit/s, half-duplex           |
| <i>ENET_LOOPBACK</i>         | MAC in loopback mode at the MII |

| member name | Function description |
|-------------|----------------------|
| <i>MODE</i> |                      |

### Enum enet\_chksumconf\_enum

**Table 3-249. Enum enet\_chksumconf\_enum**

| member name  | Function description  |
|--|---|
| <i>ENET_NO_AUTOCHECKSUM</i>                                    | disable IP frame checksum function  |
| <i>ENET_AUTOCHEC</i><br><i>KSUM_DROP_FAIL</i><br><i>FRAMES</i> | enable IP frame checksum function   |
| <i>ENET_AUTOCHEC</i><br><i>KSUM_ACCEPT_FRAME</i>               | enable IP frame checksum function, and the received frame with only payload error but no other errors will not be dropped |

### Enum enet\_frmrecept\_enum

**Table 3-250. Enum enet\_frmrecept\_enum**

| member name                       | Function description                                     |
|-----------------------------------|--|
| <i>ENET_PROMISCUOUS_MODE</i>      | promiscuous mode enabled                                 |
| <i>ENET_RECEIVEALL</i>            | all received frame are forwarded to application          |
| <i>ENET_BROADCAST_FRAMES_PASS</i> | the address filters pass all received broadcast frames   |
| <i>ENET_BROADCAST_FRAMES_DROP</i> | the address filters filter all incoming broadcast frames |

### Enum enet\_registers\_type\_enum

**Table 3-251. Enum enet\_registers\_type\_enum**

| member name        | Function description  |
|--------------------|---|
| <i>ALL_MAC_REG</i> | get the registers within the offset scope range ENET_MAC_CFG to ENET_MAC_FCTH     |
| <i>ALL_MSC_REG</i> | get the registers within the offset scope range ENET_MSC_CTL to ENET_MSC_RGUFCNT  |
| <i>ALL_PTP_REG</i> | get the registers within the offset scope range ENET_PTP_TSCTL to ENET_PTP_PPSCTL |
| <i>ALL_DMA_REG</i> | get the registers within the offset scope range ENET_DMA_BCTL to ENET_DMA_CRBADDR |

**Enum enet\_dmadirection\_enum**
**Table 3-252. Enum enet\_dmadirection\_enum**

| member name        | Function description |
|--------------------|----------------------|
| <i>ENET_DMA_TX</i> | DMA Tx descriptors   |
| <i>ENET_DMA_RX</i> | DMA Rx descriptors   |

**Enum enet\_phydirection\_enum**
**Table 3-253. Enum enet\_phydirection\_enum**

| member name           | Function description        |
|-----------------------|-----------------------------|
| <i>ENET_PHY_WRITE</i> | write data to phy register  |
| <i>ENET_PHY_READ</i>  | read data from phy register |

**Enum enet\_regdirection\_enum**
**Table 3-254. Enum enet\_regdirection\_enum**

| member name           | Function description |
|-----------------------|----------------------|
| <i>ENET_REG_READ</i>  | read register        |
| <i>ENET_REG_WRITE</i> | write register       |

**Enum enet\_macaddress\_enum**
**Table 3-255. Enum enet\_macaddress\_enum**

| member name               | Function description     |
|---------------------------|--------------------------|
| <i>ENET_MAC_ADDR_ESS0</i> | set MAC address 0 filter |
| <i>ENET_MAC_ADDR_ESS1</i> | set MAC address 1 filter |
| <i>ENET_MAC_ADDR_ESS2</i> | set MAC address 2 filter |
| <i>ENET_MAC_ADDR_ESS3</i> | set MAC address 3 filter |

**Enum enet\_descstate\_enum**
**Table 3-256. Enum enet\_descstate\_enum**

| member name                   | Function description   |
|-------------------------------|--|
| <i>TXDESC_COLLISION_COUNT</i> | the number of collisions occurred before the frame was transmitted       |
| <i>TXDESC_BUFFER_1_ADDR</i>   | the buffer1 address of the Tx frame                                      |
| <i>RXDESC_FRAME_LENGTH</i>    | the byte length of the received frame that was transferred to the buffer |
| <i>RXDESC_BUFFER</i>          | receive buffer 1 size  |

| member name          | Function description                |
|----------------------|-------------------------------------|
| _1_SIZE              |                                     |
| RXDESC_BUFFER_2_SIZE | receive buffer 2 size               |
| RXDESC_BUFFER_1_ADDR | the buffer1 address of the Rx frame |

### enet\_deinit

The description of enet\_deinit is shown as below:

**Table 3-257. Function enet\_deinit**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_deinit   |
| <b>Function prototype</b>    | void enet_deinit(void);   |
| <b>Function descriptions</b> | deinitialize the ENET, and reset structure parameters for ENET initialization |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | rcu_periph_reset_enable /rcu_periph_reset_disable()/enet_initpara_reset       |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* deinitialize the ENET */
enet_deinit();
```

### enet\_initpara\_config

The description of enet\_initpara\_config is shown as below:

**Table 3-258. Function enet\_initpara\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_initpara_config   |
| <b>Function prototype</b>    | void enet_initpara_config(enet_option_enum option, uint32_t para);   |
| <b>Function descriptions</b> | configure the parameters which are usually less cared for initialization, this function must be called before enet_init(), otherwise configuration will be no effect                         |
| <b>Precondition</b>          | enet_initpara_reset(void)  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| option                       | different function option, which is related to several parameters, refer to <a href="#">Table 3-247. Enum enet_option_enum</a><br>only one parameter can be selected which is shown as below |

|  |   |
|--|---|
| <i>FORWARD_OPTION</i>  | choose to configure the frame forward related parameters  |
| <i>DMABUS_OPTION</i>   | choose to configure the DMA bus mode related parameters   |
| <i>DMA_MAXBURST_OPTION</i>   | choose to configure the DMA max burst related parameters  |
| <i>DMA_ARBITRATION_OPTION</i>  | choose to configure the DMA arbitration related parameters  |
| <i>STORE_OPTION</i>  | choose to configure the store forward mode related parameters   |
| <i>DMA_OPTION</i>  | choose to configure the DMA related parameters  |
| <i>VLAN_OPTION</i>   | choose to configure vlan related parameters   |
| <i>FLOWCTL_OPTION</i>  | choose to configure flow control related parameters   |
| <i>HASHH_OPTION</i>  | choose to configure hash high   |
| <i>HASHL_OPTION</i>  | choose to configure hash low  |
| <i>FILTER_OPTION</i>   | choose to configure frame filter related parameters   |
| <i>HALFDUPLEX_OPTION</i>   | choose to configure halfduplex mode related parameters  |
| <i>TIMER_OPTION</i>  | choose to configure time counter related parameters   |
| <i>INTERFRAMEGAP_OPTION</i>  | choose to configure the inter frame gap related parameters  |
| <b>Input parameter{in}</b>   |   |
| <b>para</b><br>(the value according to the parameter <b>option</b> ) | all the related values should be configured which are shown as below<br>example: <b>para</b> = (value1   value2   value3...)  |
| When value of parameter <b>option</b> is <i>FORWARD_OPTION</i>       |   |
| value1   | <i>ENET_AUTO_PADCRC_DROP_ENABLE / ENET_AUTO_PADCRC_DROP_DISABLE</i>   |
| value2   | <i>ENET_FORWARD_ERRFRAMES_ENABLE / ENET_FORWARD_ERRFRAMES_DISABLE</i>   |
| value3   | <i>ENET_FORWARD_UNDERSZ_GOODFRAMES_ENABLE / ENET_FORWARD_UNDERSZ_GOODFRAMES_DISABLE</i>   |
| When value of parameter <b>option</b> is <i>DMABUS_OPTION</i>        |   |
| value1   | <i>ENET_ADDRESS_ALIGN_ENABLE / ENET_ADDRESS_ALIGN_DISABLE</i>   |
| value2   | <i>ENET_FIXED_BURST_ENABLE / ENET_FIXED_BURST_DISABLE</i>   |
| When value of parameter <b>option</b> is <i>DMA_MAXBURST_OPTION</i>  |   |
| value1   | <i>ENET_RXDP_1BEAT / ENET_RXDP_2BEAT / ENET_RXDP_4BEAT / ENET_RXDP_8BEAT / ENET_RXDP_16BEAT / ENET_RXDP_32BEAT / ENET_RXDP_4xPGBL_4BEAT / ENET_RXDP_4xPGBL_8BEAT / ENET_RXDP_4xPGBL_16BEAT / ENET_RXDP_4xPGBL_32BEAT / ENET_RXDP_4xPGBL_64BEAT / ENET_RXDP_4xPGBL_128BEAT</i> |
| value2   | <i>ENET_PGBL_1BEAT / ENET_PGBL_2BEAT / ENET_PGBL_4BEAT / ENET_PGBL_8BEAT / ENET_PGBL_16BEAT / ENET_PGBL_32BEAT / ENET_PGBL_4xPGBL_4BEAT / ENET_PGBL_4xPGBL_8BEAT / ENET_PGBL_4xPGBL_16BEAT / ENET_PGBL_4xPGBL_32BEAT / ENET_PGBL_4xPGBL_64BEAT / ENET_PGBL_4xPGBL_128BEAT</i> |

|  |   |
|--|---|
| value3   | <i>ENET_RX_TX_DIFFERENT_PGBL / ENET_RX_TX_SAME_PGBL</i>   |
| When value of parameter <b>option</b> is <i>DMA_ARBITRATION_OPTION</i> |   |
| value1   | <i>ENET_ARBITRATION_RXPRIORTX / ENET_ARBITRATION_RXTX_1_1 / ENET_ARBITRATION_RXTX_2_1 / ENET_ARBITRATION_RXTX_3_1 / ENET_ARBITRATION_RXTX_4_1</i>   |
| When value of parameter <b>option</b> is <i>STORE_OPTION</i>           |   |
| value1   | <i>ENET_RX_MODE_STOREFORWARD / ENET_RX_MODE_CUTTHROUGH</i>  |
| value2   | <i>ENET_TX_MODE_STOREFORWARD / ENET_TX_MODE_CUTTHROUGH</i>  |
| value3   | <i>ENET_RX_THRESHOLD_64BYTES / ENET_RX_THRESHOLD_32BYTES / ENET_RX_THRESHOLD_96BYTES / ENET_RX_THRESHOLD_128BYTES</i>   |
| value4   | <i>ENET_TX_THRESHOLD_64BYTES / ENET_TX_THRESHOLD_128BYTES / ENET_TX_THRESHOLD_192BYTES / ENET_TX_THRESHOLD_256BYTES / ENET_TX_THRESHOLD_40BYTES / ENET_TX_THRESHOLD_32BYTES / ENET_TX_THRESHOLD_24BYTES / ENET_TX_THRESHOLD_16BYTES</i> |
| When value of parameter <b>option</b> is <i>DMA_OPTION</i>             |   |
| value1   | <i>ENET_FLUSH_RXFRAME_ENABLE / ENET_FLUSH_RXFRAME_DISABLE</i>   |
| value2   | <i>ENET_SECONDFRAME_OPT_ENABLE / ENET_SECONDFRAME_OPT_DISABLE</i>   |
| When value of parameter <b>option</b> is <i>VLAN_OPTION</i>            |   |
| value1   | <i>ENET_VLANTAGCOMPARISON_12BIT / ENET_VLANTAGCOMPARISON_16BIT</i>  |
| value2   | <i>MAC_VLT_VLI(regval)</i>  |
| When value of parameter <b>option</b> is <i>FLOWCTL_OPTION</i>         |   |
| value1   | <i>MAC_FCTL_PTM(regval)</i>   |
| value2   | <i>ENET_ZERO_QUANTA_PAUSE_ENABLE / ENET_ZERO_QUANTA_PAUSE_DISABLE</i>   |
| value3   | <i>ENET_PAUSETIME_MINUS4 / ENET_PAUSETIME_MINUS28 / ENET_PAUSETIME_MINUS144 / ENET_PAUSETIME_MINUS256</i>   |
| value4   | <i>ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDTECT / ENET_UNIQUE_PAUSEDTECT</i>   |
| value5   | <i>ENET_RX_FLOWCONTROL_ENABLE / ENET_RX_FLOWCONTROL_DISABLE</i>   |
| value6   | <i>ENET_TX_FLOWCONTROL_ENABLE / ENET_TX_FLOWCONTROL_DISABLE</i>   |
| When value of parameter <b>option</b> is <i>HASHH_OPTION</i>           |   |
| value1   | <i>0x0~0xFFFF FFFFU</i>   |
| When value of parameter <b>option</b> is <i>HASHL_OPTION</i>           |   |
| value1   | <i>0x0~0xFFFF FFFFU</i>   |
| When value of parameter <b>option</b> is <i>FILTER_OPTION</i>          |   |
| value1   | <i>ENET_SRC_FILTER_NORMAL_ENABLE /</i>  |

|  |  |
|--|--|
|  | <i>ENET_SRC_FILTER_INVERSE_ENABLE / ENET_SRC_FILTER_DISABLE</i>  |
| value2   | <i>ENET_DEST_FILTER_INVERSE_ENABLE / ENET_DEST_FILTER_INVERSE_DISABLE</i>  |
| value3   | <i>ENET_MULTICAST_FILTER_HASH_OR_PERFECT / ENET_MULTICAST_FILTER_HASH / ENET_MULTICAST_FILTER_PERFECT / ENET_MULTICAST_FILTER_NONE</i>   |
| value4   | <i>ENET_UNICAST_FILTER EITHER / ENET_UNICAST_FILTER_HASH / ENET_UNICAST_FILTER_PERFECT</i>   |
| value5   | <i>ENET_PCFRM_PREVENT_ALL / ENET_PCFRM_PREVENT_PAUSEFRAME / ENET_PCFRM_FORWARD_ALL / ENET_PCFRM_FORWARD_FILTERED</i>   |
| When value of parameter <b>option</b> is <i>HALFDUPLEX_OPTION</i>    |  |
| value1   | <i>ENET_CARRIERSENSE_ENABLE / ENET_CARRIERSENSE_DISABLE</i>  |
| value2   | <i>ENET_RECEIVEOWN_ENABLE / ENET_RECEIVEOWN_DISABLE</i>  |
| value3   | <i>ENET_RETRYTRANSMISSION_ENABLE / ENET_RETRYTRANSMISSION_DISABLE</i>  |
| value4   | <i>ENET_BACKOFFLIMIT_10 / ENET_BACKOFFLIMIT_8 / ENET_BACKOFFLIMIT_4 / ENET_BACKOFFLIMIT_1</i>  |
| value5   | <i>ENET_DEFERRALCHECK_ENABLE / ENET_DEFERRALCHECK_DISABLE</i>  |
| When value of parameter <b>option</b> is <i>TIMER_OPTION</i>         |  |
| value1   | <i>ENET_WATCHDOG_ENABLE / ENET_WATCHDOG_DISABLE</i>  |
| value2   | <i>ENET_JABBER_ENABLE / ENET_JABBER_DISABLE</i>  |
| When value of parameter <b>option</b> is <i>INTERFRAMEGAP_OPTION</i> |  |
| value1   | <i>ENET_INTERFRAMEGAP_96BIT / ENET_INTERFRAMEGAP_88BIT / ENET_INTERFRAMEGAP_80BIT / ENET_INTERFRAMEGAP_72BIT / ENET_INTERFRAMEGAP_64BIT / ENET_INTERFRAMEGAP_56BIT / ENET_INTERFRAMEGAP_48BIT / ENET_INTERFRAMEGAP_40BIT</i> |
| <b>Output parameter{out}</b>   |  |
| -  | -  |
| <b>Return value</b>  |  |
| -  | -  |

Example:

```
/* config DMA option of the ENET */
enet_initpara_reset();

enet_initpara_config(DMA_OPTION, ENET_FLUSH_RXFRAME_ENABLE | ENET_SECONDFRAME_OPT_ENABLE | ENET_NORMAL_DESCRIPTOR);
```

### **enet\_init**

The description of enet\_init is shown as below:

Table 3-259. Function enet\_init

|  |  |
|--|--|
| <b>Function name</b>                       | enet_init  |
| <b>Function prototype</b>                  | ErrStatus enet_init(enet_mediemode_enum mediemode,<br>enetc_chksumconf_enum checksum, enet_frmrecept_enum recept);                             |
| <b>Function descriptions</b>               | initialize ENET peripheral with generally concerned parameters and the less cared parameters   |
| <b>Precondition</b>                        | enet_deinit ()   |
| <b>The called functions</b>                | enet_phy_config /enet_phy_write_read   |
| <b>Input parameter{in}</b>                 |  |
| <b>mediemode</b>                           | PHY mode and mac loopback configurations, refer to <a href="#">Table 3-248. Enum enet_mediemode_enum</a><br>only one parameter can be selected |
| <i>ENET_AUTO_NEGOTIATION</i>               | PHY auto negotiation   |
| <i>ENET_100M_FULLDUPLEX</i>                | 100Mbit/s, full-duplex   |
| <i>ENET_100M_HALFDUPLEX</i>                | 100Mbit/s, half-duplex   |
| <i>ENET_10M_FULLDUPLEX</i>                 | 10Mbit/s, full-duplex  |
| <i>ENET_10M_HALFDUPLEX</i>                 | 10Mbit/s, half-duplex  |
| <i>ENET_LOOPBACKMODE</i>                   | MAC in loopback mode at the MII  |
| <b>Input parameter{in}</b>                 |  |
| <b>checksum</b>                            | IP frame checksum offload function, refer to <a href="#">Table 3-249. Enum enet_chksumconf_enum</a><br>only one parameter can be selected      |
| <i>ENET_NO_AUTOCHECKSUM</i>                | disable IP frame checksum function   |
| <i>ENET_AUTOCHECKSUM_DROP_FAILFRAMES</i>   | enable IP frame checksum function  |
| <i>ENET_AUTOCHECKSUM_ACCEPT_FAILFRAMES</i> | enable IP frame checksum function, and the received frame with only payload error but no other errors will not be dropped                      |
| <b>Input parameter{in}</b>                 |  |
| <b>recept</b>                              | frame filter function, refer to <a href="#">Table 3-250. Enum enet_frmrecept_enum</a><br>only one parameter can be selected                    |
| <i>ENET_PROMISCUOUS_MODE</i>               | promiscuous mode enabled   |
| <i>ENET_RECEIVEALL</i>                     | all received frame are forwarded to application  |
| <i>ENET_BROADCAST_F</i>                    | the address filters pass all received broadcast frames   |

|                         |  |  |
|-------------------------|--|--|
| <i>RAMES_PASS</i>       |  |  |
| <i>ENET_BROADCAST_F</i> | the address filters filter all incoming broadcast frames |  |
| <i>RAMES_DROP</i>       | <b>Output parameter{out}</b>                             |  |
| -                       | -  |  |
| <b>Return value</b>     |  |  |
| <b>ErrStatus</b>        | ERROR or SUCCESS   |  |

Example:

```
/* initialize ENET peripheral */

ErrStatus enet_init_status;

enet_init_status = enet_init(ENET_AUTO_NEGOTIATION, ENET_AUTOCHECKSUM_DR
OP_FAILFRAMES, ENET_BROADCAST_FRAMES_PASS);
```

### **enet\_software\_reset**

The description of **enet\_software\_reset** is shown as below:

**Table 3-260. Function enet\_software\_reset**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_software_reset  |
| <b>Function prototype</b>    | ErrStatus enet_software_reset(void);                           |
| <b>Function descriptions</b> | reset all core internal registers located in CLK_TX and CLK_RX |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>ErrStatus</b>             | ERROR or SUCCESS   |

Example:

```
/* reset all core internal registers located in CLK_TX and CLK_RX */

ErrStatus reval_state = ERROR;

reval_state = enet_software_reset();
```

### **enet\_rxframe\_size\_get**

The description of **enet\_rxframe\_size\_get** is shown as below:

**Table 3-261. Function enet\_rxframe\_size\_get**

|                           |                                       |
|---------------------------|---------------------------------------|
| <b>Function name</b>      | enet_rxframe_size_get                 |
| <b>Function prototype</b> | uint32_t enet_rxframe_size_get(void); |

|                              |   |
|------------------------------|---|
| <b>Function descriptions</b> | check receive frame valid and return frame size |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | enet_rxframe_drop()                             |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| uint32_t                     | size of received frame 0x0 - 0x3FFF             |

Example:

```
/* check receive frame valid */

uint32_t reval;

reval = enet_rxframe_size_get();
```

### enet\_descriptors\_chain\_init

The description of enet\_descriptors\_chain\_init is shown as below:

**Table 3-262. Function enet\_descriptors\_chain\_init**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_descriptors_chain_init   |
| <b>Function prototype</b>    | void enet_descriptors_chain_init(enet_dmadirection_enum direction);   |
| <b>Function descriptions</b> | initialize the DMA Tx/Rx descriptors' parameters in chain mode  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| direction                    | the descriptors which users want to init, refer to <a href="#">Table 3-252. Enum enet_dmadirection_enum</a><br>only one parameter can be selected which is shown as below |
| ENET_DMA_TX                  | DMA Tx descriptors  |
| ENET_DMA_RX                  | DMA Rx descriptors  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* initialize the DMA Tx/Rx descriptors' parameters in chain mode */

enet_descriptors_chain_init(ENET_DMA_TX);
```

### enet\_descriptors\_ring\_init

The description of enet\_descriptors\_ring\_init is shown as below:

**Table 3-263. Function enet\_descriptors\_ring\_init**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_descriptors_ring_init  |
| <b>Function prototype</b>    | void enet_descriptors_ring_init(enet_dmadirection_enum direction);  |
| <b>Function descriptions</b> | initialize the DMA Tx/Rx descriptors' parameters in ring mode   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>direction</b>             | the descriptors which users want to init, refer to <a href="#">Table 3-252. Enum enet_dmadirection_enum</a><br>only one parameter can be selected which is shown as below |
| <i>ENET_DMA_TX</i>           | DMA Tx descriptors  |
| <i>ENET_DMA_RX</i>           | DMA Rx descriptors  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* initialize the DMA Tx/Rx descriptors' parameters in ring mode */
enet_descriptors_ring_init(ENET_DMA_TX);
```

### enet\_frame\_receive

The description of enet\_frame\_receive is shown as below:

**Table 3-264. Function enet\_frame\_receive**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_frame_receive  |
| <b>Function prototype</b>    | ErrStatus enet_frame_receive(uint8_t *buffer, uint32_t bufsize);  |
| <b>Function descriptions</b> | handle current received frame data to application buffer  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>bufsize</b>               | the size of buffer which is the parameter in function, (0 -- 1524)  |
| <b>Output parameter{out}</b> |   |
| <b>buffer</b>                | pointer to the received frame data<br>if the input is NULL, user should copy data in application by himself |
| <b>Return value</b>          |   |
| <b>ErrStatus</b>             | ERROR or SUCCESS  |

Example:

```
/* transfer received frame data to application buffer */
uint8_t data_buffer[1500];
```

```

  uint32_t data_size;
  enet_frame_receive(data_buffer, data_size);

```

### **enet\_frame\_transmit**

The description of enet\_frame\_transmit is shown as below:

**Table 3-265. Function enet\_frame\_transmit**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_frame_transmit  |
| <b>Function prototype</b>    | ErrStatus enet_frame_transmit(uint8_t *buffer, uint32_t length);   |
| <b>Function descriptions</b> | handle application buffer data to transmit it  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>buffer</b>                | pointer to the frame data to be transmitted<br>if the input is NULL, user should handle the data in application by himself |
| <b>Input parameter{in}</b>   |  |
| <b>length</b>                | the length of frame data to be transmitted, (0 -- 1524)  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>ErrStatus</b>             | ERROR or SUCCESS   |

Example:

```

/* transfer buffer data of application */

uint8_t data_buffer[1500];
uint32_t data_size = 800;
enet_frame_transmit(data_buffer, data_size);

```

### **enet\_transmit\_checksum\_config**

The description of enet\_transmit\_checksum\_config is shown as below:

**Table 3-266. Function enet\_transmit\_checksum\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_transmit_checksum_config  |
| <b>Function prototype</b>    | void enet_transmit_checksum_config(enet_descriptors_struct *desc,<br>uint32_t checksum);   |
| <b>Function descriptions</b> | configure the transmit IP frame checksum offload calculation and insertion   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>desc</b>                  | the descriptor pointer which users want to configure, the structure members<br>can refer to <a href="#">Table 3-238. Structure enet_descriptors_struct</a> |

| Input parameter{in}                   |   |
|---------------------------------------|---|
| <b>checksum</b>                       | IP frame checksum configuration<br>only one parameter can be selected which is shown as below |
| <i>ENET_CHECKSUM_DISABLE</i>          | checksum insertion disabled   |
| <i>ENET_CHECKSUM_IP_V4HEADER</i>      | only IP header checksum calculation and insertion are enabled                                 |
| <i>ENET_CHECKSUM_TCUPDPICMSEGMENT</i> | TCP/UDP/ICMP checksum insertion calculated but pseudo-header                                  |
| <i>ENET_CHECKSUM_TCUPDPICM_FULL</i>   | TCP/UDP/ICMP checksum insertion fully calculated  |
| Output parameter{out}                 |   |
| -                                     | -   |
| Return value                          |   |
| -                                     | -   |

Example:

```
/* configure the transmit IP frame checksum offload calculation and insertion */
enet_descriptors_struct rx_desc;
enet_transmit_checksum_config(rx_desc, ENET_CHECKSUM_TCUPDPICM_FULL);
```

### enet\_enable

The description of enet\_enable is shown as below:

**Table 3-267. Function enet\_enable**

| <b>Function name</b>         | enet_enable   |
|------------------------------|---|
| <b>Function prototype</b>    | void enet_enable(void);                                     |
| <b>Function descriptions</b> | ENET Tx and Rx function enable (include MAC and DMA module) |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | enet_tx_enable /enet_rx_enable                              |
| Input parameter{in}          |   |
| -                            | -   |
| Output parameter{out}        |   |
| -                            | -   |
| Return value                 |   |
| -                            | -   |

Example:

```
/* enable the ENET */
enet_enable();
```

### **enet\_disable**

The description of enet\_disable is shown as below:

**Table 3-268. Function enet\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_disable   |
| <b>Function prototype</b>    | void enet_disable(void);                                     |
| <b>Function descriptions</b> | ENET Tx and Rx function disable (include MAC and DMA module) |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | enet_tx_disable /enet_rx_disable                             |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable the ENET */
enet_disable();
```

### **enet\_mac\_address\_set**

The description of enet\_mac\_address\_set is shown as below:

**Table 3-269. Function enet\_mac\_address\_set**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_mac_address_set  |
| <b>Function prototype</b>    | void enet_mac_address_set(enet_macaddress_enum mac_addr, uint8_t paddr[]);  |
| <b>Function descriptions</b> | configure MAC address   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| mac_addr                     | select which MAC address will be set, refer to <a href="#">Table 3-255. Enum enet_macaddress enum</a><br>only one parameter can be selected which is shown as below |
| ENET_MAC_ADDRESS_S0          | set MAC address 0 filter  |
| ENET_MAC_ADDRESS_S1          | set MAC address 1 filter  |
| ENET_MAC_ADDRESS_S2          | set MAC address 2 filter  |
| ENET_MAC_ADDRESS_S3          | set MAC address 3 filter  |

| Input parameter{in}   |   |
|-----------------------|---|
| <b>paddr</b>          | the buffer pointer which stores the MAC address little-ending store, such as MAC address is aa:bb:cc:dd:ee:22, the buffer is {22, ee, dd, cc, bb, aa} |
| Output parameter{out} |   |
| -                     | -   |
| Return value          |   |
| -                     | -   |

Example:

```
/* config mac address */
netif->hwaddr[0] = 0x02;
netif->hwaddr[1] = 0xaa;
netif->hwaddr[2] = 0xbb;
netif->hwaddr[3] = 0xcc;
netif->hwaddr[4] = 0xdd;
netif->hwaddr[5] = 0xee;
enet_mac_address_set(ENET_MAC_ADDRESS0, netif->hwaddr);
```

### **enet\_mac\_address\_get**

The description of enet\_mac\_address\_get is shown as below:

**Table 3-270. Function enet\_mac\_address\_get**

| <b>Function name</b>         | enet_mac_address_get  |
|------------------------------|---|
| <b>Function prototype</b>    | void enet_mac_address_get(enet_macaddress_enum mac_addr, uint8_t paddr[]);  |
| <b>Function descriptions</b> | get MAC address   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| Input parameter{in}          |   |
| <b>mac_addr</b>              | select which MAC address will be set, refer to <a href="#">Table 3-255. Enum enet_macaddress_enum</a><br>only one parameter can be selected which is shown as below |
| <b>ENET_MAC_ADDRESS0</b>     | set MAC address 0 filter  |
| <b>ENET_MAC_ADDRESS1</b>     | set MAC address 1 filter  |
| <b>ENET_MAC_ADDRESS2</b>     | set MAC address 2 filter  |

|                               |   |
|-------------------------------|---|
| <i>ENET_MAC_ADDRESS</i><br>S3 | set MAC address 3 filter  |
| <b>Output parameter{out}</b>  |   |
| <b>paddr</b>                  | the buffer pointer which stores the MAC address little-ending store, such as MAC address is aa:bb:cc:dd:ee:22, the buffer is {22, ee, dd, cc, bb, aa} |
| <b>Return value</b>           |   |
| -                             | -   |

Example:

```
/* get mac address */
enet_mac_address_get(ENET_MAC_ADDRESS0, netif->hwaddr);
```

### **enet\_flag\_get**

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

**Table 3-271. Function enet\_flag\_get**

|   |   |
|---|---|
| <b>Function name</b>                        | enet_flag_get   |
| <b>Function prototype</b>                   | FlagStatus enet_flag_get(enet_flag_enum enet_flag);   |
| <b>Function descriptions</b>                | get the ENET MAC/MSC/PTP/DMA status flag  |
| <b>Precondition</b>                         | -   |
| <b>The called functions</b>                 | -   |
| <b>Input parameter{in}</b>                  |   |
| <b>enet_flag</b>                            | ENET status flag, refer to <a href="#">Table 3-240. Enum enet_flag_enum</a><br>only one parameter can be selected which is shown as below |
| <i>ENET_MAC_FLAG_MP</i><br><i>KR</i>        | magic packet received flag  |
| <i>ENET_MAC_FLAG_W</i><br><i>UFR</i>        | wakeup frame received flag  |
| <i>ENET_MAC_FLAG_FL</i><br><i>OWCONTROL</i> | flow control status flag  |
| <i>ENET_MAC_FLAG_W</i><br><i>UM</i>         | WUM status flag   |
| <i>ENET_MAC_FLAG_MS</i><br><i>C</i>         | MSC status flag   |
| <i>ENET_MAC_FLAG_MS</i><br><i>CR</i>        | MSC receive status flag   |
| <i>ENET_MAC_FLAG_MS</i><br><i>CT</i>        | MSC transmit status flag  |
| <i>ENET_MAC_FLAG_TM</i><br><i>ST</i>        | time stamp trigger status flag  |
| <i>ENET_PTP_FLAG_TS</i>                     | timestamp second counter overflow flag  |

|  |  |
|--|--|
| SCO                                    |  |
| <i>ENET_PTP_FLAG_TT_M</i>              | target time match flag                             |
| <i>ENET_MSC_FLAG_RF_CE</i>             | received frames CRC error flag                     |
| <i>ENET_MSC_FLAG_RF_AE</i>             | received frames alignment error flag               |
| <i>ENET_MSC_FLAG_RG_UF</i>             | received good unicast frames flag                  |
| <i>ENET_MSC_FLAG_TG_FSC</i>            | transmitted good frames single collision flag      |
| <i>ENET_MSC_FLAG_TG_FMSC</i>           | transmitted good frames more single collision flag |
| <i>ENET_MSC_FLAG_TG_F</i>              | transmitted good frames flag                       |
| <i>ENET_DMA_FLAG_TS</i>                | transmit status flag                               |
| <i>ENET_DMA_FLAG_TP_S</i>              | transmit process stopped status flag               |
| <i>ENET_DMA_FLAG_TB_U</i>              | transmit buffer unavailable status flag            |
| <i>ENET_DMA_FLAG_TJ_T</i>              | transmit jabber timeout status flag                |
| <i>ENET_DMA_FLAG_RO</i>                | receive overflow status flag                       |
| <i>ENET_DMA_FLAG_TU</i>                | transmit underflow status flag                     |
| <i>ENET_DMA_FLAG_RS</i>                | receive status flag                                |
| <i>ENET_DMA_FLAG_RB_U</i>              | receive buffer unavailable status flag             |
| <i>ENET_DMA_FLAG_RP_S</i>              | receive process stopped status flag                |
| <i>ENET_DMA_FLAG_R_WT</i>              | receive watchdog timeout status flag               |
| <i>ENET_DMA_FLAG_ET</i>                | early transmit status flag                         |
| <i>ENET_DMA_FLAG_FB_E</i>              | fatal bus error status flag                        |
| <i>ENET_DMA_FLAG_ER</i>                | early receive status flag                          |
| <i>ENET_DMA_FLAG_AI</i>                | abnormal interrupt summary flag                    |
| <i>ENET_DMA_FLAG_NI</i>                | normal interrupt summary flag                      |
| <i>ENET_DMA_FLAG_EB_DMA_ERROR</i>      | DMA error flag                                     |
| <i>ENET_DMA_FLAG_EB_TRANSFER_ERROR</i> | transfer error flag                                |
| <i>ENET_DMA_FLAG_EB_ACCESS_ERROR</i>   | access error flag                                  |

|                              |                               |
|------------------------------|-------------------------------|
| <i>ENET_DMA_FLAG_MS_C</i>    | MSC status flag               |
| <i>ENET_DMA_FLAG_WUM</i>     | WUM status flag               |
| <i>ENET_DMA_FLAG_TS_T</i>    | timestamp trigger status flag |
| <b>Output parameter{out}</b> |                               |
| -                            | -                             |
| <b>Return value</b>          |                               |
| <b>FlagStatus</b>            | SET or RESET                  |

Example:

```
/* check whether the specified flag bit is set */
enet_flag_get(ENET_DMA_FLAG_RS);
```

### enet\_flag\_clear

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

**Table 3-272. Function enet\_flag\_clear**

|                               |  |
|-------------------------------|--|
| <b>Function name</b>          | enet_flag_clear  |
| <b>Function prototype</b>     | void enet_flag_clear(enet_flag_clear_enum enet_flag);  |
| <b>Function descriptions</b>  | clear the ENET DMA status flag   |
| <b>Precondition</b>           | -  |
| <b>The called functions</b>   | -  |
| <b>Input parameter{in}</b>    |  |
| <b>enet_flag</b>              | ENET DMA flag clear, refer to <a href="#">Table 3-241. Enum enet_flag_clear_enum</a><br>only one parameter can be selected which is shown as below |
| <i>ENET_DMA_FLAG_TS_CLR</i>   | transmit status flag clear   |
| <i>ENET_DMA_FLAG_TP_S_CLR</i> | transmit process stopped status flag clear   |
| <i>ENET_DMA_FLAG_TB_U_CLR</i> | transmit buffer unavailable status flag clear  |
| <i>ENET_DMA_FLAG_TJ_T_CLR</i> | transmit jabber timeout status flag clear  |
| <i>ENET_DMA_FLAG_RO_CLR</i>   | receive overflow status flag clear   |
| <i>ENET_DMA_FLAG_TU_CLR</i>   | transmit underflow status flag clear   |
| <i>ENET_DMA_FLAG_RS_CLR</i>   | receive status flag clear  |
| <i>ENET_DMA_FLAG_RB</i>       | receive buffer unavailable status flag clear   |

|                               |  |
|-------------------------------|--|
| <i>U_CLR</i>                  |  |
| <i>ENET_DMA_FLAG_RP_S_CLR</i> | receive process stopped status flag clear  |
| <i>ENET_DMA_FLAG_R_WT_CLR</i> | receive watchdog timeout status flag clear |
| <i>ENET_DMA_FLAG_ET_CLR</i>   | early transmit status flag clear           |
| <i>ENET_DMA_FLAG_FB_E_CLR</i> | fatal bus error status flag clear          |
| <i>ENET_DMA_FLAG_ER_CLR</i>   | early receive status flag clear            |
| <i>ENET_DMA_FLAG_AI_CLR</i>   | abnormal interrupt summary flag clear      |
| <i>ENET_DMA_FLAG_NI_CLR</i>   | normal interrupt summary flag clear        |
| <b>Output parameter{out}</b>  |  |
| -                             | -  |
| <b>Return value</b>           |  |
| -                             | -  |

Example:

```
/* clear the specified flag bit */
enet_flag_clear(ENET_DMA_FLAG_RS_CLR);
```

### enet\_interrupt\_enable

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

**Table 3-273. Function enet\_interrupt\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_interrupt_enable  |
| <b>Function prototype</b>    | void enet_interrupt_enable(enet_int_enum enet_int);  |
| <b>Function descriptions</b> | enable ENET MAC/MSC/DMA interrupt  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>enet_int</b>              | ENET interrupt, refer to <a href="#">Table 3-242. Enum enet_int enum</a><br>only one parameter can be selected which is shown as below |
| <i>ENET_MAC_INT_WUM_IM</i>   | WUM interrupt mask   |
| <i>ENET_MAC_INT_TMS_TIM</i>  | timestamp trigger interrupt mask   |
| <i>ENET_MSC_INT_RFC_EIM</i>  | received frame CRC error interrupt mask  |

|                                   |  |
|-----------------------------------|--|
| <i>ENET_MSC_INT_RFA<br/>EIM</i>   | received frames alignment error interrupt mask               |
| <i>ENET_MSC_INT_RGU<br/>FIM</i>   | received good unicast frames interrupt mask                  |
| <i>ENET_MSC_INT_TGF<br/>SCIM</i>  | transmitted good frames single collision interrupt mask      |
| <i>ENET_MSC_INT_TGF<br/>MSCIM</i> | transmitted good frames more single collision interrupt mask |
| <i>ENET_MSC_INT_TGFI<br/>M</i>    | transmitted good frames interrupt mask                       |
| <i>ENET_DMA_INT_TIE</i>           | transmit interrupt enable                                    |
| <i>ENET_DMA_INT_TPSI<br/>E</i>    | transmit process stopped interrupt enable                    |
| <i>ENET_DMA_INT_TBUI<br/>E</i>    | transmit buffer unavailable interrupt enable                 |
| <i>ENET_DMA_INT_TJTI<br/>E</i>    | transmit jabber timeout interrupt enable                     |
| <i>ENET_DMA_INT_ROIE</i>          | receive overflow interrupt enable                            |
| <i>ENET_DMA_INT_TUIE</i>          | transmit underflow interrupt enable                          |
| <i>ENET_DMA_INT_RIE</i>           | receive interrupt enable                                     |
| <i>ENET_DMA_INT_RBUI<br/>E</i>    | receive buffer unavailable interrupt enable                  |
| <i>ENET_DMA_INT_RPSI<br/>E</i>    | receive process stopped interrupt enable                     |
| <i>ENET_DMA_INT_RWT<br/>IE</i>    | receive watchdog timeout interrupt enable                    |
| <i>ENET_DMA_INT_ETIE</i>          | early transmit interrupt enable                              |
| <i>ENET_DMA_INT_FBEI<br/>E</i>    | fatal bus error interrupt enable                             |
| <i>ENET_DMA_INT_ERIE</i>          | early receive interrupt enable                               |
| <i>ENET_DMA_INT_AIE</i>           | abnormal interrupt summary enable                            |
| <i>ENET_DMA_INT_NIE</i>           | normal interrupt summary enable                              |
| <b>Output parameter{out}</b>      |  |
| -                                 | -  |
| <b>Return value</b>               |  |
| -                                 | -  |

Example:

```
/* enable normal interrupt summary */
enet_interrupt_enable(ENET_DMA_INT_NIE);
```

### enet\_interrupt\_disable

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

**Table 3-274. Function enet\_interrupt\_disable**

|                                   |  |
|-----------------------------------|--|
| <b>Function name</b>              | enet_interrupt_disable   |
| <b>Function prototype</b>         | void enet_interrupt_disable(enet_int_enum enet_int);   |
| <b>Function descriptions</b>      | disable ENET MAC/MSC/DMA interrupt   |
| <b>Precondition</b>               | -  |
| <b>The called functions</b>       | -  |
| <b>Input parameter{in}</b>        |  |
| <b>enet_int</b>                   | ENET interrupt, refer to <a href="#">Table 3-242. Enum enet_int_enum</a><br>only one parameter can be selected which is shown as below |
| <i>ENET_MAC_INT_WUM<br/>IM</i>    | WUM interrupt mask   |
| <i>ENET_MAC_INT_TMS<br/>TIM</i>   | timestamp trigger interrupt mask   |
| <i>ENET_MSC_INT_RFC<br/>EIM</i>   | received frame CRC error interrupt mask  |
| <i>ENET_MSC_INT_RFA<br/>EIM</i>   | received frames alignment error interrupt mask   |
| <i>ENET_MSC_INT_RGU<br/>FIM</i>   | received good unicast frames interrupt mask  |
| <i>ENET_MSC_INT_TGF<br/>SCIM</i>  | transmitted good frames single collision interrupt mask  |
| <i>ENET_MSC_INT_TGF<br/>MSCIM</i> | transmitted good frames more single collision interrupt mask   |
| <i>ENET_MSC_INT_TGFI<br/>M</i>    | transmitted good frames interrupt mask   |
| <i>ENET_DMA_INT_TIE</i>           | transmit interrupt enable  |
| <i>ENET_DMA_INT_TPSI<br/>E</i>    | transmit process stopped interrupt enable  |
| <i>ENET_DMA_INT_TBUI<br/>E</i>    | transmit buffer unavailable interrupt enable   |
| <i>ENET_DMA_INT_TJTI<br/>E</i>    | transmit jabber timeout interrupt enable   |
| <i>ENET_DMA_INT_ROIE</i>          | receive overflow interrupt enable  |
| <i>ENET_DMA_INT_TUIE</i>          | transmit underflow interrupt enable  |
| <i>ENET_DMA_INT_RIE</i>           | receive interrupt enable   |
| <i>ENET_DMA_INT_RBUI<br/>E</i>    | receive buffer unavailable interrupt enable  |
| <i>ENET_DMA_INT_RPSI<br/>E</i>    | receive process stopped interrupt enable   |
| <i>ENET_DMA_INT_RWT</i>           | receive watchdog timeout interrupt enable  |

| <i>IE</i>                    |                                   |
|------------------------------|-----------------------------------|
| <i>ENET_DMA_INT_ETIE</i>     | early transmit interrupt enable   |
| <i>ENET_DMA_INT_FBEI_E</i>   | fatal bus error interrupt enable  |
| <i>ENET_DMA_INT_ERIE</i>     | early receive interrupt enable    |
| <i>ENET_DMA_INT_AIE</i>      | abnormal interrupt summary enable |
| <i>ENET_DMA_INT_NIE</i>      | normal interrupt summary enable   |
| <b>Output parameter{out}</b> |                                   |
| -                            | -                                 |
| <b>Return value</b>          |                                   |
| -                            | -                                 |

Example:

```
/* disable normal interrupt summary */
enet_interrupt_disable(ENET_DMA_INT_NIE);
```

### enet\_interrupt\_flag\_get

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

**Table 3-275. Function enet\_interrupt\_flag\_get**

|                                |  |
|--------------------------------|--|
| <b>Function name</b>           | enet_interrupt_flag_get  |
| <b>Function prototype</b>      | FlagStatus enet_interrupt_flag_get(enet_int_flag_enum int_flag);   |
| <b>Function descriptions</b>   | get ENET MAC/MSC/DMA interrupt flag  |
| <b>Precondition</b>            | -  |
| <b>The called functions</b>    | -  |
| <b>Input parameter{in}</b>     |  |
| <i>int_flag</i>                | ENET interrupt flag, refer to <a href="#">Table 3-243. Enum enet_int_flag_enum</a><br>only one parameter can be selected which is shown as below |
| <i>ENET_MAC_INT_FLA_G_WUM</i>  | WUM status flag  |
| <i>ENET_MAC_INT_FLA_G_MSC</i>  | MSC status flag  |
| <i>ENET_MAC_INT_FLA_G_MSCR</i> | MSC receive status flag  |
| <i>ENET_MAC_INT_FLA_G_MSCT</i> | MSC transmit status flag   |
| <i>ENET_MAC_INT_FLA_G_TMST</i> | time stamp trigger status flag   |
| <i>ENET_MSC_INT_FLA_G_RFCE</i> | received frames CRC error flag   |
| <i>ENET_MSC_INT_FLA_G_RFAE</i> | received frames alignment error flag   |

|  |  |
|--|--|
| <i>ENET_MSC_INT_FLA</i><br><i>G_RGUF</i>   | received good unicast frames flag                  |
| <i>ENET_MSC_INT_FLA</i><br><i>G_TGFSC</i>  | transmitted good frames single collision flag      |
| <i>ENET_MSC_INT_FLA</i><br><i>G_TGFMSC</i> | transmitted good frames more single collision flag |
| <i>ENET_MSC_INT_FLA</i><br><i>G_TGF</i>    | transmitted good frames flag                       |
| <i>ENET_DMA_INT_FLA</i><br><i>G_TS</i>     | transmit status flag                               |
| <i>ENET_DMA_INT_FLA</i><br><i>G_TPS</i>    | transmit process stopped status flag               |
| <i>ENET_DMA_INT_FLA</i><br><i>G_TBU</i>    | transmit buffer unavailable status flag            |
| <i>ENET_DMA_INT_FLA</i><br><i>G_TJT</i>    | transmit jabber timeout status flag                |
| <i>ENET_DMA_INT_FLA</i><br><i>G_RO</i>     | receive overflow status flag                       |
| <i>ENET_DMA_INT_FLA</i><br><i>G_TU</i>     | transmit underflow status flag                     |
| <i>ENET_DMA_INT_FLA</i><br><i>G_RS</i>     | receive status flag                                |
| <i>ENET_DMA_INT_FLA</i><br><i>G_RBU</i>    | receive buffer unavailable status flag             |
| <i>ENET_DMA_INT_FLA</i><br><i>G_RPS</i>    | receive process stopped status flag                |
| <i>ENET_DMA_INT_FLA</i><br><i>G_RWT</i>    | receive watchdog timeout status flag               |
| <i>ENET_DMA_INT_FLA</i><br><i>G_ET</i>     | early transmit status flag                         |
| <i>ENET_DMA_INT_FLA</i><br><i>G_FBE</i>    | fatal bus error status flag                        |
| <i>ENET_DMA_INT_FLA</i><br><i>G_ER</i>     | early receive status flag                          |
| <i>ENET_DMA_INT_FLA</i><br><i>G_AI</i>     | abnormal interrupt summary flag                    |
| <i>ENET_DMA_INT_FLA</i><br><i>G_NI</i>     | normal interrupt summary flag                      |
| <i>ENET_DMA_INT_FLA</i><br><i>G_MSC</i>    | MSC status flag                                    |
| <i>ENET_DMA_INT_FLA</i><br><i>G_WUM</i>    | WUM status flag                                    |
| <i>ENET_DMA_INT_FLA</i>                    | timestamp trigger status flag                      |

|                              |              |
|------------------------------|--------------|
| <i>G_TST</i>                 |              |
| <b>Output parameter{out}</b> |              |
| -                            | -            |
| <b>Return value</b>          |              |
| <b>FlagStatus</b>            | SET or RESET |

Example:

```
/* check whether the specified flag bit is set or not */
enet_interrupt_flag_get(ENET_DMA_INT_FLAG_RS);
```

### **enet\_interrupt\_flag\_clear**

The description of `enet_interrupt_flag_clear` is shown as below:

**Table 3-276. Function `enet_interrupt_flag_clear`**

|   |  |
|---|--|
| <b>Function name</b>                    | <code>enet_interrupt_flag_clear</code>   |
| <b>Function prototype</b>               | <code>void enet_interrupt_flag_clear(enet_int_flag_clear_enum int_flag_clear);</code>  |
| <b>Function descriptions</b>            | clear ENET DMA interrupt flag  |
| <b>Precondition</b>                     | -  |
| <b>The called functions</b>             | -  |
| <b>Input parameter{in}</b>              |  |
| <code>int_flag_clear</code>             | clear ENET interrupt flag, refer to <a href="#">Table 3-244. Enum enet_int_flag_clear_enum</a><br>only one parameter can be selected which is shown as below |
| <code>ENET_DMA_INT_FLA_G_TS_CLR</code>  | transmit status flag   |
| <code>ENET_DMA_INT_FLA_G_TPS_CLR</code> | transmit process stopped status flag   |
| <code>ENET_DMA_INT_FLA_G_TBU_CLR</code> | transmit buffer unavailable status flag  |
| <code>ENET_DMA_INT_FLA_G_TJT_CLR</code> | transmit jabber timeout status flag  |
| <code>ENET_DMA_INT_FLA_G_RO_CLR</code>  | receive overflow status flag   |
| <code>ENET_DMA_INT_FLA_G_TU_CLR</code>  | transmit underflow status flag   |
| <code>ENET_DMA_INT_FLA_G_RS_CLR</code>  | receive status flag  |
| <code>ENET_DMA_INT_FLA_G_RBU_CLR</code> | receive buffer unavailable status flag   |
| <code>ENET_DMA_INT_FLA_G_RPS_CLR</code> | receive process stopped status flag  |
| <code>ENET_DMA_INT_FLA</code>           | receive watchdog timeout status flag   |

|   |                                 |
|---|---------------------------------|
| <i>G_RWT_CLR</i>                            |                                 |
| <i>ENET_DMA_INT_FLA</i><br><i>G_ET_CLR</i>  | early transmit status flag      |
| <i>ENET_DMA_INT_FLA</i><br><i>G_FBE_CLR</i> | fatal bus error status flag     |
| <i>ENET_DMA_INT_FLA</i><br><i>G_ER_CLR</i>  | early receive status flag       |
| <i>ENET_DMA_INT_FLA</i><br><i>G_AI_CLR</i>  | abnormal interrupt summary flag |
| <i>ENET_DMA_INT_FLA</i><br><i>G_NI_CLR</i>  | normal interrupt summary flag   |
| <b>Output parameter{out}</b>                |                                 |
| -   | -                               |
| <b>Return value</b>                         |                                 |
| -   | -                               |

Example:

```
/* clear receive status flag */
enet_interrupt_flag_clear(ENET_DMA_INT_FLAG_RS);
```

### **enet\_tx\_enable**

The description of enet\_tx\_enable is shown as below:

**Table 3-277. Function enet\_tx\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_tx_enable                                       |
| <b>Function prototype</b>    | void enet_tx_enable(void);                           |
| <b>Function descriptions</b> | ENET Tx function enable (include MAC and DMA module) |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | enet_txfifo_flush()                                  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable transport function of MAC and DMA */
enet_tx_enable();
```

### **enet\_tx\_disable**

The description of enet\_tx\_disable is shown as below:

**Table 3-278. Function enet\_tx\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_tx_disable                                       |
| <b>Function prototype</b>    | void enet_tx_disable(void);                           |
| <b>Function descriptions</b> | ENET Tx function disable (include MAC and DMA module) |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | enet_txfifo_flush()                                   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable transport function of MAC and DMA */
enet_tx_disable();
```

### **enet\_rx\_enable**

The description of enet\_rx\_enable is shown as below:

**Table 3-279. Function enet\_rx\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_rx_enable                                       |
| <b>Function prototype</b>    | void enet_rx_enable(void);                           |
| <b>Function descriptions</b> | ENET Rx function enable (include MAC and DMA module) |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable reception function of MAC and DMA */
enet_rx_enable();
```

### **enet\_rx\_disable**

The description of enet\_rx\_disable is shown as below:

**Table 3-280. Function enet\_rx\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_rx_disable                                       |
| <b>Function prototype</b>    | void enet_rx_disable(void);                           |
| <b>Function descriptions</b> | ENET Rx function disable (include MAC and DMA module) |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable reception function of MAC and DMA */
enet_rx_disable();
```

### **enet\_registers\_get**

The description of enet\_registers\_get is shown as below:

**Table 3-281. Function enet\_registers\_get**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_registers_get   |
| <b>Function prototype</b>    | void enet_registers_get(enet_registers_type_enum type, uint32_t *preg, uint32_t num);  |
| <b>Function descriptions</b> | put registers value into the application buffer  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>type</b>                  | register type which will be get, refer to <a href="#">Table 3-251. Enum enet registers type enum</a><br>only one parameter can be selected which is shown as below |
| <i>ALL_MAC_REG</i>           | get the registers within the offset scope range ENET_MAC_CFG to ENET_MAC_FCTH  |
| <i>ALL_MSC_REG</i>           | get the registers within the offset scope range ENET_MSC_CTL to ENET_MSC_RGUFCTN   |
| <i>ALL_PTP_REG</i>           | get the registers within the offset scope range ENET_PTP_TSCTL to ENET_PTP_PPSCTL  |
| <i>ALL_DMA_REG</i>           | get the registers within the offset scope range ENET_DMA_BCTL to ENET_DMA_CRBADDR  |

| Input parameter{in}   |   |
|-----------------------|---|
| <b>num</b>            | the number of registers that the user want to get, (0 -- 54)  |
| Output parameter{out} |   |
| <b>preg</b>           | the application buffer pointer for storing the register value |
| Return value          |   |
| -                     | -   |

Example:

```
/* get all mac registers value */

uint32_t register_buffer[5];

enet_registers_get(ALL_MAC_REG, 5, register_buffer);
```

### **enet\_address\_filter\_enable**

The description of enet\_address\_filter\_enable is shown as below:

**Table 3-282. Function enet\_address\_filter\_enable**

| <b>Function name</b>         | enet_address_filter_enable   |
|------------------------------|--|
| <b>Function prototype</b>    | void enet_address_filter_enable(enet_macaddress_enum mac_addr);  |
| <b>Function descriptions</b> | enable the MAC address filter  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| Input parameter{in}          |  |
| <b>mac_addr</b>              | select which MAC address will be enable, refer to <a href="#">Table 3-255. Enum enet_macaddress_enum</a><br>only one parameter can be selected which is shown as below |
| <i>ENET_MAC_ADDRESS1</i>     | enable MAC address 1 filter  |
| <i>ENET_MAC_ADDRESS2</i>     | enable MAC address 2 filter  |
| <i>ENET_MAC_ADDRESS3</i>     | enable MAC address 3 filter  |
| Output parameter{out}        |  |
| -                            | -  |
| Return value                 |  |
| -                            | -  |

Example:

```
/* enable the MAC address 1 filter */

enet_address_filter_enable(ENET_MAC_ADDRESS1);
```

### **enet\_address\_filter\_disable**

The description of enet\_address\_filter\_disable is shown as below:

**Table 3-283. Function enet\_address\_filter\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_address_filter_disable  |
| <b>Function prototype</b>    | void enet_address_filter_disable(enet_macaddress_enum mac_addr);   |
| <b>Function descriptions</b> | disable the MAC address filter   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>mac_addr</b>              | select which MAC address will be enable, refer to <a href="#">Table 3-255. Enum enet_macaddress enum</a><br>only one parameter can be selected which is shown as below |
| <i>ENET_MAC_ADDRESS1</i>     | enable MAC address 1 filter  |
| <i>ENET_MAC_ADDRESS2</i>     | enable MAC address 2 filter  |
| <i>ENET_MAC_ADDRESS3</i>     | enable MAC address 3 filter  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable the MAC address 1 filter */
enet_address_filter_disable(ENET_MAC_ADDRESS1);
```

### **enet\_address\_filter\_config**

The description of enet\_address\_filter\_config is shown as below:

**Table 3-284. Function enet\_address\_filter\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_address_filter_config   |
| <b>Function prototype</b>    | void enet_address_filter_config(enet_macaddress_enum mac_addr,<br>uint32_t addr_mask, uint32_t filter_type);   |
| <b>Function descriptions</b> | configure the MAC address filter   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>mac_addr</b>              | select which MAC address will be enable, refer to <a href="#">Table 3-255. Enum enet_macaddress enum</a><br>only one parameter can be selected which is shown as below |

|                                   |   |
|-----------------------------------|---|
| <i>ENET_MAC_ADDRESS_S1</i>        | enable MAC address 1 filter   |
| <i>ENET_MAC_ADDRESS_S2</i>        | enable MAC address 2 filter   |
| <i>ENET_MAC_ADDRESS_S3</i>        | enable MAC address 3 filter   |
| <b>Input parameter{in}</b>        |   |
| <b>addr_mask</b>                  | select which MAC address bytes will be mask<br>one or more parameters can be selected which are shown as below      |
| <i>ENET_ADDRESS_MASK_SK_BYTE0</i> | mask ENET_MAC_ADDR1L[7:0] bits  |
| <i>ENET_ADDRESS_MASK_SK_BYTE1</i> | mask ENET_MAC_ADDR1L[15:8] bits   |
| <i>ENET_ADDRESS_MASK_SK_BYTE2</i> | mask ENET_MAC_ADDR1L[23:16] bits  |
| <i>ENET_ADDRESS_MASK_SK_BYTE3</i> | mask ENET_MAC_ADDR1L [31:24] bits   |
| <i>ENET_ADDRESS_MASK_SK_BYTE4</i> | mask ENET_MAC_ADDR1H [7:0] bits   |
| <i>ENET_ADDRESS_MASK_SK_BYTE5</i> | mask ENET_MAC_ADDR1H [15:8] bits  |
| <b>Input parameter{in}</b>        |   |
| <b>filter_type</b>                | select which MAC address filter type will be selected<br>only one parameter can be selected which is shown as below |
| <i>ENET_ADDRESS_FILTER_ER_SA</i>  | The MAC address is used to compared with the SA field of the received frame   |
| <i>ENET_ADDRESS_FILTER_ER_DA</i>  | The MAC address is used to compared with the DA field of the received frame   |
| <b>Output parameter{out}</b>      |   |
| -                                 | -   |
| <b>Return value</b>               |   |
| -                                 | -   |

Example:

```
/* config the MAC address 1 filter */
enet_address_filter_config(ENET_MAC_ADDRESS1, ENET_ADDRESS_MASK_BYTE0 |
ENET_ADDRESS_MASK_BYTE1 | ENET_ADDRESS_MASK_BYTE2, ENET_ADDRESS_
_FILTER_DA);
```

### **enet\_phy\_config**

The description of `enet_phy_config` is shown as below:

**Table 3-285. Function enet\_phy\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_phy_config  |
| <b>Function prototype</b>    | ErrStatus enet_phy_config(void);                                     |
| <b>Function descriptions</b> | PHY interface configuration (configure SMI clock and reset PHY chip) |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | rcu_clock_freq_get()/enet_phy_write_read()                           |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>ErrStatus</b>             | ERROR or SUCCESS   |

Example:

```
/* config PHY interface */
enet_phy_config();
```

### **enet\_phy\_write\_read**

The description of enet\_phy\_write\_read is shown as below:

**Table 3-286. Function enet\_phy\_write\_read**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_phy_write_read  |
| <b>Function prototype</b>    | ErrStatus enet_phy_write_read(enet_phydirection_enum direction, uint16_t phy_address, uint16_t phy_reg, uint16_t *pvalue);       |
| <b>Function descriptions</b> | write to / read from a PHY register  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>direction</b>             | only one parameter can be selected which is shown as below, refer to<br><a href="#">Table 3-253. Enum enet_phydirection_enum</a> |
| <i>ENET_PHY_WRITE</i>        | write data to phy register   |
| <i>ENET_PHY_READ</i>         | read data from phy register  |
| <b>Input parameter{in}</b>   |  |
| <b>phy_address</b>           | 0x0 - 0x1F   |
| <b>Input parameter{in}</b>   |  |
| <b>phy_reg</b>               | 0x0 - 0x1F   |
| <b>Input parameter{in}</b>   |  |
| <b>pvalue</b>                | the value will be written to the PHY register in ENET_PHY_WRITE direction  |
| <b>Output parameter{out}</b> |  |
| <b>pvalue</b>                | the value will be read from the PHY register in ENET_PHY_READ direction  |
| <b>Return value</b>          |  |
| <b>ErrStatus</b>             | ERROR or SUCCESS   |

Example:

```

/* write 0 to PHY BCR register */

uint16_t temp_phy = 0U;

phy_state = enet_phy_write_read(ENET_PHY_WRITE, PHY_ADDRESS, PHY_REG_BCR,
&temp_phy);
  
```

### **enet\_phyloopback\_enable**

The description of enet\_phyloopback\_enable is shown as below:

**Table 3-287. Function enet\_phyloopback\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_phyloopback_enable                  |
| <b>Function prototype</b>    | ErrStatus enet_phyloopback_enable(void); |
| <b>Function descriptions</b> | enable the loopback function of PHY chip |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | enet_phy_write_read()                    |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>ErrStatus</b>             | ERROR or SUCCESS                         |

Example:

```

/* enable the loopback function of PHY chip */

ErrStatus phy_state = ERROR;

phy_state = enet_phyloopback_enable();
  
```

### **enet\_phyloopback\_disable**

The description of enet\_phyloopback\_disable is shown as below:

**Table 3-288. Function enet\_phyloopback\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_phyloopback_disable                  |
| <b>Function prototype</b>    | ErrStatus enet_phyloopback_disable(void); |
| <b>Function descriptions</b> | disable the loopback function of PHY chip |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | enet_phy_write_read                       |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |

| Return value |                  |
|--------------|------------------|
| ErrStatus    | ERROR or SUCCESS |

Example:

```
/* disable the loopback function of PHY chip */
ErrStatus phy_state = ERROR;
phy_state = enet_phyloopback_disable();
```

### **enet\_forward\_feature\_enable**

The description of enet\_forward\_feature\_enable is shown as below:

**Table 3-289. Function enet\_forward\_feature\_enable**

|   |   |
|---|---|
| <b>Function name</b>                    | enet_forward_feature_enable   |
| <b>Function prototype</b>               | void enet_forward_feature_enable(uint32_t feature);   |
| <b>Function descriptions</b>            | enable ENET forward feature   |
| <b>Precondition</b>                     | -   |
| <b>The called functions</b>             | -   |
| <b>Input parameter{in}</b>              |   |
| <b>feature</b>                          | the feature of ENET forward mode<br>one or more parameters can be selected which are shown as below |
| <i>ENET_AUTO_PADCR_C_DROP</i>           | the function of the MAC strips the Pad/FCS field on received frames                                 |
| <i>ENET_FORWARD_ER_RFRAMES</i>          | the function that all frame received with error except runt error are forwarded to memory           |
| <i>ENET_FORWARD_UNDERSZ_GOODFRAME_S</i> | the function that forwarding undersized good frames   |
| <b>Output parameter{out}</b>            |   |
| -                                       | -   |
| <b>Return value</b>                     |   |
| -                                       | -   |

Example:

```
/* enable the function that forwarding undersized good frames */
enet_forward_feature_enable(ENET_FORWARD_UNDERSZ_GOODFRAMES);
```

### **enet\_forward\_feature\_disable**

The description of enet\_forward\_feature\_disable is shown as below:

**Table 3-290. Function enet\_forward\_feature\_disable**

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

|   |   |
|---|---|
| <b>Function prototype</b>               | void enet_forward_feature_disable(uint32_t feature);  |
| <b>Function descriptions</b>            | disable ENET forward feature  |
| <b>Precondition</b>                     | -   |
| <b>The called functions</b>             | -   |
| <b>Input parameter{in}</b>              |   |
| <b>feature</b>                          | the feature of ENET forward mode<br>one or more parameters can be selected which are shown as below |
| <i>ENET_AUTO_PADCR_C_DROP</i>           | the function of the MAC strips the Pad/FCS field on received frames                                 |
| <i>ENET_FORWARD_ER_RFRAMES</i>          | the function that all frame received with error except runt error are forwarded to memory           |
| <i>ENET_FORWARD_UNDERSZ_GOODFRAME_S</i> | the function that forwarding undersized good frames   |
| <b>Output parameter{out}</b>            |   |
| -                                       | -   |
| <b>Return value</b>                     |   |
| -                                       | -   |

Example:

```
/* enable the function that forwarding undersized good frames */
enet_forward_feature_enable(ENET_FORWARD_UNDERSZ_GOODFRAME_S);
```

### enet\_filter\_feature\_enable

The description of enet\_filter\_feature\_enable is shown as below:

**Table 3-291. Function enet\_filter\_feature\_enable**

|                                   |  |
|-----------------------------------|--|
| <b>Function name</b>              | enet_filter_feature_enable   |
| <b>Function prototype</b>         | void enet_filter_feature_enable(uint32_t feature);   |
| <b>Function descriptions</b>      | enable ENET filter feature   |
| <b>Precondition</b>               | -  |
| <b>The called functions</b>       | -  |
| <b>Input parameter{in}</b>        |  |
| <b>feature</b>                    | the feature of ENET filter mode<br>one or more parameters can be selected which are shown as below |
| <i>ENET_SRC_FILTER</i>            | filter source address function   |
| <i>ENET_SRC_FILTER_INVERSE</i>    | inverse source address filtering result function   |
| <i>ENET_DEST_FILTER_INVERSE</i>   | inverse DA filtering result function   |
| <i>ENET_MULTICAST_FILTER_PASS</i> | pass all multicast frames function   |

|  |                                 |
|--|---------------------------------|
| <i>ENET_MULTICAST_FILTER_MODE_HASH</i> | HASH multicast filter function  |
| <i>ENET_UNICAST_FILTER_MODE_HASH</i>   | HASH unicast filter function    |
| <i>ENET_FILTER_MODE_EITHER</i>         | HASH or perfect filter function |
| <b>Output parameter{out}</b>           |                                 |
| -                                      | -                               |
| <b>Return value</b>                    |                                 |
| -                                      | -                               |

Example:

```
/* enable filter source address function */
enet_filter_feature_enable(ENET_SRC_FILTER);
```

### **enet\_filter\_feature\_disable**

The description of `enet_filter_feature_disable` is shown as below:

**Table 3-292. Function `enet_filter_feature_disable`**

|                                   |  |
|-----------------------------------|--|
| <b>Function name</b>              | enet_filter_feature_disable  |
| <b>Function prototype</b>         | void enet_filter_feature_disable(uint32_t feature);  |
| <b>Function descriptions</b>      | disable ENET filter feature  |
| <b>Precondition</b>               | -  |
| <b>The called functions</b>       | -  |
| <b>Input parameter{in}</b>        |  |
| <b>feature</b>                    | the feature of ENET filter mode<br>one or more parameters can be selected which are shown as below |
| <i>ENET_SRC_FILTER</i>            | filter source address function   |
| <i>ENET_SRC_FILTER_INVERSE</i>    | inverse source address filtering result function   |
| <i>ENET_DEST_FILTER_INVERSE</i>   | inverse DA filtering result function   |
| <i>ENET_MULTICAST_FILTER_PASS</i> | pass all multicast frames function   |
| <i>ENET_MULTICAST_FILTER_HASH</i> | HASH multicast filter function   |
| <i>ENET_UNICAST_FILTER_HASH</i>   | HASH unicast filter function   |
| <i>ENET_FILTER_MODE_EITHER</i>    | HASH or perfect filter function  |
| <b>Output parameter{out}</b>      |  |
| -                                 | -  |

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

Example:

```
/* disable filter source address function */
enet_filter_feature_disable(ENET_SRC_FILTER);
```

### **enet\_pauseframe\_generate**

The description of enet\_pauseframe\_generate is shown as below:

**Table 3-293. Function enet\_pauseframe\_generate**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_pauseframe_generate   |
| <b>Function prototype</b>    | ErrStatus enet_pauseframe_generate(void);  |
| <b>Function descriptions</b> | generate the pause frame, ENET will send pause frame after enable transmit flow control this function only use in full-duplex mode |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>ErrStatus</b>             | ERROR or SUCCESS   |

Example:

```
/* generate the pause frame */
ErrStatus reval;
reval = enet_pauseframe_generate();
```

### **enet\_pauseframe\_detect\_config**

The description of enet\_pauseframe\_detect\_config is shown as below:

**Table 3-294. Function enet\_pauseframe\_detect\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_pauseframe_detect_config   |
| <b>Function prototype</b>    | void enet_pauseframe_detect_config(uint32_t detect);                                  |
| <b>Function descriptions</b> | configure the pause frame detect type   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>detect</b>                | pause frame detect type<br>only one parameter can be selected which is shown as below |

|  |   |
|--|---|
| <i>ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDTECT</i> | besides the unique multicast address, MAC can also use the MAC0 address to detecting pause frame  |
| <i>ENET_UNIQUE_PAUSEDTECT</i>                  | only the unique multicast address for pause frame which is specified in IEEE802.3 can be detected |
| <b>Output parameter{out}</b>                   |   |
| -  | -   |
| <b>Return value</b>                            |   |
| -  | -   |

Example:

```
/* config the pause frame detect type as ENET_UNIQUE_PAUSEDTECT */
enet_pauseframe_detect_config(ENET_UNIQUE_PAUSEDTECT);
```

### **enet\_pauseframe\_config**

The description of enet\_pauseframe\_config is shown as below:

**Table 3-295. Function enet\_pauseframe\_config**

|                                 |   |
|---------------------------------|---|
| <b>Function name</b>            | enet_pauseframe_config  |
| <b>Function prototype</b>       | void enet_pauseframe_config(uint32_t pausetime, uint32_t pause_threshold);  |
| <b>Function descriptions</b>    | configure the pause frame parameters  |
| <b>Precondition</b>             | -   |
| <b>The called functions</b>     | -   |
| <b>Input parameter{in}</b>      |   |
| <b>pausetime</b>                | pause time in transmit pause control frame, (0 – 0xFFFF)  |
| <b>Input parameter{in}</b>      |   |
| <b>pause_threshold</b>          | the threshold of the pause timer for retransmitting frames automatically, this value must make sure to be less than configured pause time, only one parameter can be selected which is shown as below |
| <i>ENET_PAUSETIME_MI_NUS4</i>   | pause time minus 4 slot times   |
| <i>ENET_PAUSETIME_MI_NUS28</i>  | pause time minus 28 slot times  |
| <i>ENET_PAUSETIME_MI_NUS144</i> | pause time minus 144 slot times   |
| <i>ENET_PAUSETIME_MI_NUS256</i> | pause time minus 256 slot times   |
| <b>Output parameter{out}</b>    |   |
| -                               | -   |
| <b>Return value</b>             |   |
| -                               | -   |

Example:

```
/* config pause time minus 4 slot times */

enet_pauseframe_config(30, ENET_PAUSETIME_MINUS4);
```

### **enet\_flowcontrol\_threshold\_config**

The description of enet\_flowcontrol\_threshold\_config is shown as below:

**Table 3-296. Function enet\_flowcontrol\_threshold\_config**

|  |   |
|--|---|
| <b>Function name</b>                               | enet_flowcontrol_threshold_config   |
| <b>Function prototype</b>                          | void enet_flowcontrol_threshold_config(uint32_t deactive, uint32_t active);   |
| <b>Function descriptions</b>                       | configure the threshold of the flow control(deactive and active threshold)  |
| <b>Precondition</b>                                | -   |
| <b>The called functions</b>                        | -   |
| <b>Input parameter{in}</b>                         |   |
| <b>deactive</b>                                    | the threshold of the deactive flow control, this value should always be less than active flow control value, only one parameter can be selected which is shown as below |
| <i>ENET_DEACTIVE_TH<br/>RESHOLD_256BYTES</i>       | threshold level is 256 bytes  |
| <i>ENET_DEACTIVE_TH<br/>RESHOLD_512BYTES</i>       | threshold level is 512 bytes  |
| <i>ENET_DEACTIVE_TH<br/>RESHOLD_768BYTES</i>       | threshold level is 768 bytes  |
| <i>ENET_DEACTIVE_TH<br/>RESHOLD_1024BYTE<br/>S</i> | threshold level is 1024 bytes   |
| <i>ENET_DEACTIVE_TH<br/>RESHOLD_1280BYTE<br/>S</i> | threshold level is 1280 bytes   |
| <i>ENET_DEACTIVE_TH<br/>RESHOLD_1536BYTE<br/>S</i> | threshold level is 1536 bytes   |
| <i>ENET_DEACTIVE_TH<br/>RESHOLD_1792BYTE<br/>S</i> | threshold level is 1792 bytes   |
| <b>Input parameter{in}</b>                         |   |
| <b>active</b>                                      | the threshold of the active flow control, only one parameter can be selected which is shown as below  |
| <i>ENET_ACTIVE_THRE<br/>SHOLD_256BYTES</i>         | threshold level is 256 bytes  |
| <i>ENET_ACTIVE_THRE<br/>SHOLD_512BYTES</i>         | threshold level is 512 bytes  |

|   |                               |
|---|-------------------------------|
| <i>ENET_ACTIVE_THRE<br/>SHOLD_768BYTES</i>  | threshold level is 768 bytes  |
| <i>ENET_ACTIVE_THRE<br/>SHOLD_1024BYTES</i> | threshold level is 1024 bytes |
| <i>ENET_ACTIVE_THRE<br/>SHOLD_1280BYTES</i> | threshold level is 1280 bytes |
| <i>ENET_ACTIVE_THRE<br/>SHOLD_1536BYTES</i> | threshold level is 1536 bytes |
| <i>ENET_ACTIVE_THRE<br/>SHOLD_1792BYTES</i> | threshold level is 1792 bytes |
| <b>Output parameter{out}</b>                |                               |
| -   | -                             |
| <b>Return value</b>                         |                               |
| -   | -                             |

Example:

```
/* configure the threshold of the flow control */

enet_flowcontrol_threshold_config(ENET_DEACTIVE_THRESHOLD_256BYTES, ENET_A
CTIVE_THRESHOLD_256BYTES);
```

### **enet\_flowcontrol\_feature\_enable**

The description of enet\_flowcontrol\_feature\_enable is shown as below:

**Table 3-297. Function enet\_flowcontrol\_feature\_enable**

|                                    |  |
|------------------------------------|--|
| <b>Function name</b>               | enet_flowcontrol_feature_enable  |
| <b>Function prototype</b>          | void enet_flowcontrol_feature_enable(uint32_t feature);  |
| <b>Function descriptions</b>       | enable ENET flow control feature   |
| <b>Precondition</b>                | -  |
| <b>The called functions</b>        | -  |
| <b>Input parameter{in}</b>         |  |
| <b>feature</b>                     | the feature of ENET flow control mode<br>one or more parameters can be selected which are shown as below |
| <i>ENET_ZERO_QUANT<br/>A_PAUSE</i> | the automatic zero-quanta generation function  |
| <i>ENET_TX_FLOWCON<br/>TROL</i>    | the flow control operation in the MAC  |
| <i>ENET_RX_FLOWCON<br/>TROL</i>    | decoding function for the received pause frame and process it  |
| <i>ENET_BACK_PRESSU<br/>RE</i>     | back pressure operation in the MAC(only use in half-duplex mode)   |
| <b>Output parameter{out}</b>       |  |
| -                                  | -  |

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

Example:

```
/* enable the flow control operation in the MAC */
enet_flowcontrol_feature_enable(ENET_ZERO_QUANTA_PAUSE);
```

### **enet\_flowcontrol\_feature\_disable**

The description of enet\_flowcontrol\_feature\_disable is shown as below:

**Table 3-298. Function enet\_flowcontrol\_feature\_disable**

|                                |  |
|--------------------------------|--|
| <b>Function name</b>           | enet_flowcontrol_feature_disable   |
| <b>Function prototype</b>      | void enet_flowcontrol_feature_disable(uint32_t feature);   |
| <b>Function descriptions</b>   | disable ENET flow control feature  |
| <b>Precondition</b>            | -  |
| <b>The called functions</b>    | -  |
| <b>Input parameter{in}</b>     |  |
| <b>feature</b>                 | the feature of ENET flow control mode<br>one or more parameters can be selected which are shown as below |
| <i>ENET_ZERO_QUANT_A_PAUSE</i> | the automatic zero-quanta generation function  |
| <i>ENET_TX_FLOWCONTROL</i>     | the flow control operation in the MAC  |
| <i>ENET_RX_FLOWCONTROL</i>     | decoding function for the received pause frame and process it  |
| <i>ENET_BACK_PRESSURE</i>      | back pressure operation in the MAC(only use in half-duplex mode)   |
| <b>Output parameter{out}</b>   |  |
| -                              | -  |
| <b>Return value</b>            |  |
| -                              | -  |

Example:

```
/* disable the automatic zero-quanta generation function */
enet_flowcontrol_feature_disable(ENET_ZERO_QUANTA_PAUSE);
```

### **enet\_dmaprocess\_state\_get**

The description of enet\_dmaprocess\_state\_get is shown as below:

**Table 3-299. Function enet\_dmaprocess\_state\_get**

|                           |   |
|---------------------------|---|
| <b>Function name</b>      | enet_dmaprocess_state_get   |
| <b>Function prototype</b> | uint32_t enet_dmaprocess_state_get(enet_dmadirection_enum direction); |

|                              |   |
|------------------------------|---|
| <b>Function descriptions</b> | get the dma transmit/receive process state  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>direction</b>             | choose the direction of DMA process which users want to resume<br>only one parameter can be selected which is shown as below  |
| <i>ENET_DMA_TX</i>           | DMA transmit process  |
| <i>ENET_DMA_RX</i>           | DMA receive process   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>uint32_t</b>              | state of dma process, the value range shows below:<br><br>ENET_RX_STATE_STOPPED / ENET_RX_STATE_FETCHING /<br>ENET_RX_STATE_WAITING / ENET_RX_STATE_SUSPENDED /<br>ENET_RX_STATE_CLOSING / ENET_RX_STATE_QUEUEING /<br>ENET_TX_STATE_STOPPED / ENET_TX_STATE_FETCHING /<br>ENET_TX_STATE_WAITING / ENET_TX_STATE_READING /<br>ENET_TX_STATE_SUSPENDED / ENET_TX_STATE_CLOSING |

Example:

```

/* get the dma receive process state */

uint32_t reval;

reval = enet_dmaprocess_state_get(ENET_DMA_RX);

if(ENET_RX_STATE_SUSPENDED == reval){

    do...

}

```

### **enet\_dmaprocess\_resume**

The description of `enet_dmaprocess_resume` is shown as below:

**Table 3-300. Function `enet_dmaprocess_resume`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_dmaprocess_resume   |
| <b>Function prototype</b>    | void enet_dmaprocess_resume(enet_dmadirection_enum direction);   |
| <b>Function descriptions</b> | poll the DMA transmission/reception enable by writing any value to the ENET_DMA_TPEN/ENET_DMA_RPEN register, this will make the DMA to resume transmission/reception |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>direction</b>             | choose the direction of DMA process which users want to resume   |

|                              |  |
|------------------------------|--|
|                              | only one parameter can be selected which is shown as below |
| ENET_DMA_TX                  | DMA transmit process                                       |
| ENET_DMA_RX                  | DMA receive process  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable DMA receive process */

enet_dmaprocess_resume(ENET_DMA_RX);
```

### **enet\_rxprocess\_check\_recovery**

The description of enet\_rxprocess\_check\_recovery is shown as below:

**Table 3-301. Function enet\_rxprocess\_check\_recovery**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_rxprocess_check_recovery             |
| <b>Function prototype</b>    | void enet_rxprocess_check_recovery(void); |
| <b>Function descriptions</b> | check and recover the Rx process          |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* check and recover the Rx process */

enet_rxprocess_check_recovery();
```

### **enet\_txfifo\_flush**

The description of enet\_txfifo\_flush is shown as below:

**Table 3-302. Function enet\_txfifo\_flush**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_txfifo_flush  |
| <b>Function prototype</b>    | ErrStatus enet_txfifo_flush(void);   |
| <b>Function descriptions</b> | flush the ENET transmit FIFO, and wait until the flush operation completes |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |

|                              |                  |
|------------------------------|------------------|
| -                            | -                |
| <b>Output parameter{out}</b> |                  |
| -                            | -                |
| <b>Return value</b>          |                  |
| <b>ErrStatus</b>             | ERROR or SUCCESS |

Example:

```
/* flush the ENET transmit FIFO */
```

```
ErrStatus reval;
```

```
reval = enet_txfifo_flush();
```

### **enet\_current\_desc\_address\_get**

The description of `enet_current_desc_address_get` is shown as below:

**Table 3-303. Function `enet_current_desc_address_get`**

|                                    |   |
|------------------------------------|---|
| <b>Function name</b>               | enet_current_desc_address_get   |
| <b>Function prototype</b>          | uint32_t enet_current_desc_address_get(enet_desc_reg_enum addr_get);  |
| <b>Function descriptions</b>       | get the transmit/receive address of current descriptor, or current buffer, or descriptor table  |
| <b>Precondition</b>                | -   |
| <b>The called functions</b>        | -   |
| <b>Input parameter{in}</b>         |   |
| <b>addr_get</b>                    | choose the address which users want to get, refer to <a href="#">Table 3-245. Enum enet_desc_reg_enum</a><br>only one parameter can be selected which is shown as below |
| <i>ENET_RX_DESC_TAB<br/>LE</i>     | the start address of the receive descriptor table   |
| <i>ENET_RX_CURRENT_<br/>DESC</i>   | the start descriptor address of the current receive descriptor read by the RxDMA controller   |
| <i>ENET_RX_CURRENT_<br/>BUFFER</i> | the current receive buffer address being read by the RxDMA controller   |
| <i>ENET_TX_DESC_TAB<br/>LE</i>     | the start address of the transmit descriptor table  |
| <i>ENET_TX_CURRENT_<br/>DESC</i>   | the start descriptor address of the current transmit descriptor read by the TxDMA controller  |
| <i>ENET_TX_CURRENT_<br/>BUFFER</i> | the current transmit buffer address being read by the TxDMA controller  |
| <b>Output parameter{out}</b>       |   |
| -                                  | -   |
| <b>Return value</b>                |   |
| <b>uint32_t</b>                    | 0- 0xFFFFFFFF   |

Example:

```

/* get the start address of the receive descriptor table */

uint32_t reval;

reval = enet_current_desc_address_get(ENET_RX_DESC_TABLE);
  
```

### **enet\_desc\_information\_get**

The description of enet\_desc\_information\_get is shown as below:

**Table 3-304. Function enet\_desc\_information\_get**

|                               |   |
|-------------------------------|---|
| <b>Function name</b>          | enet_desc_information_get   |
| <b>Function prototype</b>     | uint32_t enet_desc_information_get(enet_descriptors_struct *desc,<br>enetcstate_enum info_get);   |
| <b>Function descriptions</b>  | get the Tx or Rx descriptor information   |
| <b>Precondition</b>           | -   |
| <b>The called functions</b>   | -   |
| <b>Input parameter{in}</b>    |   |
| <b>desc</b>                   | the descriptor pointer which users want to get information, the structure<br>members can refer to <a href="#">Table 3-238. Structure<br/>enet_descriptors_struct</a>            |
| <b>Input parameter{in}</b>    |   |
| <b>info_get</b>               | the descriptor information type which is selected, refer to <a href="#">Table 3-256.<br/>Enum enetcstate enum</a><br>only one parameter can be selected which is shown as below |
| <b>RXDESC_BUFFER_1_SIZE</b>   | receive buffer 1 size   |
| <b>RXDESC_BUFFER_2_SIZE</b>   | receive buffer 2 size   |
| <b>RXDESC_FRAME_LENGTH</b>    | the byte length of the received frame that was transferred to the buffer  |
| <b>TXDESC_COLLISION_COUNT</b> | the number of collisions occurred before the frame was transmitted  |
| <b>RXDESC_BUFFER_1_ADDR</b>   | the buffer1 address of the Rx frame   |
| <b>TXDESC_BUFFER_1_ADDR</b>   | the buffer1 address of the Tx frame   |
| <b>Output parameter{out}</b>  |   |
| -                             | -   |
| <b>Return value</b>           |   |
| <b>uint32_t</b>               | descriptor information if value is 0xFFFFFFFFU, means the false input<br>parameter  |

Example:

```

/* get the reception buffer 1 size */

uint32_t reval;

enet_descriptors_struct rx_desc;

reval = enet_desc_information_get(rx_desc, RXDESC_BUFFER_1_SIZE);

enet_missed_frame_counter_get

```

The description of enet\_missed\_frame\_counter\_get is shown as below:

**Table 3-305. Function enet\_missed\_frame\_counter\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_missed_frame_counter_get   |
| <b>Function prototype</b>    | void enet_missed_frame_counter_get(uint32_t *rxfifo_drop, uint32_t *rdma_drop); |
| <b>Function descriptions</b> | get the number of missed frames during receiving                                |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| <b>rxfifo_drop</b>           | pointer to the number of frames dropped by RxFIFO                               |
| <b>Output parameter{out}</b> |   |
| <b>rdma_drop</b>             | pointer to the number of frames missed by the RxDMA controller                  |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```

/* get the number of missed frames during receiving */

uint32_t rxcnt, txcnt;

enet_missed_frame_counter_get(&rxcnt, &txcnt);

```

### **enet\_desc\_flag\_get**

The description of enet\_desc\_flag\_get is shown as below:

**Table 3-306. Function enet\_desc\_flag\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_desc_flag_get  |
| <b>Function prototype</b>    | FlagStatus enet_desc_flag_get(enet_descriptors_struct *desc, uint32_t desc_flag); |
| <b>Function descriptions</b> | get the bit flag of ENET DMA descriptor   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |

|   |  |
|---|--|
| <b>desc</b>   | the descriptor pointer which users want to get flag, the structure members can refer to <a href="#">Table 3-238. Structure enet_descriptors_struct</a> |
| <b>Input parameter{in}</b>  |  |
| <b>desc_flag</b><br>(the value according to the parameter <b>desc</b> ) | the bit flag of ENET DMA descriptor<br>only one parameter can be selected which is shown as below  |
| When type of parameter <b>desc</b> is TX                                |  |
| <i>ENET_TDES0_DB</i>  | deferred   |
| <i>ENET_TDES0_UFE</i>   | underflow error  |
| <i>ENET_TDES0_EXD</i>   | excessive deferral   |
| <i>ENET_TDES0_VFRM</i>  | VLAN frame   |
| <i>ENET_TDES0_ECO</i>   | excessive collision  |
| <i>ENET_TDES0_LCO</i>   | late collision   |
| <i>ENET_TDES0_NCA</i>   | no carrier   |
| <i>ENET_TDES0_LCA</i>   | loss of carrier  |
| <i>ENET_TDES0_IPPE</i>  | IP payload error   |
| <i>ENET_TDES0_FRMF</i>  | frame flushed  |
| <i>ENET_TDES0_JT</i>  | jabber timeout   |
| <i>ENET_TDES0_ES</i>  | error summary  |
| <i>ENET_TDES0_IPHE</i>  | IP header error  |
| <i>ENET_TDES0_TTMSS</i>   | transmit timestamp status  |
| <i>ENET_TDES0_TCHM</i>  | the second address chained mode  |
| <i>ENET_TDES0_TERM</i>  | transmit end of ring mode  |
| <i>ENET_TDES0_TTSEN</i>   | transmit timestamp function enable   |
| <i>ENET_TDES0_DPAD</i>  | disable adding pad   |
| <i>ENET_TDES0_DCRC</i>  | disable CRC  |
| <i>ENET_TDES0_FSG</i>   | first segment  |
| <i>ENET_TDES0_LSG</i>   | last segment   |
| <i>ENET_TDES0_INTC</i>  | interrupt on completion  |
| <i>ENET_TDES0_DAV</i>   | DAV bit  |
| When type of parameter <b>desc</b> is RX                                |  |
| <i>ENET_RDES0_PCERR</i>   | payload checksum error   |
| <i>ENET_RDES0_CERR</i>  | CRC error  |
| <i>ENET_RDES0_DBERR</i>   | dribble bit error  |
| <i>ENET_RDES0_RERR</i>  | receive error  |
| <i>ENET_RDES0_RWDT</i>  | receive watchdog timeout   |
| <i>ENET_RDES0_FRMT</i>  | frame type   |
| <i>ENET_RDES0_LCO</i>   | late collision   |
| <i>ENET_RDES0_IPHER</i><br><i>R</i>                                     | IP frame header error  |
| <i>ENET_RDES0_LDES</i>  | last descriptor  |
| <i>ENET_RDES0_FDES</i>  | first descriptor   |

|                              |                                 |
|------------------------------|---------------------------------|
| <i>ENET_RDES0_VTAG</i>       | VLAN tag                        |
| <i>ENET_RDES0_OERR</i>       | overflow error                  |
| <i>ENET_RDES0_LERR</i>       | length error                    |
| <i>ENET_RDES0_SAFF</i>       | SA filter fail                  |
| <i>ENET_RDES0_DERR</i>       | descriptor error                |
| <i>ENET_RDES0_ERRS</i>       | error summary                   |
| <i>ENET_RDES0_DAFF</i>       | destination address filter fail |
| <i>ENET_RDES0_DAV</i>        | descriptor available            |
| <b>Output parameter{out}</b> |                                 |
| -                            | -                               |
| <b>Return value</b>          |                                 |
| <b>FlagStatus</b>            | SET or RESET                    |

Example:

```
/* get the bit flag of ENET DMA descriptor */

FlagStatus reval;

enet_descriptors_struct p_txdesc;

reval = enet_desc_flag_get(p_txdesc, ENET_TDES0_TCHM);
```

### **enet\_desc\_flag\_set**

The description of `enet_desc_flag_set` is shown as below:

**Table 3-307. Function `enet_desc_flag_set`**

|   |   |
|---|---|
| <b>Function name</b>  | enet_desc_flag_set  |
| <b>Function prototype</b>   | void enet_desc_flag_set(enet_descriptors_struct *desc, uint32_t desc_flag);   |
| <b>Function descriptions</b>  | set the bit flag of ENET DMA descriptor   |
| <b>Precondition</b>   | -   |
| <b>The called functions</b>   | -   |
| <b>Input parameter{in}</b>  |   |
| <b>desc</b>   | the descriptor pointer which users want to set flag, the structure members can refer to <a href="#">Table 3-238. Structure <code>enet_descriptors_struct</code></a> |
| <b>Input parameter{in}</b>  |   |
| <b>desc_flag</b><br>(the value according to the parameter <b>desc</b> ) | the bit flag of ENET DMA descriptor<br>only one parameter can be selected which is shown as below   |
| When type of parameter <b>desc</b> is TX                                |   |
| <i>ENET_TDES0_VFRM</i>  | VLAN frame  |
| <i>ENET_TDES0_FRMF</i>  | frame flushed   |
| <i>ENET_TDES0_TCHM</i>  | the second address chained mode   |
| <i>ENET_TDES0_TERM</i>  | transmit end of ring mode   |
| <i>ENET_TDES0_TTSEN</i>   | transmit timestamp function enable  |

|  |                         |
|--|-------------------------|
| <i>ENET_TDES0_DPAD</i>                   | disable adding pad      |
| <i>ENET_TDES0_DCRC</i>                   | disable CRC             |
| <i>ENET_TDES0_FSG</i>                    | first segment           |
| <i>ENET_TDES0_LSG</i>                    | last segment            |
| <i>ENET_TDES0_INTC</i>                   | interrupt on completion |
| <i>ENET_TDES0_DAV</i>                    | DAV bit                 |
| When type of parameter <b>desc</b> is RX |                         |
| <i>ENET_RDES0_DAV</i>                    | descriptor available    |
| <b>Output parameter{out}</b>             |                         |
| -  | -                       |
| <b>Return value</b>                      |                         |
| -  | -                       |

Example:

```
/* set VLAN frame bit flag of ENET DMA descriptor */

enet_descriptors_struct p_txdesc;

enet_desc_flag_set(p_txdesc, ENET_TDES0_VFRM);
```

### **enet\_desc\_flag\_clear**

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

**Table 3-308. Function enet\_desc\_flag\_clear**

|   |  |
|---|--|
| <b>Function name</b>  | <code>enet_desc_flag_clear</code>  |
| <b>Function prototype</b>   | <code>void enet_desc_flag_clear(enet_descriptors_struct *desc, uint32_t desc_flag);</code>   |
| <b>Function descriptions</b>  | clear the bit flag of ENET DMA descriptor  |
| <b>Precondition</b>   | -  |
| <b>The called functions</b>   | -  |
| <b>Input parameter{in}</b>  |  |
| <b>desc</b>   | the descriptor pointer which users want to set flag, the structure members can refer to <a href="#">Table 3-238. Structure enet_descriptors_struct</a> |
| <b>Input parameter{in}</b>  |  |
| <b>desc_flag</b><br>(the value according to the parameter <b>desc</b> ) | the bit flag of ENET DMA descriptor<br>only one parameter can be selected which is shown as below  |
| When type of parameter <b>desc</b> is TX                                |  |
| <i>ENET_TDES0_VFRM</i>  | VLAN frame   |
| <i>ENET_TDES0_FRMF</i>  | frame flushed  |
| <i>ENET_TDES0_TCHM</i>  | the second address chained mode  |
| <i>ENET_TDES0_TERM</i>  | transmit end of ring mode  |
| <i>ENET_TDES0_TTSEN</i>   | transmit timestamp function enable   |
| <i>ENET_TDES0_DPAD</i>  | disable adding pad   |

|  |                         |
|--|-------------------------|
| <i>ENET_TDES0_DCRC</i>                   | disable CRC             |
| <i>ENET_TDES0_FSG</i>                    | first segment           |
| <i>ENET_TDES0_LSG</i>                    | last segment            |
| <i>ENET_TDES0_INTC</i>                   | interrupt on completion |
| <i>ENET_TDES0_DAV</i>                    | DAV bit                 |
| When type of parameter <b>desc</b> is RX |                         |
| <i>ENET_RDES0_DAV</i>                    | descriptor available    |
| <b>Output parameter{out}</b>             |                         |
| -  | -                       |
| <b>Return value</b>                      |                         |
| -  | -                       |

Example:

```
/* clear VLAN frame bit flag of ENET DMA descriptor */

enet_descriptors_struct p_txdesc;

enet_desc_flag_clear(p_txdesc, ENET_TDES0_TCHM);
```

### **enet\_desc\_receive\_complete\_bit\_enable**

The description of `enet_desc_receive_complete_bit_enable` is shown as below:

**Table 3-309. Function `enet_desc_receive_complete_bit_enable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_desc_receive_complete_bit_enable   |
| <b>Function prototype</b>    | void enet_desc_receive_complete_bit_enable(enet_descriptors_struct *desc);  |
| <b>Function descriptions</b> | when receiving the completed, set RS bit in ENET_DMA_STAT register will set   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>desc</b>                  | the descriptor pointer which users want to get flag, the structure members can refer to <a href="#">Table 3-238. Structure <code>enet_descriptors_struct</code></a> |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable receive complete bit */

enet_descriptors_struct p_rxdesc;

enet_desc_receive_complete_bit_enable(p_rxdesc);
```

### **enet\_desc\_receive\_complete\_bit\_disable**

The description of enet\_desc\_receive\_complete\_bit\_disable is shown as below:

**Table 3-310. Function enet\_desc\_receive\_complete\_bit\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_desc_receive_complete_bit_disable   |
| <b>Function prototype</b>    | void enet_desc_receive_complete_bit_disable(enet_descriptors_struct *desc);  |
| <b>Function descriptions</b> | when receiving the completed, set RS bit in ENET_DMA_STAT register will not set  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>desc</b>                  | the descriptor pointer which users want to get flag, the structure members can refer to <a href="#">Table 3-238. Structure enet_descriptors_struct</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable receive complete bit */
enet_descriptors_struct p_rxdesc;
enet_desc_receive_complete_bit_disable(p_rxdesc);
```

### **enet\_rxframe\_drop**

The description of enet\_rxframe\_drop is shown as below:

**Table 3-311. Function enet\_rxframe\_drop**

|                              |                               |
|------------------------------|-------------------------------|
| <b>Function name</b>         | enet_rxframe_drop             |
| <b>Function prototype</b>    | void enet_rxframe_drop(void); |
| <b>Function descriptions</b> | drop current receive frame    |
| <b>Precondition</b>          | -                             |
| <b>The called functions</b>  | -                             |
| <b>Input parameter{in}</b>   |                               |
| -                            | -                             |
| <b>Output parameter{out}</b> |                               |
| -                            | -                             |
| <b>Return value</b>          |                               |
| -                            | -                             |

Example:

```
/* drop current receive frame */
```

```
enet_rxframe_drop();
```

### **enet\_dma\_feature\_enable**

The description of enet\_dma\_feature\_enable is shown as below:

**Table 3-312. Function enet\_dma\_feature\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_dma_feature_enable  |
| <b>Function prototype</b>    | void enet_dma_feature_enable(uint32_t feature);  |
| <b>Function descriptions</b> | enable DMA feature   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>feature</b>               | the feature of DMA mode<br>one or more parameters can be selected which are shown as below |
| <i>ENET_NO_FLUSH_RXFRAME</i> | RxDMA does not flushes frames function   |
| <i>ENET_SECONDFRAME_OPT</i>  | TxDMA controller operate on second frame function  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable RxDMA does not flushes frames function */
enet_dma_feature_enable(ENET_NO_FLUSH_RXFRAME);
```

### **enet\_dma\_feature\_disable**

The description of enet\_dma\_feature\_disable is shown as below:

**Table 3-313. Function enet\_dma\_feature\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_dma_feature_disable   |
| <b>Function prototype</b>    | void enet_dma_feature_disable(uint32_t feature);   |
| <b>Function descriptions</b> | disable DMA feature  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>feature</b>               | the feature of DMA mode<br>one or more parameters can be selected which are shown as below |
| <i>ENET_NO_FLUSH_RXFRAME</i> | RxDMA does not flushes frames function   |
| <i>ENET_SECONDFRAM</i>       | TxDMA controller operate on second frame function  |

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

Example:

```
/* disable RxDMA does not flushes frames function */
enet_dma_feature_disable(ENET_NO_FLUSH_RXFRAME);
```

### **enet\_ptp\_normal\_descriptors\_chain\_init**

The description of enet\_ptp\_normal\_descriptors\_chain\_init is shown as below:

**Table 3-314. Function enet\_ptp\_normal\_descriptors\_chain\_init**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_ptp_normal_descriptors_chain_init   |
| <b>Function prototype</b>    | void enet_ptp_normal_descriptors_chain_init(enet_dmadirection_enum direction, enet_descriptors_struct *desc_ptptab);   |
| <b>Function descriptions</b> | initialize the DMA Tx/Rx descriptors's parameters in normal chain mode with PTP function   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>direction</b>             | the descriptors which users want to init<br>only one parameter can be selected which is shown as below   |
| <i>ENET_DMA_TX</i>           | DMA Tx descriptors   |
| <i>ENET_DMA_RX</i>           | DMA Rx descriptors   |
| <b>Input parameter{in}</b>   |  |
| <b>desc_ptptab</b>           | pointer to the first descriptor address of PTP Rx descriptor table, the structure members can refer to <a href="#"><u>Table 3-238. Structure enet_descriptors_struct</u></a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* initialize the DMA Rx descriptors's parameters in normal chain mode with PTP function */
enet_descriptors_struct ptp_rxdesc_tab[ENET_RXBUF_NUM];
enet_ptp_normal_descriptors_chain_init(ENET_DMA_RX, ptp_rxdesc_tab);
```

### **enet\_ptp\_normal\_descriptors\_ring\_init**

The description of enet\_ptp\_normal\_descriptors\_ring\_init is shown as below:

**Table 3-315. Function enet\_ptp\_normal\_descriptors\_ring\_init**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_ptp_normal_descriptors_ring_init   |
| <b>Function prototype</b>    | void enet_ptp_normal_descriptors_ring_init(enet_dmadirection_enum direction, enet_descriptors_struct *desc_ptptab);   |
| <b>Function descriptions</b> | initialize the DMA Tx/Rx descriptors's parameters in normal ring mode with PTP function   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| direction                    | the descriptors which users want to init<br>only one parameter can be selected which is shown as below  |
| ENET_DMA_TX                  | DMA Tx descriptors  |
| ENET_DMA_RX                  | DMA Rx descriptors  |
| <b>Input parameter{in}</b>   |   |
| desc_ptptab                  | pointer to the first descriptor address of PTP Rx descriptor table, the structure members can refer to <a href="#">Table 3-238. Structure enet_descriptors_struct</a> |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* initialize the DMA Rx descriptors's parameters in normal ring mode with PTP function */
enet_descriptors_struct ptp_rxdesc_tab[ENET_RXBUF_NUM];
enet_ptp_normal_descriptors_ring_init(ENET_DMA_RX, ptp_rxdesc_tab);
```

### **enet\_ptpframe\_receive\_normal\_mode**

The description of enet\_ptpframe\_receive\_normal\_mode is shown as below:

**Table 3-316. Function enet\_ptpframe\_receive\_normal\_mode**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_ptpframe_receive_normal_mode   |
| <b>Function prototype</b>    | ErrStatus enet_ptpframe_receive_normal_mode(uint8_t *buffer, uint32_t bufsize, uint32_t timestamp[]); |
| <b>Function descriptions</b> | receive a packet data with timestamp values to application buffer, when the DMA is in normal mode     |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |

|                              |  |
|------------------------------|--|
| <b>bufsize</b>               | the size of buffer which is the parameter in function  |
| <b>Output parameter{out}</b> |  |
| <b>timestamp</b>             | pointer to the table which stores the timestamp high and low   |
| <b>Output parameter{out}</b> |  |
| <b>buffer</b>                | pointer to the application buffer<br>if the input is NULL, user should copy data in application by himself |
| <b>Return value</b>          |  |
| <b>ErrStatus</b>             | ERROR or SUCCESS   |

Example:

```
/* receive a packet data with timestamp values to application buffer in DMA normal mode */

uint32_t rx_buffer[500];

uint32_t rx_buffer[500];

uint32_t time_stamp[2];

enet_ptpframe_receive_normal_mode(rx_buffer, 500, time_stamp);
```

### **enet\_ptpframe\_transmit\_normal\_mode**

The description of enet\_ptpframe\_transmit\_normal\_mode is shown as below:

**Table 3-317. Function enet\_ptpframe\_transmit\_normal\_mode**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_ptpframe_transmit_normal_mode  |
| <b>Function prototype</b>    | ErrStatus enet_ptpframe_transmit_normal_mode(uint8_t *buffer, uint32_t length, uint32_t timestamp[]);         |
| <b>Function descriptions</b> | send data with timestamp values in application buffer as a transmit packet,<br>when the DMA is in normal mode |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>buffer</b>                | pointer on the application buffer<br>if the input is NULL, user should copy data in application by himself    |
| <b>Input parameter{in}</b>   |   |
| <b>length</b>                | the length of frame data to be transmitted  |
| <b>Output parameter{out}</b> |   |
| <b>timestamp</b>             | pointer to the table which stores the timestamp high and low<br>if the input is NULL, timestamp is ignored    |
| <b>Return value</b>          |   |
| <b>ErrStatus</b>             | ERROR or SUCCESS  |

Example:

```
/* send data and timestamp values in application buffer as a transmit packet with DMA normal  
mode */
```

```

uint32_t tx_buffer[500];
uint32_t time_stamp[2];
enet_ptpframe_transmit_normal_mode(tx_buffer, 500, time_stamp);
  
```

### **enet\_wum\_filter\_register\_pointer\_reset**

The description of enet\_wum\_filter\_register\_pointer\_reset is shown as below:

**Table 3-318. Function enet\_wum\_filter\_register\_pointer\_reset**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_wum_filter_register_pointer_reset             |
| <b>Function prototype</b>    | void enet_wum_filter_register_pointer_reset(void); |
| <b>Function descriptions</b> | wakeup frame filter register pointer 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 wakeup frame filter register pointer */
enet_wum_filter_register_pointer_reset();
  
```

### **enet\_wum\_filter\_config**

The description of enet\_wum\_filter\_config is shown as below:

**Table 3-319. Function enet\_wum\_filter\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_wum_filter_config   |
| <b>Function prototype</b>    | void enet_wum_filter_config(uint32_t pdata[]);   |
| <b>Function descriptions</b> | set the remote wakeup frame registers  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>pdata</b>                 | pointer to buffer data which is written to remote wakeup frame registers (8 words total) |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* set the remote wakeup frame registers */
```

```
uint32_t wum_data[8];
enet_wum_filter_config(wum_data);
```

### **enet\_wum\_feature\_enable**

The description of enet\_wum\_feature\_enable is shown as below:

**Table 3-320. Function enet\_wum\_feature\_enable**

|                                    |   |
|------------------------------------|---|
| <b>Function name</b>               | enet_wum_feature_enable   |
| <b>Function prototype</b>          | void enet_wum_feature_enable(uint32_t feature);                             |
| <b>Function descriptions</b>       | enable wakeup management features   |
| <b>Precondition</b>                | -   |
| <b>The called functions</b>        | -   |
| <b>Input parameter{in}</b>         |   |
| <b>feature</b>                     | one or more parameters can be selected which are shown as below             |
| <i>ENET_WUM_POWER_DOWN</i>         | power down mode   |
| <i>ENET_WUM_MAGIC_PACKET_FRAME</i> | enable a wakeup event due to magic packet reception                         |
| <i>ENET_WUM_WAKE_UP_FRAME</i>      | enable a wakeup event due to wakeup frame reception                         |
| <i>ENET_WUM_GLOBAL_UNICAST</i>     | any received unicast frame passed filter is considered to be a wakeup frame |
| <b>Output parameter{out}</b>       |   |
| -                                  | -   |
| <b>Return value</b>                |   |
| -                                  | -   |

Example:

```
/* enable power down mode */
enet_wum_feature_enable(ENET_WUM_POWER_DOWN);
```

### **enet\_wum\_feature\_disable**

The description of enet\_wum\_feature\_disable is shown as below:

**Table 3-321. Function enet\_wum\_feature\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_wum_feature_disable                        |
| <b>Function prototype</b>    | void enet_wum_feature_disable(uint32_t feature) |
| <b>Function descriptions</b> | disable wakeup management features              |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |

| Input parameter{in}                |   |
|------------------------------------|---|
| <b>feature</b>                     | one or more parameters can be selected which are shown as below             |
| <i>ENET_WUM_MAGIC_PACKET_FRAME</i> | enable a wakeup event due to magic packet reception                         |
| <i>ENET_WUM_WAKE_UP_FRAME</i>      | enable a wakeup event due to wakeup frame reception                         |
| <i>ENET_WUM_GLOBAL_UNICAST</i>     | any received unicast frame passed filter is considered to be a wakeup frame |
| Output parameter{out}              |   |
| -                                  | -   |
| Return value                       |   |
| -                                  | -   |

Example:

```
/* disable power down mode */
enet_wum_feature_disable(ENET_WUM_POWER_DOWN);
```

### **enet\_msc\_counters\_reset**

The description of enet\_msc\_counters\_reset is shown as below:

**Table 3-322. Function enet\_msc\_counters\_reset**

| <b>Function name</b>         | enet_msc_counters_reset             |
|------------------------------|-------------------------------------|
| <b>Function prototype</b>    | void enet_msc_counters_reset(void); |
| <b>Function descriptions</b> | reset the MAC statistics counters   |
| <b>Precondition</b>          | -                                   |
| <b>The called functions</b>  | -                                   |
| Input parameter{in}          |                                     |
| -                            | -                                   |
| Output parameter{out}        |                                     |
| -                            | -                                   |
| Return value                 |                                     |
| -                            | -                                   |

Example:

```
/* reset the MAC statistics counters */
enet_msc_counters_reset();
```

### **enet\_msc\_feature\_enable**

The description of enet\_msc\_feature\_enable is shown as below:

**Table 3-323. Function enet\_msc\_feature\_enable**

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

|  |   |
|--|---|
| <b>Function prototype</b>                  | void enet_msc_feature_enable(uint32_t feature);                 |
| <b>Function descriptions</b>               | enable the MAC statistics counter features                      |
| <b>Precondition</b>                        | -   |
| <b>The called functions</b>                | -   |
| <b>Input parameter{in}</b>                 |   |
| <b>feature</b>                             | one or more parameters can be selected which are shown as below |
| <i>ENET_MSC_COUNTE<br/>R_STOP_ROLLOVER</i> | counter stop rollover   |
| <i>ENET_MSC_RESET_O<br/>N_READ</i>         | reset on read   |
| <i>ENET_MSC_COUNTE<br/>RS_FREEZE</i>       | MSC counter freeze  |
| <b>Output parameter{out}</b>               |   |
| -  | -   |
| <b>Return value</b>                        |   |
| -  | -   |

Example:

```
/* enable counter stop rollover function */
enet_msc_feature_enable(ENET_MSC_COUNTER_STOP_ROLLOVER);
```

### **enet\_msc\_feature\_disable**

The description of enet\_msc\_feature\_disable is shown as below:

**Table 3-324. Function enet\_msc\_feature\_disable**

|  |   |
|--|---|
| <b>Function name</b>                       | enet_msc_feature_disable  |
| <b>Function prototype</b>                  | void enet_msc_feature_disable(uint32_t feature);                |
| <b>Function descriptions</b>               | disable the MAC statistics counter features                     |
| <b>Precondition</b>                        | -   |
| <b>The called functions</b>                | -   |
| <b>Input parameter{in}</b>                 |   |
| <b>feature</b>                             | one or more parameters can be selected which are shown as below |
| <i>ENET_MSC_COUNTE<br/>R_STOP_ROLLOVER</i> | counter stop rollover   |
| <i>ENET_MSC_RESET_O<br/>N_READ</i>         | reset on read   |
| <i>ENET_MSC_COUNTE<br/>RS_FREEZE</i>       | MSC counter freeze  |
| <b>Output parameter{out}</b>               |   |
| -  | -   |
| <b>Return value</b>                        |   |
| -  | -   |

Example:

```
/* disable counter stop rollover function */

enet_msc_feature_disable(ENET_MSC_COUNTER_STOP_ROLLOVER);
```

### **enet\_msc\_counters\_get**

The description of enet\_msc\_counters\_get is shown as below:

**Table 3-325. Function enet\_msc\_counters\_get**

|                                 |  |
|---------------------------------|--|
| <b>Function name</b>            | enet_msc_counters_get  |
| <b>Function prototype</b>       | uint32_t enet_msc_counters_get(enet_msc_counter_enum counter);                               |
| <b>Function descriptions</b>    | get MAC statistics counter   |
| <b>Precondition</b>             | -  |
| <b>The called functions</b>     | -  |
| <b>Input parameter{in}</b>      |  |
| <b>counter</b>                  | MSC counters which is selected<br>only one parameter can be selected which is shown as below |
| <i>ENET_MSC_TX_SCCN<br/>T</i>   | MSC transmitted good frames after a single collision counter                                 |
| <i>ENET_MSC_TX_MSC<br/>CNT</i>  | MSC transmitted good frames after more than a single collision counter                       |
| <i>ENET_MSC_TX_TGFC<br/>NT</i>  | MSC transmitted good frames counter  |
| <i>ENET_MSC_RX_RFCE<br/>CNT</i> | MSC received frames with CRC error counter   |
| <i>ENET_MSC_RX_RFAE<br/>CNT</i> | MSC received frames with alignment error counter   |
| <i>ENET_MSC_RX_RGU<br/>FCNT</i> | MSC received good unicast frames counter   |
| <b>Output parameter{out}</b>    |  |
| -                               | -  |
| <b>Return value</b>             |  |
| <b>uint32_t</b>                 | the MSC counter value  |

Example:

```
/* get MSC transmitted good frames after a single collision counter value*/

uin32_t reval;

reval = enet_msc_counters_get(ENET_MSC_TX_SCCNT);
```

### **enet\_ptp\_subsecond\_2\_nanosecond**

The description of enet\_ptp\_subsecond\_2\_nanosecond is shown as below:

**Table 3-326. Function enet\_ptp\_subsecond\_2\_nanosecond**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_ptp_subsecond_2_nanosecond                               |
| <b>Function prototype</b>    | uint32_t enet_ptp_subsecond_2_nanosecond(uint32_t subsecond); |
| <b>Function descriptions</b> | change subsecond to nanosecond                                |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>subsecond</b>             | subsecond value (0 – 0x7FFF FFFF)                             |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>uint32_t</b>              | the nanosecond value (0 -- 499999999)                         |

Example:

```
/* change subsecond to nanosecond */

uint32_t reval;

reval = enet_ptp_subsecond_2_nanosecond(50);
```

### **enet\_ptp\_nanosecond\_2\_subsecond**

The description of enet\_ptp\_nanosecond\_2\_subsecond is shown as below:

**Table 3-327. Function enet\_ptp\_nanosecond\_2\_subsecond**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_ptp_nanosecond_2_subsecond                                |
| <b>Function prototype</b>    | uint32_t enet_ptp_nanosecond_2_subsecond(uint32_t nanosecond); |
| <b>Function descriptions</b> | change nanosecond to subsecond                                 |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>nanosecond</b>            | nanosecond value (0 -- 499999999)                              |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>uint32_t</b>              | the subsecond value (0 – 0x7FFF FFFF)                          |

Example:

```
/* change nanosecond to subsecond */

uint32_t reval;

reval = enet_ptp_nanosecond_2_subsecond(50);
```

### **enet\_ptp\_feature\_enable**

The description of enet\_ptp\_feature\_enable is shown as below:

**Table 3-328. Function enet\_ptp\_feature\_enable**

|  |   |
|--|---|
| <b>Function name</b>                       | enet_ptp_feature_enable   |
| <b>Function prototype</b>                  | void enet_ptp_feature_enable(uint32_t feature);   |
| <b>Function descriptions</b>               | enable the PTP features   |
| <b>Precondition</b>                        | -   |
| <b>The called functions</b>                | -   |
| <b>Input parameter{in}</b>                 |   |
| <b>feature</b>                             | the feature of ENET PTP mode<br>one or more parameters can be selected which are shown as below |
| <i>ENET_RXTX_TIMESTAMP</i><br><i>AMP</i>   | timestamp function for transmit and receive frames  |
| <i>ENET_PTP_TIMESTAMP</i><br><i>MP_INT</i> | timestamp interrupt trigger   |
| <b>Output parameter{out}</b>               |   |
| -  | -   |
| <b>Return value</b>                        |   |
| -  | -   |

Example:

```
/* enable PTP function for all received frames */
enet_ptp_feature_enable(ENET_RXTX_TIMESTAMP);
```

### **enet\_ptp\_feature\_disable**

The description of enet\_ptp\_feature\_disable is shown as below:

**Table 3-329. Function enet\_ptp\_feature\_disable**

|  |   |
|--|---|
| <b>Function name</b>                       | enet_ptp_feature_disable  |
| <b>Function prototype</b>                  | void enet_ptp_feature_disable(uint32_t feature);  |
| <b>Function descriptions</b>               | disable the PTP features  |
| <b>Precondition</b>                        | -   |
| <b>The called functions</b>                | -   |
| <b>Input parameter{in}</b>                 |   |
| <b>feature</b>                             | the feature of ENET PTP mode<br>one or more parameters can be selected which are shown as below |
| <i>ENET_RXTX_TIMESTAMP</i><br><i>AMP</i>   | timestamp function for transmit and receive frames  |
| <i>ENET_PTP_TIMESTAMP</i><br><i>MP_INT</i> | timestamp interrupt trigger   |
| <b>Output parameter{out}</b>               |   |

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

Example:

```
/* disable PTP function for all received frames */

enet_ptp_feature_disable(ENET_RXTX_TIMESTAMP);
```

### **enet\_ptp\_timestamp\_function\_config**

The description of enet\_ptp\_timestamp\_function\_config is shown as below:

**Table 3-330. Function enet\_ptp\_timestamp\_function\_config**

|                                |  |
|--------------------------------|--|
| <b>Function name</b>           | enet_ptp_timestamp_function_config   |
| <b>Function prototype</b>      | ErrStatus enet_ptp_timestamp_function_config(enet_ptp_function_enum func); |
| <b>Function descriptions</b>   | configure the PTP timestamp function                                       |
| <b>Precondition</b>            | -  |
| <b>The called functions</b>    | -  |
| <b>Input parameter{in}</b>     |  |
| <b>func</b>                    | only one parameter can be selected which is shown as below                 |
| <i>ENET_PTP_ADDEND_UPDATE</i>  | addend register update   |
| <i>ENET_PTP_SYSTIME_UPDATE</i> | timestamp update   |
| <i>ENET_PTP_SYSTIME_INIT</i>   | timestamp initialize   |
| <i>ENET_PTP_FINEMOD_E</i>      | the system timestamp uses the fine method for updating                     |
| <i>ENET_PTP_COARSE_MODE</i>    | the system timestamp uses the coarse method for updating                   |
| <b>Output parameter{out}</b>   |  |
| -                              | -  |
| <b>Return value</b>            |  |
| <b>ErrStatus</b>               | SUCCESS or ERROR   |

Example:

```
/* config addend register update function */

ErrStatus rval;

rval = enet_ptp_timestamp_function_config(ENET_PTP_ADDEND_UPDATE);
```

### **enet\_ptp\_subsecond\_increment\_config**

The description of enet\_ptp\_subsecond\_increment\_config is shown as below:

**Table 3-331. Function enet\_ptp\_subsecond\_increment\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_ptp_subsecond_increment_config  |
| <b>Function prototype</b>    | void enet_ptp_subsecond_increment_config(uint32_t subsecond);  |
| <b>Function descriptions</b> | configure system time subsecond increment value  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>subsecond</b>             | the value will be added to the subsecond value of system time, this value must be between 0 and 0xFF |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure 0x1F as system time subsecond increment value */
enet_ptp_subsecond_increment_config(0x1F);
```

### **enet\_ptp\_timestamp\_addend\_config**

The description of enet\_ptp\_timestamp\_addend\_config is shown as below:

**Table 3-332. Function enet\_ptp\_timestamp\_addend\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_ptp_timestamp_addend_config  |
| <b>Function prototype</b>    | void enet_ptp_timestamp_addend_config(uint32_t add);  |
| <b>Function descriptions</b> | adjusting the clock frequency only in fine update mode  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>add</b>                   | the value will be added to the accumulator register to achieve time synchronization (0 – 0xFFFF FFFF) |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* added 0x1FFF to the accumulator register */
enet_ptp_timestamp_addend_config(0x1FFF);
```

### **enet\_ptp\_timestamp\_update\_config**

The description of enet\_ptp\_timestamp\_update\_config is shown as below:

**Table 3-333. Function enet\_ptp\_timestamp\_update\_config**

|                                     |   |
|-------------------------------------|---|
| <b>Function name</b>                | enet_ptp_timestamp_update_config  |
| <b>Function prototype</b>           | void enet_ptp_timestamp_update_config(uint32_t sign, uint32_t second, uint32_t subsecond);                |
| <b>Function descriptions</b>        | initialize or add/subtract to second of the system time   |
| <b>Precondition</b>                 | -   |
| <b>The called functions</b>         | -   |
| <b>Input parameter{in}</b>          |   |
| <b>sign</b>                         | timestamp update positive or negative sign,<br>only one parameter can be selected which is shown as below |
| <i>ENET_PTP_ADD_TO_TIME</i>         | update value is added to system time  |
| <i>ENET_PTP_SUBSTRACT_FROM_TIME</i> | timestamp update value is subtracted from system time   |
| <b>Input parameter{in}</b>          |   |
| <b>second</b>                       | initializing or adding/subtracting to second of the system time (0 – 0xFFFF FFFF)                         |
| <b>Input parameter{in}</b>          |   |
| <b>subsecond</b>                    | the current subsecond of the system time with 0.46 ns accuracy (0 – 0x7FFF FFFF)                          |
| <b>Output parameter{out}</b>        |   |
| -                                   | -   |
| <b>Return value</b>                 |   |
| -                                   | -   |

Example:

```
/* initialize system time with timestamp update value */
enet_ptp_timestamp_update_config(ENET_PTP_ADD_TO_TIME, 0, 0);
```

### **enet\_ptp\_expected\_time\_config**

The description of enet\_ptp\_expected\_time\_config is shown as below:

**Table 3-334. Function enet\_ptp\_expected\_time\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_ptp_expected_time_config   |
| <b>Function prototype</b>    | void enet_ptp_expected_time_config(uint32_t second, uint32_t nanosecond); |
| <b>Function descriptions</b> | configure the expected target time  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |

| Input parameter{in}   |  |
|-----------------------|--|
| <b>second</b>         | the expected target second time (0 – 0xFFFF FFFF)              |
| Input parameter{in}   |  |
| <b>nanosecond</b>     | the expected target nanosecond time (signed) (0 – 0xFFFF FFFF) |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

Example:

```
/* configure the expected target time */
enet_ptp_expected_time_config(2000, 0);
```

### **enet\_ptp\_system\_time\_get**

The description of enet\_ptp\_system\_time\_get is shown as below:

**Table 3-335. Function enet\_ptp\_system\_time\_get**

| <b>Function name</b>         | enet_ptp_system_time_get   |
|------------------------------|--|
| <b>Function prototype</b>    | void enet_ptp_system_time_get(enet_ptp_systime_struct *systime_struct);  |
| <b>Function descriptions</b> | get the current system time  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| Input parameter{in}          |  |
| -                            | -  |
| Output parameter{out}        |  |
| <b>systime_struct</b>        | pointer to a enet_ptp_systime_struct structure which contains parameters of PTP system time, the structure members can refer to <a href="#">Table 3-239. Structure enet_ptp_systime_struct</a> |
| Return value                 |  |
| -                            | -  |

Example:

```
/* get the current system time */
enet_ptp_systime_struct systime;
enet_ptp_system_time_get(&systime);
```

### **enet\_ptp\_start**

The description of enet\_ptp\_start is shown as below:

**Table 3-336. Function enet\_ptp\_start**

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

|                                 |  |
|---------------------------------|--|
| <b>Function prototype</b>       | void enet_ptp_start(int32_t updatemethod, uint32_t init_sec, uint32_t init_subsec, uint32_t carry_cfg, uint32_t accuracy_cfg);   |
| <b>Function descriptions</b>    | configure and start PTP timestamp counter  |
| <b>Precondition</b>             | -  |
| <b>The called functions</b>     | enet_interrupt_disable/ enet_ptp_feature_enable/<br>enet_ptp_subsecond_increment_config/<br>enet_ptp_timestamp_addend_config/ enet_ptp_timestamp_function_config/<br>enet_ptp_flag_get/ enet_ptp_timestamp_update_config |
| <b>Input parameter{in}</b>      |  |
| <b>updatemethod</b>             | method for updating  |
| <i>ENET_PTP_FINEMOD<br/>E</i>   | fine correction method   |
| <i>ENET_PTP_COARSE<br/>MODE</i> | coarse correction method   |
| <b>Input parameter{in}</b>      |  |
| <b>init_sec</b>                 | second value for initializing system time (0 – 0xFFFF FFFF)  |
| <b>Input parameter{in}</b>      |  |
| <b>init_subsec</b>              | subsecond value for initializing system time (0 – 0x7FFF FFFF)   |
| <b>Input parameter{in}</b>      |  |
| <b>carry_cfg</b>                | the value to be added to the accumulator register (in fine method is used) (0 – 0xFFFF FFFF)   |
| <b>Input parameter{in}</b>      |  |
| <b>accuracy_cfg</b>             | the value to be added to the subsecond value of system time (0 – 0xFF)   |
| <b>Output parameter{out}</b>    |  |
| -                               | -  |
| <b>Return value</b>             |  |
| -                               | -  |

Example:

```
/* gconfigure and start PTP timestamp counter*/
enet_ptp_start(ENET_PTP_FINEMODE, 0, 0, 50, 0);
```

### **enet\_ptp\_finecorrection\_adjfreq**

The description of enet\_ptp\_finecorrection\_adjfreq is shown as below:

**Table 3-337. Function enet\_ptp\_finecorrection\_adjfreq**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_ptp_finecorrection_adjfreq                                      |
| <b>Function prototype</b>    | void enet_ptp_finecorrection_adjfreq(int32_t carry_cfg);             |
| <b>Function descriptions</b> | adjust frequency in fine method by configure addend register         |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | enet_ptp_timestamp_addend_config/ enet_ptp_timestamp_function_config |
| <b>Input parameter{in}</b>   |  |

|                              |   |
|------------------------------|---|
| <b>carry_cfg</b>             | the value to be added to the accumulator register (0 – 0xFFFF FFFF) |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* adjust frequency in fine method by configure addend register */

enet_ptp_finecorrection_adjfreq(50);
```

### **enet\_ptp\_coarsecorrection\_systime\_update**

The description of enet\_ptp\_coarsecorrection\_systime\_update is shown as below:

**Table 3-338. Function enet\_ptp\_coarsecorrection\_systime\_update**

|  |  |
|--|--|
| <b>Function name</b>   | enet_ptp_coarsecorrection_systime_update   |
| <b>Function prototype</b>  | void enet_ptp_coarsecorrection_systime_update(enet_ptp_systime_struct *systime_struct);  |
| <b>Function descriptions</b>   | update system time in coarse method  |
| <b>Precondition</b>  | -  |
| <b>The called functions</b>  |  |
| enet_ptp_nanosecond_2_subsecond/ enet_ptp_timestamp_update_config/<br>enet_ptp_timestamp_function_config/ enet_ptp_flag_get/<br>enet_ptp_timestamp_addend_config |  |
| <b>Input parameter{in}</b>   |  |
| <b>systime_struct</b>  | pointer to a enet_ptp_systime_struct structure which contains parameters of<br>PTP system time, the structure members can refer to <a href="#">Table 3-239.</a><br><a href="#">Structure enet_ptp_systime_struct</a> |
| <b>Output parameter{out}</b>   |  |
| -  | -  |
| <b>Return value</b>  |  |
| -  | -  |

Example:

```
/* update system time in coarse method */

enet_ptp_systime_struct systime_struct;

systime_struct.second = 0x0000FFFF;
systime_struct.nanosecond = 0;
systime_struct.sign = ENET_PTP_TIME_POSITIVE;
enet_ptp_coarsecorrection_systime_update(&systime_struct);
```

### **enet\_ptp\_finecorrection\_settime**

The description of enet\_ptp\_finecorrection\_settime is shown as below:

**Table 3-339. Function enet\_ptp\_finecorrection\_settime**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_ptp_finecorrection_settime  |
| <b>Function prototype</b>    | void enet_ptp_finecorrection_settime(enet_ptp_systime_struct *<br>systime_struct);   |
| <b>Function descriptions</b> | set system time in fine method   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | enet_ptp_nanosecond_2_subsecond/ enet_ptp_timestamp_update_config/<br>enet_ptp_timestamp_function_config/ enet_ptp_flag_get  |
| <b>Input parameter{in}</b>   |  |
| <b>systime_struct</b>        | pointer to a enet_ptp_systime_struct structure which contains parameters of<br>PTP system time, the structure members can refer to <a href="#">Table 3-239.</a><br><a href="#">Structure enet_ptp_systime_struct</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* set system time in fine method */

enet_ptp_systime_struct systime_struct;

systime_struct.second = 0x0000FFFF;
systime_struct.nanosecond = 0;
systime_struct.sign = ENET_PTP_TIME_POSITIVE;
enet_ptp_finecorrection_settime(&systime_struct);
```

### **enet\_ptp\_flag\_get**

The description of enet\_ptp\_flag\_get is shown as below:

**Table 3-340. Function enet\_ptp\_flag\_get**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | enet_ptp_flag_get                            |
| <b>Function prototype</b>    | FlagStatus enet_ptp_flag_get(uint32_t flag); |
| <b>Function descriptions</b> | get the ptp flag status                      |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>flag</b>                  | ptp flag status to be checked                |
| <b>ENET_PTP_ADDEND_</b>      | addend register update                       |

|                                |                      |
|--------------------------------|----------------------|
| <i>UPDATE</i>                  |                      |
| <i>ENET_PTP_SYSTIME_UPDATE</i> | timestamp update     |
| <i>ENET_PTP_SYSTIME_INIT</i>   | timestamp initialize |
| <b>Output parameter{out}</b>   |                      |
| -                              | -                    |
| <b>Return value</b>            |                      |
| <b>FlagStatus</b>              | SET or RESET         |

Example:

```

/* get the ptp flag status */

FlagStatus reval;

reval = enet_ptp_flag_get(ENET_PTP_ADDEND_UPDATE);

```

### **enet\_initpara\_reset**

The description of **enet\_initpara\_reset** is shown as below:

**Table 3-341. Function enet\_initpara\_reset**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | enet_initpara_reset   |
| <b>Function prototype</b>    | void enet_initpara_reset(void);   |
| <b>Function descriptions</b> | reset the ENET initpara struct, call it before using enet_initpara_config() |
| <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 ENET initpara struct */

enet_initpara_reset();

```

## **3.12. 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 [3.12.1](#), the EXMC firmware functions are introduced in chapter [3.12.2](#).

### 3.12.1. Descriptions of Peripheral registers

EXMC registers are listed in the table shown as below:

**Table 3-342. EXMC Registers**

| Registers                       | Descriptions  |
|---------------------------------|---|
| EXMC_SNCTLx<br>(x=0, 1, 2, 3)   | SRAM/NOR Flash control registers                                  |
| EXMC_SNTCFGx<br>(x=0, 1, 2, 3)  | SRAM/NOR Flash timing configuration registers                     |
| EXMC_SNWTCFGx<br>(x=0, 1, 2, 3) | SRAM/NOR Flash write timing configuration registers               |
| EXMC_NPCTLx<br>(x=1, 2, 3)      | NAND Flash/PC card control registers                              |
| EXMC_NPINTENx<br>(x=1, 2, 3)    | NAND Flash/PC card interrupt enable registers                     |
| EXMC_NPCTCFGx<br>(x=1, 2, 3)    | NAND Flash/PC card common space timing configuration registers    |
| EXMC_NPATCFGx<br>(x=1, 2, 3)    | NAND Flash/PC card attribute space timing configuration registers |
| EXMC_PIOTCFG3                   | PC Card I/O space timing configuration register                   |
| EXMC_NECCx<br>(x=1, 2)          | NAND Flash ECC registers  |
| EXMC_SDCTLx(x=0, 1)             | SDRAM control registers   |
| EXMC_SDTCFGx(x=0, 1)            | SDRAM timing configuration registers                              |
| EXMC_SDCMD                      | SDRAM command register  |
| EXMC_SDARI                      | SDRAM auto-refresh interval register                              |
| EXMC_SDSTAT                     | SDRAM status register   |
| EXMC_SDRSCTL                    | SDRAM read sample control register                                |
| EXMC_SINIT                      | SPI initialization register                                       |
| EXMC_SRCMD                      | SPI read command register   |
| EXMC_SWCMD                      | SPI write command register  |
| EXMC_SIDL                       | SPI ID low register   |
| EXMC_SIDH                       | SPI ID high register  |

### 3.12.2. Descriptions of Peripheral functions

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

**Table 3-343. EXMC firmware function**

| Function name       | Function description              |
|---------------------|-----------------------------------|
| exmc_norsram_deinit | deinitialize EXMC NOR/SRAM region |

| <b>Function name</b>                | <b>Function description</b>  |
|-------------------------------------|--|
| exmc_norsram_struct_para_init       | initialize exmc_norsram_parameter_struct with the default values       |
| exmc_norsram_init                   | initialize EXMC NOR/SRAM region  |
| exmc_norsram_enable                 | enable EXMC NOR/PSRAM bank region                                      |
| exmc_norsram_disable                | disable EXMC NOR/PSRAM bank region                                     |
| exmc_nand_deinit                    | deinitialize EXMC NAND bank  |
| exmc_nand_struct_para_init          | initialize exmc_nand_parameter_struct with the default values          |
| exmc_nand_init                      | initialize EXMC NAND bank  |
| exmc_nand_enable                    | enable EXMC NAND bank  |
| exmc_nand_disable                   | disable EXMC NAND bank   |
| exmc_nand_ecc_config                | enable or disable the EXMC NAND ECC function                           |
| exmc_ecc_get                        | get the EXMC ECC value   |
| exmc_pccard_deinit                  | deinitialize EXMC PC card bank   |
| exmc_pccard_struct_para_init        | initialize exmc_pccard_parameter_struct with the default values        |
| exmc_pccard_init                    | initialize EXMC PC card bank   |
| exmc_pccard_enable                  | enable PC card bank  |
| exmc_pccard_disable                 | disable PC card bank   |
| exmc_sdram_deinit                   | deinitialize EXMC SDRAM device   |
| exmc_sdram_struct_para_init         | initialize exmc_sdram_init_struct with the default values              |
| exmc_sdram_init                     | initialize EXMC SDRAM device   |
| exmc_sdram_struct_command_para_init | initialize exmc_sdram_command_parameter_struct with the default values |
| exmc_sdram_command_config           | configure the SDRAM memory command                                     |
| exmc_sdram_refresh_count_set        | set auto-refresh interval  |
| exmc_sdram_autorefresh_number_set   | set the number of successive auto-refresh command                      |
| exmc_sdram_write_protection_config  | configure the write protection function                                |
| exmc_sdram_bankstatus_get           | get the status of SDRAM device0 or device1                             |
| exmc_sdram_readsampel_config        | configure the delayed sample clock of read data                        |
| exmc_sdram_readsampel_enable        | enable read sample   |
| exmc_sdram_readsampel_disable       | disable read sample  |
| exmc_sqippsram_deinit               | deinitialize EXMC SQPIPSRAM  |
| exmc_sqippsram_struct_para_init     | initialize exmc_sqippsram_init_struct with the default values          |
| exmc_sqippsram_init                 | initialize EXMC SQPIPSRAM  |
| exmc_sqippsram_read_command_set     | set the read command   |
| exmc_sqippsram_write_command_set    | set the write command  |
| exmc_sqippsram_read_id_command_send | send SPI read ID command   |
| exmc_sqippsram_write_cmd_send       | send SPI special command which does not have address and data phase    |
| exmc_sqippsram_low_id_get           | get the EXMC SPI ID low data   |

| Function name                          | Function description  |
|--|---|
| exmc_sqipssram_high_id_get             | get the EXMC SPI ID high data                                       |
| exmc_sqipssram_send_command_status_get | get the bit value of EXMC send write command bit or read ID command |
| exmc_flag_get                          | get EXMC flag status  |
| exmc_flag_clear                        | clear EXMC flag   |
| exmc_interrupt_enable                  | enable EXMC interrupt   |
| exmc_interrupt_disable                 | disable EXMC interrupt  |
| exmc_interrupt_flag_get                | get EXMC interrupt flag   |
| exmc_interrupt_flag_clear              | clear EXMC interrupt flag   |

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

Table 3-344. 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_setuptime    | configure the data setup time, asynchronous access mode valid   |
| asyn_address_holdtime  | configure the address hold time, asynchronous access mode valid |
| asyn_address_setuptime | configure the data setup time, asynchronous access mode valid   |

### Structure exmc\_norsram\_parameter\_struct

Table 3-345. Structure exmc\_norsram\_parameter\_struct

| Member name      | Function description   |
|------------------|--|
| norsram_region   | select the region of EXMC NOR/SRAM bank                            |
| write_mode       | the write mode, synchronous mode or asynchronous mode              |
| extended_mode    | enable or disable the extended mode                                |
| asyn_wait        | enable or disable the asynchronous wait function                   |
| nwait_signal     | enable or disable the NWAIT signal while in synchronous burst mode |
| memory_write     | enable or disable the write operation                              |
| nwait_config     | NWAIT signal configuration   |
| wrap_burst_mode  | enable or disable the wrap burst mode                              |
| nwait_polarity   | specifies the polarity of NWAIT signal from memory                 |
| burst_mode       | enable or disable the burst mode                                   |
| databus_width    | specifies the databus width of external memory                     |
| memory_type      | specifies the type of external memory                              |
| address_data_mux | specifies whether the data bus and address bus are multiplexed     |

| Member name       | Function description   |
|-------------------|--|
| read_write_timing | 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. The structure members can refer to <a href="#">Table 3-344. Structure exmc_norsram_timing_parameter_struct</a> |
| write_timing      | timing parameters for write when the extended mode is used. The structure members can refer to <a href="#">Table 3-344. Structure exmc_norsram_timing_parameter_struct</a>   |

### Structure exmc\_nand\_pccard\_timing\_parameter\_struct

**Table 3-346. Structure exmc\_nand\_pccard\_timing\_parameter\_struct**

| Member name     | Function description   |
|-----------------|--|
| databus_hiztime | configure the databus HiZ time for write operation                         |
| holdtime        | configure the address hold time(or the data hold time for write operation) |
| waittime        | configure the minimum wait time  |
| setuptime       | configure the address setup time   |

### Structure exmc\_nand\_parameter\_struct

**Table 3-347. Structure exmc\_nand\_parameter\_struct**

| Member name            | Function description   |
|------------------------|--|
| nand_bank              | select the bank of NAND  |
| ecc_size               | the page size for the ECC calculation  |
| atr_latency            | configure the latency of ALE low to RB low   |
| ctr_latency            | configure the latency of CLE low to RB low   |
| ecc_logic              | enable or disable the ECC calculation logic  |
| databus_width          | the NAND flash databus width   |
| wait_feature           | enable or disable the wait feature   |
| common_space_timing    | the timing parameters for NAND flash common space. The structure members can refer to <a href="#">Table 3-346. Structure exmc_nand_pccard_timing_parameter_struct</a>    |
| attribute_space_timing | the timing parameters for NAND flash attribute space. The structure members can refer to <a href="#">Table 3-346. Structure exmc_nand_pccard_timing_parameter_struct</a> |

### Structure exmc\_pccard\_parameter\_struct

**Table 3-348. Structure exmc\_pccard\_parameter\_struct**

| Member name         | Function description   |
|---------------------|--|
| atr_latency         | configure the latency of ALE low to RB low                               |
| ctr_latency         | configure the latency of CLE low to RB low                               |
| wait_feature        | enable or disable the wait feature                                       |
| common_space_timing | the timing parameters for NAND flash common space. The structure members |

| Member name            | Function description   |
|------------------------|--|
| ng                     | can refer to <a href="#">Table 3-346. Structure exmc_nand_pccard_timing_parameter_struct</a>   |
| attribute_space_timing | the timing parameters for NAND flash attribute space. The structure members can refer to <a href="#">Table 3-346. Structure exmc_nand_pccard_timing_parameter_struct</a> |
| io_space_timing        | the timing parameters for NAND flash IO space. The structure members can refer to <a href="#">Table 3-346. Structure exmc_nand_pccard_timing_parameter_struct</a>        |

### Structure exmc\_sdrdram\_timing\_parameter\_struct

**Table 3-349. Structure exmc\_sdrdram\_timing\_parameter\_struct**

| Member name              | Function description                   |
|--------------------------|--|
| row_to_column_delay      | configure the row to column delay      |
| row_preadge_delay        | configure the row precharge delay      |
| write_recovery_delay     | configure the write recovery delay     |
| auto_refresh_delay       | configure the auto refresh delay       |
| row_address_select_delay | configure the row address select delay |
| exit_selfrefresh_delay   | configure the exit self-refresh delay  |
| load_mode_register_delay | configure the load mode register delay |

### Structure exmc\_sdrdram\_parameter\_struct

**Table 3-350. Structure exmc\_sdrdram\_parameter\_struct**

| Member name          | Function description  |
|----------------------|---|
| sdrdram_device       | device of SDRAM   |
| pipeline_read_delay  | the delay for reading data after CAS latency in HCLK clock cycles |
| burst_read_switch    | enable or disable the burst read                                  |
| sdclock_config       | the SDCLK memory clock for both SDRAM banks                       |
| write_protection     | enable or disable SDRAM bank write protection function            |
| cas_latency          | configure the SDRAM CAS latency                                   |
| internal_bank_number | the number internal banks   |
| data_width           | the databus width of SDRAM memory                                 |
| row_address_width    | the bit width of a row address                                    |
| column_address_width | the bit width of a column address                                 |

| Member name | Function description   |
|-------------|--|
| timing      | the timing parameters for write and read SDRAM. The structure members can refer to <a href="#">Table 3-349. Structure exmc_sdram_timing_parameter_struct</a> |

### Structure exmc\_sdram\_command\_parameter\_struct

**Table 3-351. Structure exmc\_sdram\_command\_parameter\_struct**

| Member name           | Function description   |
|-----------------------|--|
| mode_register_content | the SDRAM mode register content  |
| auto_refresh_number   | the number of successive auto-refresh cycles will be send when CMD = 011 |
| bank_select           | the bank which command will be sent to                                   |
| command               | the commands that will be sent to SDRAM                                  |

### Structure exmc\_sqipssram\_parameter\_struct

**Table 3-352. Structure exmc\_sqipssram\_parameter\_struct**

| Member name     | Function description                  |
|-----------------|---------------------------------------|
| sample_polarity | read data sample polarity             |
| id_length       | SPI PSRAM ID length                   |
| address_bits    | bit number of SPI PSRAM address phase |
| command_bits    | bit number of SPI PSRAM command phase |

### exmc\_norsram\_deinit

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

**Table 3-353. Function exmc\_norsram\_deinit**

|  |   |
|--|---|
| <b>Function name</b>                   | exmc_norsram_deinit                                     |
| <b>Function prototype</b>              | void exmc_norsram_deinit(uint32_t exmc_norsram_region); |
| <b>Function descriptions</b>           | deinitialize EXMC NOR/SRAM region                       |
| <b>Precondition</b>                    | -   |
| <b>The called functions</b>            | -   |
| <b>Input parameter{in}</b>             |   |
| exmc_norsram_region                    | select the region of bank0                              |
| EXMC_BANK0_NORS<br>RAM_REGIONx(x=0..3) | region x of bank0                                       |
| <b>Output parameter{out}</b>           |   |
| -                                      | -   |
| <b>Return value</b>                    |   |
| -                                      | -   |

Example:

```

/* deinitialize EXMC NOR/SRAM region 0 */

exmc_norsram_deinit(EXMC_BANK0_NORSRAM_REGION0);

```

### **exmc\_norsram\_struct\_para\_init**

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

**Table 3-354. Function exmc\_norsram\_struct\_para\_init**

|                                      |  |
|--------------------------------------|--|
| <b>Function name</b>                 | exmc_norsram_struct_para_init  |
| <b>Function prototype</b>            | void exmc_norsram_struct_para_init(exmc_norsram_parameter_struct*<br>exmc_norsram_init_struct);  |
| <b>Function descriptions</b>         | initialize exmc_norsram_parameter_struct with the default values   |
| <b>Precondition</b>                  | -  |
| <b>The called functions</b>          | -  |
| <b>Input parameter{in}</b>           |  |
| -                                    | -  |
| <b>Output parameter{out}</b>         |  |
| <b>exmc_norsram_init_st<br/>ruct</b> | structure for initialization, the structure members can refer to <a href="#">Table 3-345.</a><br><a href="#">Structure exmc_norsram_parameter_struct</a> |
| <b>Return value</b>                  |  |
| -                                    | -  |

Example:

```

/* initialize the structure nor_init_struct */

exmc_norsram_parameter_struct nor_init_struct;

exmc_norsram_struct_para_init(&nor_init_struct);

```

### **exmc\_norsram\_init**

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

**Table 3-355. 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 region  |
| <b>Precondition</b>                  | -  |
| <b>The called functions</b>          | -  |
| <b>Input parameter{in}</b>           |  |
| <b>exmc_norsram_init_st<br/>ruct</b> | structure for initialization, the structure members can refer to <a href="#">Table 3-345.</a><br><a href="#">Structure exmc_norsram_parameter_struct</a> |
| <b>Output parameter{out}</b>         |  |
| -                                    | -  |

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

Example:

```

/* initialize EXMC NOR/SRAM region */

exmc_norsram_parameter_struct nor_init_struct;

exmc_norsram_timing_parameter_struct nor_timing_init_struct;

/* configure timing parameter */

nor_timing_init_struct.asyn_access_mode = EXMC_ACCESS_MODE_A;

nor_timing_init_struct.syn_data_latency = EXMC_DATALAT_2_CLK;

nor_timing_init_struct.syn_clk_division = EXMC_SYN_CLOCK_RATIO_DISABLE;

nor_timing_init_struct.bus_latency = 1;

nor_timing_init_struct.asyn_data_setuptime = 7;

nor_timing_init_struct.asyn_address_holdtime = 2;

nor_timing_init_struct.asyn_address_setuptime = 5;

/* configure EXMC bus parameters */

nor_init_struct.norsram_region = EXMC_BANK0_NORSRAM_REGION0;

nor_init_struct.write_mode = EXMC_ASYN_WRITE;

nor_init_struct.extended_mode = DISABLE;

nor_init_struct.asyn_wait = DISABLE;

nor_init_struct.nwait_signal = DISABLE;

nor_init_struct.memory_write = ENABLE;

nor_init_struct.nwait_config = EXMC_NWAIT_CONFIG_BEFORE;

nor_init_struct.wrap_burst_mode = DISABLE;

nor_init_struct.nwait_polarity = EXMC_NWAIT_POLARITY_LOW;

nor_init_struct.burst_mode = DISABLE;

nor_init_struct.databus_width = EXMC_NOR_DATABUS_WIDTH_16B;

nor_init_struct.memory_type = EXMC_MEMORY_TYPE_NOR;

nor_init_struct.address_data_mux = ENABLE;

nor_init_struct.read_write_timing = &nor_timing_init_struct;

nor_init_struct.write_timing = &nor_timing_init_struct;

```

```
exmc_norsram_init(&nor_init_struct);
```

### **exmc\_norsram\_enable**

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

**Table 3-356. Function exmc\_norsram\_enable**

|  |  |
|--|--|
| <b>Function name</b>                           | exmc_norsram_enable                                    |
| <b>Function prototype</b>                      | void exmc_norsram_enable(uint32_t exmc_norsram_region) |
| <b>Function descriptions</b>                   | enable EXMC NOR/PSRAM bank region                      |
| <b>Precondition</b>                            | -  |
| <b>The called functions</b>                    | -  |
| <b>Input parameter{in}</b>                     |  |
| <b>exmc_norsram_region</b>                     | specifies the region of NOR/PSRAM bank                 |
| <i>EXMC_BANK0_NORS<br/>RAM_REGIONx(x=0..3)</i> | region x of bank0                                      |
| <b>Output parameter{out}</b>                   |  |
| -  | -  |
| <b>Return value</b>                            |  |
| -  | -  |

Example:

```
/* enable region 0 of bank0 */
exmc_norsram_enable(EXMC_BANK0_NORSRAM_REGION0);
```

### **exmc\_norsram\_disable**

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

**Table 3-357. Function exmc\_norsram\_disable**

|  |  |
|--|--|
| <b>Function name</b>                           | exmc_norsram_disable                                     |
| <b>Function prototype</b>                      | void exmc_norsram_disable(uint32_t exmc_norsram_region); |
| <b>Function descriptions</b>                   | disable EXMC NOR/PSRAM bank region                       |
| <b>Precondition</b>                            | -  |
| <b>The called functions</b>                    | -  |
| <b>Input parameter{in}</b>                     |  |
| <b>exmc_norsram_region</b>                     | specifies the region of NOR/PSRAM bank                   |
| <i>EXMC_BANK0_NORS<br/>RAM_REGIONx(x=0..3)</i> | region x of bank0  |
| <b>Output parameter{out}</b>                   |  |
| -  | -  |
| <b>Return value</b>                            |  |

Example:

```
/* disable region 0 of bank0 */
exmc_norsram_disable(EXMC_BANK0_NORSRAM_REGION0);
```

### **exmc\_nand\_deinit**

The description of exmc\_nand\_deinit is shown as below:

**Table 3-358. Function exmc\_nand\_deinit**

|  |   |
|--|---|
| <b>Function name</b>                     | exmc_nand_deinit                                |
| <b>Function prototype</b>                | void exmc_nand_deinit(uint32_t exmc_nand_bank); |
| <b>Function descriptions</b>             | deinitialize EXMC NAND bank                     |
| <b>Precondition</b>                      | -   |
| <b>The called functions</b>              | -   |
| <b>Input parameter{in}</b>               |   |
| <b>exmc_nand_bank</b>                    | select the bank of NAND                         |
| <i>EXMC_BANKx_NAND(</i><br><i>x=1,2)</i> | bank of NAND                                    |
| <b>Output parameter{out}</b>             |   |
| -  | -   |
| <b>Return value</b>                      |   |
| -  | -   |

Example:

```
/* deinitialize bank1 */
exmc_nand_deinit (EXMC_BANK1_NAND);
```

### **exmc\_nand\_struct\_para\_init**

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

**Table 3-359. Function exmc\_nand\_struct\_para\_init**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | exmc_nand_struct_para_init  |
| <b>Function prototype</b>    | void exmc_nand_struct_para_init(exmc_nand_parameter_struct*<br>exmc_nand_init_struct);        |
| <b>Function descriptions</b> | initialize exmc_nand_parameter_struct with the default values                                 |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| <b>exmc_nand_init_struct</b> | structure for initialization, the structure members can refer to <a href="#">Table 3-347.</a> |

|                     |   |
|---------------------|---|
| <b>t</b>            | <a href="#"><b>Structure exmc_nand_parameter_struct</b></a> |
| <b>Return value</b> |   |
| -                   | -   |

Example:

```
/* initialize the structure nand_init_struct */

exmc_nand_parameter_struct nand_init_struct;
exmc_norsram_struct_para_init(&nor_init_struct);
```

### **exmc\_nand\_init**

The description of exmc\_nand\_init is shown as below:

**Table 3-360. Function exmc\_nand\_init**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | exmc_nand_init  |
| <b>Function prototype</b>    | void exmc_nand_init(exmc_nand_parameter_struct*<br>exmc_nand_init_struct);                    |
| <b>Function descriptions</b> | initialize EXMC NAND bank   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>exmc_nand_init_struct</b> | structure for initialization, the structure members can refer to <a href="#">Table 3-347.</a> |
| <b>t</b>                     | <a href="#"><b>Structure exmc_nand_parameter_struct</b></a>                                   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* initialize EXMC NAND bank */

exmc_nand_parameter_struct nand_init_struct;
exmc_nand_pccard_timing_parameter_struct nand_timing_init_struct;
nand_timing_init_struct.setuptime = 1;
nand_timing_init_struct.waittime = 3;
nand_timing_init_struct.holdtime = 2;
nand_timing_init_struct.databus_hiztime = 2;
nand_init_struct.nand_bank = EXMC_BANK1_NAND;
nand_init_struct.ecc_size = EXMC_ECC_SIZE_2048BYTES;
nand_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_HCLK;
```

```

nand_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_HCLK;
nand_init_struct.ecc_logic = ENABLE;
nand_init_struct.databus_width = EXMC_NAND_DATABUS_WIDTH_8B;
nand_init_struct.wait_feature = ENABLE;
nand_init_struct.common_space_timing = &nand_timing_init_struct;
nand_init_struct.attribute_space_timing = &nand_timing_init_struct;
exmc_nand_init(&nand_init_struct);
  
```

### **exmc\_nand\_enable**

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

**Table 3-361. Function exmc\_nand\_enable**

|  |   |
|--|---|
| <b>Function name</b>                     | exmc_nand_enable                                |
| <b>Function prototype</b>                | void exmc_nand_enable(uint32_t exmc_nand_bank); |
| <b>Function descriptions</b>             | enable EXMC NAND bank                           |
| <b>Precondition</b>                      | -   |
| <b>The called functions</b>              | -   |
| <b>Input parameter{in}</b>               |   |
| <b>exmc_nand_bank</b>                    | specifies the NAND bank                         |
| <i>EXMC_BANKx_NAND(</i><br><i>x=1,2)</i> | bank of NAND                                    |
| <b>Output parameter{out}</b>             |   |
| -  | -   |
| <b>Return value</b>                      |   |
| -  | -   |

Example:

```

/* enable NAND bank */

exmc_nand_enable(EXMC_BANK1_NAND);
  
```

### **exmc\_nand\_disable**

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

**Table 3-362. Function exmc\_nand\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | exmc_nand_disable                                |
| <b>Function prototype</b>    | void exmc_nand_disable(uint32_t exmc_nand_bank); |
| <b>Function descriptions</b> | disable EXMC NAND bank                           |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |

| Input parameter{in}                      |                         |
|--|-------------------------|
| <b>exmc_nand_bank</b>                    | specifies the NAND bank |
| <i>EXMC_BANKx_NAND(</i><br><i>x=1,2)</i> | bank of NAND            |
| Output parameter{out}                    |                         |
| -  | -                       |
| Return value                             |                         |
| -  | -                       |

Example:

```
/* disable NAND bank */

exmc_nand_disable(EXMC_BANK1_NAND);
```

### **exmc\_nand\_ecc\_config**

The description of exmc\_nand\_ecc\_config is shown as below:

**Table 3-363. Function exmc\_nand\_ecc\_config**

| Function name                            | exmc_nand_ecc_config  |
|--|---|
| <b>Function prototype</b>                | void exmc_nand_ecc_config(uint32_t exmc_nand_bank, ControlStatus newvalue); |
| <b>Function descriptions</b>             | enable or disable the EXMC NAND ECC function                                |
| <b>Precondition</b>                      | -   |
| <b>The called functions</b>              | -   |
| Input parameter{in}                      |   |
| <b>exmc_nand_bank</b>                    | specifies the NAND bank   |
| <i>EXMC_BANKx_NAND(</i><br><i>x=1,2)</i> | bank of NAND  |
| Input parameter{in}                      |   |
| <b>newvalue</b>                          | control value   |
| <i>ENABLE</i>                            | enable function   |
| <i>DISABLE</i>                           | disable function  |
| Output parameter{out}                    |   |
| -  | -   |
| Return value                             |   |
| -  | -   |

Example:

```
/* enable the EXMC NAND ECC function */

exmc_nand_ecc_config(EXMC_BANK1_NAND, ENABLE);
```

### **exmc\_ecc\_get**

The description of exmc\_ecc\_get is shown as below:

**Table 3-364. Function exmc\_ecc\_get**

|  |   |
|--|---|
| <b>Function name</b>                     | exmc_ecc_get                                    |
| <b>Function prototype</b>                | uint32_t exmc_ecc_get(uint32_t exmc_nand_bank); |
| <b>Function descriptions</b>             | get the EXMC ECC value                          |
| <b>Precondition</b>                      | -   |
| <b>The called functions</b>              | -   |
| <b>Input parameter{in}</b>               |   |
| <b>exmc_nand_bank</b>                    | specifies the NAND bank                         |
| <b>EXMC_BANKx_NAND(</b><br><b>x=1,2)</b> | bank of NAND                                    |
| <b>Output parameter{out}</b>             |   |
| -  | -   |
| <b>Return value</b>                      |   |
| <b>uint32_t</b>                          | 0-0xFFFFFFFF                                    |

Example:

```
/* get the EXMC ECC value */

uint32_t value;

value = exmc_ecc_get(EXMC_BANK1_NAND);
```

### **exmc\_pccard\_deinit**

The description of exmc\_pccard\_deinit is shown as below:

**Table 3-365. Function exmc\_pccard\_deinit**

|                              |                                |
|------------------------------|--------------------------------|
| <b>Function name</b>         | exmc_pccard_deinit             |
| <b>Function prototype</b>    | void exmc_pccard_deinit(void); |
| <b>Function descriptions</b> | deinitialize EXMC PC card 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 PC card bank */
```

```
exmc_pccard_deinit();
```

### **exmc\_pccard\_struct\_para\_init**

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

**Table 3-366. Function exmc\_pccard\_struct\_para\_init**

|                                |   |
|--------------------------------|---|
| <b>Function name</b>           | exmc_pccard_struct_para_init  |
| <b>Function prototype</b>      | void exmc_pccard_struct_para_init(exmc_pccard_parameter_struct*<br>exmc_pccard_init_struct);  |
| <b>Function descriptions</b>   | initialize exmc_pccard_parameter_struct with the default values   |
| <b>Precondition</b>            | -   |
| <b>The called functions</b>    | -   |
| <b>Input parameter{in}</b>     |   |
| -                              | -   |
| <b>Output parameter{out}</b>   |   |
| <b>exmc_pccard_init_struct</b> | structure for initialization, the structure members can refer to <a href="#">Table 3-348.</a><br><a href="#">Structure exmc_pccard_parameter_struct</a> |
| <b>Return value</b>            |   |
| -                              | -   |

Example:

```
/* initialize the structure pccard_init_struct */
exmc_pccard_parameter_struct pccard_init_struct;
exmc_pccard_struct_para_init(&pccard_init_struct);
```

### **exmc\_pccard\_init**

The description of exmc\_pccard\_init is shown as below:

**Table 3-367. Function exmc\_pccard\_init**

|                                |   |
|--------------------------------|---|
| <b>Function name</b>           | exmc_pccard_init  |
| <b>Function prototype</b>      | void exmc_pccard_init(exmc_pccard_parameter_struct*<br>exmc_pccard_init_struct);  |
| <b>Function descriptions</b>   | initialize EXMC PC card bank  |
| <b>Precondition</b>            | -   |
| <b>The called functions</b>    | -   |
| <b>Input parameter{in}</b>     |   |
| <b>exmc_pccard_init_struct</b> | structure for initialization, the structure members can refer to <a href="#">Table 3-348.</a><br><a href="#">Structure exmc_pccard_parameter_struct</a> |
| <b>Output parameter{out}</b>   |   |
| -                              | -   |
| <b>Return value</b>            |   |
| -                              | -   |

Example:

```

/* initialize EXMC PC card bank */

exmc_pccard_parameter_struct pccard_init_struct;
exmc_nand_pccard_timing_parameter_struct pccard_timing_init_struct;
pccard_timing_init_struct.databus_hiztime = 2;
pccard_timing_init_struct.holdtime = 2;
pccard_timing_init_struct.waittime = 3;
pccard_timing_init_struct.setuptime = 1;
pccard_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_HCLK;
pccard_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_HCLK;
pccard_init_struct.wait_feature = DISABLE;
pccard_init_struct.common_space_timing = &pccard_timing_init_struct;
pccard_init_struct.attribute_space_timing = &pccard_timing_init_struct;
pccard_init_struct.io_space_timing = &pccard_timing_init_struct;
exmc_pccard_init(&pccard_init_struct);

```

### **exmc\_pccard\_enable**

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

**Table 3-368. Function exmc\_pccard\_enable**

|                              |                                |
|------------------------------|--------------------------------|
| <b>Function name</b>         | exmc_pccard_enable             |
| <b>Function prototype</b>    | void exmc_pccard_enable(void); |
| <b>Function descriptions</b> | enable PC card 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 PC card bank */

exmc_pccard_enable();

```

### **exmc\_pccard\_disable**

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

**Table 3-369. Function exmc\_pccard\_disable**

|                              |                                 |
|------------------------------|---------------------------------|
| <b>Function name</b>         | exmc_pccard_disable             |
| <b>Function prototype</b>    | void exmc_pccard_disable(void); |
| <b>Function descriptions</b> | disable PC card 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 PC card bank */

exmc_pccard_disable();
```

### **exmc\_sdram\_deinit**

The description of exmc\_sdram\_deinit is shown as below:

**Table 3-370. Function exmc\_sdram\_deinit**

|   |   |
|---|---|
| <b>Function name</b>                          | exmc_sdram_deinit                                   |
| <b>Function prototype</b>                     | void exmc_sdram_deinit(uint32_t exmc_sdram_device); |
| <b>Function descriptions</b>                  | deinitialize EXMC SDRAM device                      |
| <b>Precondition</b>                           | -   |
| <b>The called functions</b>                   | -   |
| <b>Input parameter{in}</b>                    |   |
| <b>exmc_sdram_device</b>                      | select the SDRAM device                             |
| <b>EXMC_SDRAM_DEVICE</b><br><i>Ex(x=0, 1)</i> | device x of SDRAM                                   |
| <b>Output parameter{out}</b>                  |   |
| -   | -   |
| <b>Return value</b>                           |   |
| -   | -   |

Example:

```
/* deinitialize EXMC SDRAM device */

exmc_sdram_deinit(EXMC_SDRAM_DEVICE0);
```

### **exmc\_sdrdram\_struct\_para\_init**

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

**Table 3-371. Function exmc\_sdrdram\_struct\_para\_init**

|                                 |  |
|---------------------------------|--|
| <b>Function name</b>            | exmc_sdrdram_struct_para_init  |
| <b>Function prototype</b>       | void exmc_sdrdram_struct_para_init(exmc_sdrdram_parameter_struct*<br>exmc_sdrdram_init_struct);  |
| <b>Function descriptions</b>    | initialize exmc_sdrdram_init_struct with the default values  |
| <b>Precondition</b>             | -  |
| <b>The called functions</b>     | -  |
| <b>Input parameter{in}</b>      |  |
| -                               | -  |
| <b>Output parameter{out}</b>    |  |
| <b>exmc_sdrdram_init_struct</b> | structure for initialization, the structure members can refer to <a href="#">Table 3-350.</a><br><a href="#">Structure exmc_sdrdram_parameter_struct</a> |
| <b>Return value</b>             |  |
| -                               | -  |

Example:

```
/* initialize the structure exmc_sdrdram_init_struct */

exmc_sdrdram_parameter_struct exmc_sdrdram_init_struct;
exmc_sdrdram_struct_para_init(&exmc_sdrdram_init_struct);
```

### **exmc\_sdrdram\_init**

The description of exmc\_sdrdram\_init is shown as below:

**Table 3-372. Function exmc\_sdrdram\_init**

|                                      |  |
|--------------------------------------|--|
| <b>Function name</b>                 | exmc_sdrdram_init  |
| <b>Function prototype</b>            | void exmc_sdrdram_init(exmc_sdrdram_parameter_struct*<br>exmc_sdrdram_init_struct);  |
| <b>Function descriptions</b>         | initialize EXMC SDRAM device   |
| <b>Precondition</b>                  | -  |
| <b>The called functions</b>          | -  |
| <b>Input parameter{in}</b>           |  |
| <b>exmc_sdrdram_parameter_struct</b> | structure for initialization, the structure members can refer to <a href="#">Table 3-350.</a><br><a href="#">Structure exmc_sdrdram_parameter_struct</a> |
| <b>Output parameter{out}</b>         |  |
| -                                    | -  |
| <b>Return value</b>                  |  |
| -                                    | -  |

Example:

```

exmc_sdrdram_parameter_struct      sdram_init_struct;

sdram_init_struct.sdrdram_device = sdrdram_device;

sdram_init_struct.column_address_width = EXMC_SDRAM_COW_ADDRESS_9;

sdram_init_struct.row_address_width = EXMC_SDRAM_ROW_ADDRESS_13;

sdram_init_struct.data_width = EXMC_SDRAM_DATABUS_WIDTH_16B;

sdram_init_struct.internal_bank_number = EXMC_SDRAM_4_INTER_BANK;

sdram_init_struct.cas_latency = EXMC_CAS_LATENCY_3_SDCLK;

sdram_init_struct.write_protection = DISABLE;

sdram_init_struct.sdclock_config = EXMC_SDCLK_PERIODS_2_HCLK;

sdram_init_struct.brust_read_switch = ENABLE;

sdram_init_struct.pipeline_read_delay = EXMC_PIPELINE_DELAY_1_HCLK;

sdram_init_struct.timing = &sdrdram_timing_init_struct;

exmc_sdrdram_init(&sdrdram_init_struct);
  
```

### **exmc\_sdrdram\_struct\_command\_para\_init**

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

**Table 3-373. Function exmc\_sdrdram\_struct\_command\_para\_init**

| Function name                    | exmc_sdrdram_struct_command_para_init  |  |
|----------------------------------|--|--|
| Function prototype               | void<br>exmc_sdrdram_struct_command_para_init(exmc_sdrdram_command_parameter_struct *exmc_sdrdram_command_init_struct);  |  |
| Function descriptions            | initialize exmc_sdrdram_command_parameter_struct with the default values   |  |
| Precondition                     | -  |  |
| The called functions             | -  |  |
| <b>Input parameter{in}</b>       |  |  |
| -                                | -  |  |
| <b>Output parameter{out}</b>     |  |  |
| exmc_sdrdram_command_init_struct | structure for initialization, the structure members can refer to <a href="#">Table 3-351.</a><br><a href="#">Structure exmc_sdrdram_command_parameter_struct</a> |  |
| <b>Return value</b>              |  |  |
| -                                | -  |  |

Example:

```

/* initialize the structure exmc_sdrdram_command_init_struct */

exmc_sdrdram_command_parameter_struct t exmc_sdrdram_command_init_struct;
  
```

---

```
exmc_sdrdram_struct_command_para_init (&exmc_sdrdram_command_init_struct);
```

### **exmc\_sdrdram\_command\_config**

The description of exmc\_sdrdram\_command\_config is shown as below:

**Table 3-374. Function exmc\_sdrdram\_command\_config**

|                                      |  |
|--------------------------------------|--|
| <b>Function name</b>                 | exmc_sdrdram_command_config  |
| <b>Function prototype</b>            | void<br>exmc_sdrdram_command_config(exmc_sdrdram_command_parameter_struct*<br>exmc_sdrdram_command_init_struct);   |
| <b>Function descriptions</b>         | configure the SDRAM memory command   |
| <b>Precondition</b>                  | -  |
| <b>The called functions</b>          | -  |
| <b>Input parameter{in}</b>           |  |
| exmc_sdrdram_comma<br>nd_init_struct | structure for initialization, the structure members can refer to <a href="#">Table 3-351.</a><br><a href="#">Structure exmc_sdrdram command parameter struct</a> |
| <b>Output parameter{out}</b>         |  |
| -                                    | -  |
| <b>Return value</b>                  |  |
| -                                    | -  |

Example:

```
/* configure the SDRAM memory command */

exmc_sdrdram_command_parameter_struct exmc_sdrdram_command_init_struct;
sdrdram_command_init_struct.command = EXMC_SDRAM_CLOCK_ENABLE;
sdrdram_command_init_struct.bank_select = bank_select;
sdrdram_command_init_struct.auto_refresh_number = EXMC_SDRAM_AUTO_REFRESH_1
_SDCLK;
sdrdram_command_init_struct.mode_register_content = 0U;
exmc_sdrdram_command_config(&exmc_sdrdram_command_init_struct);
```

### **exmc\_sdrdram\_refresh\_count\_set**

The description of exmc\_sdrdram\_refresh\_count\_set is shown as below:

**Table 3-375. Function exmc\_sdrdram\_refresh\_count\_set**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | exmc_sdrdram_refresh_count_set                            |
| <b>Function prototype</b>    | void exmc_sdrdram_refresh_count_set(uint32_t exmc_count); |
| <b>Function descriptions</b> | set auto-refresh interval                                 |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |

| Input parameter{in}   |  |
|-----------------------|--|
| <b>exmc_count</b>     | the number SDRAM clock cycles unit between two successive auto-refresh commands, 0x0000~0xFFFF |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

Example:

```
/* set auto-refresh interval */
exmc_sdram_refresh_count_set(0x01);
```

### **exmc\_sdram\_autorefresh\_number\_set**

The description of exmc\_sdram\_autorefresh\_number\_set is shown as below:

**Table 3-376. Function exmc\_sdram\_autorefresh\_number\_set**

| <b>Function name</b>                                 | exmc_sdram_autorefresh_number_set                             |
|--|---|
| <b>Function prototype</b>                            | void exmc_sdram_autorefresh_number_set(uint32_t exmc_number); |
| <b>Function descriptions</b>                         | set the number of successive auto-refresh command             |
| <b>Precondition</b>                                  | -   |
| <b>The called functions</b>                          | -   |
| Input parameter{in}                                  |   |
| <b>exmc_number</b>                                   | the number of successive auto-refresh cycles will be send     |
| <i>EXMC_SDRAM_AUTO_REFRESH_x_SDCLK(<br/>x=1..15)</i> | auto-refresh cycles   |
| Output parameter{out}                                |   |
| -  | -   |
| Return value   |   |
| -  | -   |

Example:

```
/* set the number of successive auto-refresh command */
exmc_sdram_autorefresh_number_set(EXMC_SDRAM_AUTO_REFRESH_2_SDCLK);
```

### **exmc\_sdram\_write\_protection\_config**

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

**Table 3-377. Function exmc\_sdram\_write\_protection\_config**

|                           |   |
|---------------------------|---|
| <b>Function name</b>      | exmc_sdram_write_protection_config                                  |
| <b>Function prototype</b> | void exmc_sdram_write_protection_config(uint32_t exmc_sdram_device, |

|                                  |   |
|----------------------------------|---|
|                                  | ControlStatus newvalue);                |
| <b>Function descriptions</b>     | configure the write protection function |
| <b>Precondition</b>              | -                                       |
| <b>The called functions</b>      | -                                       |
| <b>Input parameter{in}</b>       |   |
| <b>exmc_sdram_device</b>         | specifies the SDRAM device              |
| <i>EXMC_SDRAM_DEVICEx(x=0,1)</i> | SDRAM DEVICE                            |
| <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 the EXMC SDRAM write protection function */
exmc_sdram_write_protection_config(EXMC_SDRAM_DEVICE0, ENABLE);
```

### **exmc\_sdram\_bankstatus\_get**

The description of `exmc_sdram_bankstatus_get` is shown as below:

**Table 3-378. Function exmc\_sdram\_bankstatus\_get**

|  |   |
|--|---|
|  | exmc_sdram_bankstatus_get                                       |
| <b>Function prototype</b>              | uint32_t exmc_sdram_bankstatus_get(uint32_t exmc_sdram_device); |
| <b>Function descriptions</b>           | get the status of SDRAM device0 or device1                      |
| <b>Precondition</b>                    | -   |
| <b>The called functions</b>            | -   |
| <b>Input parameter{in}</b>             |   |
| <b>exmc_sdram_device</b>               | specifies the SDRAM device                                      |
| <i>EXMC_SDRAM_DEVICEx(x=0,1)</i>       | SDRAM DEVICE  |
| <b>Output parameter{out}</b>           |   |
| -                                      | -   |
| <b>Return value</b>                    |   |
| <b>uint32_t</b>                        | the status of SDRAM device                                      |
| <i>EXMC_SDRAM_DEVICEx_NORMAL</i>       | normal status   |
| <i>EXMC_SDRAM_DEVICEx_SELF_REFRESH</i> | self refresh status   |

|   |                   |
|---|-------------------|
| <code>EXMC_SDRAM_DEVICE_POWER_DOWN</code> | power down status |
|---|-------------------|

Example:

```
/* get the status of SDRAM device0 */
uint32_t temp;
temp = exmc_sdram_bankstatus_get(EXMC_SDRAM_DEVICE0);
```

### **exmc\_sdram\_readsampel\_config**

The description of `exmc_sdram_readsampel_config` is shown as below:

**Table 3-379. Function `exmc_sdram_readsampel_config`**

|   |   |
|---|---|
| <b>Function name</b>                                    | <code>exmc_sdram_readsampel_config</code>   |
| <b>Function prototype</b>                               | <code>void exmc_sdram_readsampel_config(uint32_t delay_cell, uint32_t extra_hclk);</code> |
| <b>Function descriptions</b>                            | configure the delayed sample clock of read data   |
| <b>Precondition</b>                                     | -   |
| <b>The called functions</b>                             | -   |
| <b>Input parameter{in}</b>                              |   |
| <code>delay_cell</code>                                 | SDRAM the delayed sample clock of read data   |
| <code>EXMC_SDRAM_X_DELAY_CELL(AY_CELL(x=0..15)</code>   | SDRAM delay cell  |
| <b>Input parameter{in}</b>                              |   |
| <code>extra_hclk</code>                                 | sample cycle of read data   |
| <code>EXMC_SDRAM_READSAMPLE_X_EXTRAHC_LK (x=0,1)</code> | sample cycle  |
| <b>Output parameter{out}</b>                            |   |
| -   | -   |
| <b>Return value</b>                                     |   |
| -   | -   |

Example:

```
/* configure the delayed sample clock of read data */
exmc_sdram_readsampel_config(EXMC_SDRAM_0_DELAY_CELL,
EXMC_SDRAM_READSAMPLE_0_EXTRAHCLK);
```

### **exmc\_sdram\_readsampel\_enable**

The description of `exmc_sdram_readsampel_enable` is shown as below:

**Table 3-380. Function exmc\_sdrdram\_readsampEnable**

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

Example:

```
/* enable read sample */

exmc_sdrdram_readsampEnable();
```

### **exmc\_sdrdram\_readsampDisable**

The description of exmc\_sdrdram\_readsampDisable is shown as below:

**Table 3-381. Function exmc\_sdrdram\_readsampDisable**

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

Example:

```
/* disable read sample */

exmc_sdrdram_readsampDisable();
```

### **exmc\_sqipSram\_deinit**

The description of exmc\_sqipSram\_deinit is shown as below:

**Table 3-382. Function exmc\_sqipSram\_deinit**

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

|                              |                                   |
|------------------------------|-----------------------------------|
| <b>Function prototype</b>    | void exmc_sqpipsram_deinit(void); |
| <b>Function descriptions</b> | deinitialize EXMC SQPIPSRAM       |
| <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 SQPIPSRAM */

exmc_sqpipsram_deinit();
```

### **exmc\_sqpipsram\_struct\_para\_init**

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

**Table 3-383. Function exmc\_sqpipsram\_struct\_para\_init**

|                                   |  |
|-----------------------------------|--|
| <b>Function name</b>              | exmc_sqpipsram_struct_para_init  |
| <b>Function prototype</b>         | void exmc_sqpipsram_struct_para_init(exmc_sqpipsram_parameter_struct*<br>exmc_sqpipsram_init_struct);  |
| <b>Function descriptions</b>      | initialize exmc_sqpipsram_init_struct with the default values  |
| <b>Precondition</b>               | -  |
| <b>The called functions</b>       | -  |
| <b>Input parameter{in}</b>        |  |
| -                                 | -  |
| <b>Output parameter{out}</b>      |  |
| <b>exmc_sqpipsram_init_struct</b> | structure for initialization, the structure members can refer to <a href="#">Table 3-352.</a><br><a href="#">Structure exmc_sqpipsram_parameter_struct</a> |
| <b>Return value</b>               |  |
| -                                 | -  |

Example:

```
/* initialize EXMC SQPIPSRAM */

exmc_sqpipsram_parameter_struct exmc_sqpipsram_init_struct;

exmc_sqpipsram_init(&exmc_sqpipsram_init_struct);
```

### **exmc\_sqpipsram\_init**

The description of exmc\_sqpipsram\_init is shown as below:

**Table 3-384. Function exmc\_sqipsram\_init**

|                                |   |
|--------------------------------|---|
| <b>Function name</b>           | exmc_sqipsram_init  |
| <b>Function prototype</b>      | void exmc_sqipsram_init(exmc_sqipsram_parameter_struct*<br>exmc_sqipsram_init_struct);  |
| <b>Function descriptions</b>   | initialize EXMC SQPIPSRAM   |
| <b>Precondition</b>            | -   |
| <b>The called functions</b>    | -   |
| <b>Input parameter{in}</b>     |   |
| exmc_sqipsram_parameter_struct | structure for initialization, the structure members can refer to <a href="#">Table 3-352.</a><br><a href="#">Structure exmc_sqipsram_parameter_struct</a> |
| <b>Output parameter{out}</b>   |   |
| -                              | -   |
| <b>Return value</b>            |   |
| -                              | -   |

Example:

```
/* initialize EXMC SQPIPSRAM */

exmc_sqipsram_parameter_struct exmc_sqipsram_init_struct;

exmc_sqipsram_init_struct->sample_polarity = EXMC_SQPIPSRAM_SAMPLE_RISING_EDGE;

exmc_sqipsram_init_struct->id_length = EXMC_SQPIPSRAM_ID_LENGTH_64B;

exmc_sqipsram_init_struct->address_bits = EXMC_SQPIPSRAM_ADDR_LENGTH_24B;

exmc_sqipsram_init_struct->command_bits = EXMC_SQPIPSRAM_COMMAND_LENGTH_8B; exmc_sqipsram_init(&exmc_sqipsram_init_struct);
```

### **exmc\_sqipsram\_read\_command\_set**

The description of exmc\_sqipsram\_read\_command\_set is shown as below:

**Table 3-385. Function exmc\_sqipsram\_read\_command\_set**

|                                  |   |
|----------------------------------|---|
| <b>Function name</b>             | exmc_sqipsram_read_command_set  |
| <b>Function prototype</b>        | void exmc_sqipsram_read_command_set(uint32_t read_command_mode,<br>uint32_t read_wait_cycle, uint32_t read_command_code); |
| <b>Function descriptions</b>     | set the read command  |
| <b>Precondition</b>              | -   |
| <b>The called functions</b>      | -   |
| <b>Input parameter{in}</b>       |   |
| read_command_mode                | configure SPI PSRAM read command mode   |
| EXMC_SQPIPSRAM_READ_MODE_DISABLE | not SPI mode  |
| EXMC_SQPIPSRAM_READ_MODE_ENABLE  | SPI mode  |

|                              |  |
|------------------------------|--|
| <i>EAD_MODE_SPI</i>          |  |
| <i>EXMC_SQPIPSRAM_R</i>      | SQPI mode                                    |
| <i>EAD_MODE_QPI</i>          |  |
| <b>Input parameter{in}</b>   |  |
| <i>read_wait_cycle</i>       | wait cycle number after address phase, 0..15 |
| <b>Input parameter{in}</b>   |  |
| <i>read_command_code</i>     | read command for AHB read transfer           |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* set the read command */

exmc_sqpipsram_read_command_set(EXMC_SQPIPSRAM_READ_MODE_SPI,      0x01,
0x01);
```

### **exmc\_sqpipsram\_write\_command\_set**

The description of exmc\_sqpipsram\_write\_command\_set is shown as below:

**Table 3-386. Function exmc\_sqpipsram\_write\_command\_set**

|  |   |
|--|---|
| <b>Function name</b>   | exmc_sqpipsram_write_command_set  |
| <b>Function prototype</b>                                      | void exmc_sqpipsram_write_command_set(uint32_t<br>write_command_mode, uint32_t write_wait_cycle, uint32_t<br>write_command_code); |
| <b>Function descriptions</b>                                   | set the write command   |
| <b>Precondition</b>  | -   |
| <b>The called functions</b>                                    | -   |
| <b>Input parameter{in}</b>                                     |   |
| <i>write_command_mod</i><br><i>e</i>                           | configure SPI PSRAM write command mode  |
| <i>EXMC_SQPIPSRAM_</i><br><i>WRITE_MODE_DISAB</i><br><i>LE</i> | not SPI mode  |
| <i>EXMC_SQPIPSRAM_</i><br><i>WRITE_MODE_SPI</i>                | SPI mode  |
| <i>EXMC_SQPIPSRAM_</i><br><i>WRITE_MODE_SQPI</i>               | SQPI mode   |
| <i>EXMC_SQPIPSRAM_</i><br><i>WRITE_MODE_QPI</i>                | QPI mode  |

|                              |  |
|------------------------------|--|
| <b>Input parameter{in}</b>   |  |
| <b>write_wait_cycle</b>      | wait cycle number after address phase, 0..15 |
| <b>Input parameter{in}</b>   |  |
| <b>write_command_code</b>    | write command for AHB write transfer         |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* set the write command */

exmc_sqipsram_write_command_set(EXMC_SQPIPSRAM_READ_MODE_SPI,      0x01,
0x01);
```

### **exmc\_sqipsram\_read\_id\_command\_send**

The description of exmc\_sqipsram\_read\_id\_command\_send is shown as below:

**Table 3-387. Function exmc\_sqipsram\_read\_id\_command\_send**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | exmc_sqipsram_read_id_command_send             |
| <b>Function prototype</b>    | void exmc_sqipsram_read_id_command_send(void); |
| <b>Function descriptions</b> | send SPI read ID command                       |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* send SPI read ID command */

exmc_sqipsram_read_id_command_send();
```

### **exmc\_sqipsram\_write\_cmd\_send**

The description of exmc\_sqipsram\_write\_cmd\_send is shown as below:

**Table 3-388. Function exmc\_sqipsram\_write\_cmd\_send**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | exmc_sqipsram_write_cmd_send  |
| <b>Function prototype</b>    | void exmc_sqipsram_write_cmd_send(void);                            |
| <b>Function descriptions</b> | send SPI special command which does not have address and data phase |
| <b>Precondition</b>          | -   |

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

Example:

```
/* send SPI special command */

exmc_sqpipsram_write_cmd_send();
```

### **exmc\_sqpipsram\_low\_id\_get**

The description of exmc\_sqpipsram\_low\_id\_get is shown as below:

**Table 3-389. Function exmc\_sqpipsram\_low\_id\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | exmc_sqpipsram_low_id_get                 |
| <b>Function prototype</b>    | uint32_t exmc_sqpipsram_low_id_get(void); |
| <b>Function descriptions</b> | get the EXMC SPI ID low data              |
| <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 ID low data                           |

Example:

```
/* get the EXMC SPI ID low data */

uint32_t temp;

temp = exmc_sqpipsram_low_id_get();
```

### **exmc\_sqpipsram\_high\_id\_get**

The description of exmc\_sqpipsram\_high\_id\_get is shown as below:

**Table 3-390. Function exmc\_sqpipsram\_low\_id\_get**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | exmc_sqpipsram_high_id_get                 |
| <b>Function prototype</b>    | uint32_t exmc_sqpipsram_high_id_get(void); |
| <b>Function descriptions</b> | get the EXMC SPI ID high data              |
| <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 ID high data |

Example:

```
/* get the EXMC SPI ID low data */

uint32_t temp;

temp = exmc_sqipssram_high_id_get();
```

### **exmc\_sqipssram\_send\_command\_state\_get**

The description of `exmc_sqipssram_send_command_state_get` is shown as below:

**Table 3-391. Function `exmc_sqipssram_send_command_state_get`**

|  |   |
|--|---|
| <b>Function name</b>                     | exmc_sqipssram_send_command_state_get   |
| <b>Function prototype</b>                | FlagStatus exmc_sqipssram_send_command_state_get(uint32_t send_command_flag); |
| <b>Function descriptions</b>             | get the bit value of EXMC send write command bit or read ID command           |
| <b>Precondition</b>                      | -   |
| <b>The called functions</b>              | -   |
| <b>Input parameter{in}</b>               |   |
| <code>send_command_flag</code>           | the send command flag   |
| <code>EXMC_SEND_COMMAN_DFLAG_RDID</code> | EXMC_SRCMD_RDID flag bit  |
| <code>EXMC_SEND_COMMAN_DFLAG_SC</code>   | EXMC_SWCMD_SC flag bit  |
| <b>Output parameter{out}</b>             |   |
| -  | -   |
| <b>Return value</b>                      |   |
| <b>FlagStatus</b>                        | SET or RESET  |

Example:

```
/* get the bit value of EXMC send write command bit */

FlagStatus send_command_flag_status;
send_command_flag_status = exmc_sqipssram_send_command_state_get(EXMC_SEND_COMMAND_FLAG_SC);
```

### exmc\_flag\_get

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

**Table 3-392. Function exmc\_flag\_get**

|                                     |   |
|-------------------------------------|---|
| <b>Function name</b>                | exmc_flag_get   |
| <b>Function prototype</b>           | FlagStatus exmc_flag_get(uint32_t bank, uint32_t flag); |
| <b>Function descriptions</b>        | get EXMC flag status                                    |
| <b>Precondition</b>                 | -   |
| <b>The called functions</b>         | -   |
| <b>Input parameter{in}</b>          |   |
| <b>bank</b>                         | specifies the NAND bank , PC card bank                  |
| <i>EXMC_BANK1_NAND</i>              | the NAND bank1  |
| <i>EXMC_BANK2_NAND</i>              | the NAND bank2  |
| <i>EXMC_BANK3_PCCA RD</i>           | the PC Card bank  |
| <i>EXMC_SDRAM_DEVIC E0</i>          | the SDRAM device0                                       |
| <i>EXMC_SDRAM_DEVIC E1</i>          | the SDRAM device1                                       |
| <b>Input parameter{in}</b>          |   |
| <b>flag</b>                         | flag  |
| <i>EXMC_NAND_PCCAR D_FLAG_RISE</i>  | interrupt rising edge status                            |
| <i>EXMC_NAND_PCCAR D_FLAG_LEVEL</i> | interrupt high-level status                             |
| <i>EXMC_NAND_PCCAR D_FLAG_FALL</i>  | interrupt falling edge status                           |
| <i>EXMC_NAND_PCCAR D_FLAG_FIFOE</i> | FIFO empty flag   |
| <i>EXMC_SDRAM_FLAG _REFRESH</i>     | refresh error interrupt flag                            |
| <i>EXMC_SDRAM_FLAG _NREADY</i>      | not ready status  |
| <b>Output parameter{out}</b>        |   |
| -                                   | -   |
| <b>Return value</b>                 |   |
| <b>FlagStatus</b>                   | SET or RESET  |

Example:

```
/* check EXMC flag is set or not */
```

```
FlagStatus status;
```

```
status = exmc_flag_get(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_FLAG_RISE);
```

### **exmc\_flag\_clear**

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

**Table 3-393. Function exmc\_flag\_clear**

|                                     |   |
|-------------------------------------|---|
| <b>Function name</b>                | exmc_flag_clear                                     |
| <b>Function prototype</b>           | void exmc_flag_clear(uint32_t bank, uint32_t flag); |
| <b>Function descriptions</b>        | clear EXMC flag                                     |
| <b>Precondition</b>                 | -   |
| <b>The called functions</b>         | -   |
| <b>Input parameter{in}</b>          |   |
| <b>bank</b>                         | specifies the NAND bank, PC card bank               |
| <i>EXMC_BANK1_NAND</i>              | the NAND bank1                                      |
| <i>EXMC_BANK2_NAND</i>              | the NAND bank2                                      |
| <i>EXMC_BANK3_PCCA RD</i>           | the PC Card bank                                    |
| <i>EXMC_SDRAM_DEVIC E0</i>          | the SDRAM device0                                   |
| <i>EXMC_SDRAM_DEVIC E1</i>          | the SDRAM device1                                   |
| <b>Input parameter{in}</b>          |   |
| <b>flag</b>                         | flag  |
| <i>EXMC_NAND_PCCAR D_FLAG_RISE</i>  | interrupt rising edge status                        |
| <i>EXMC_NAND_PCCAR D_FLAG_LEVEL</i> | interrupt high-level status                         |
| <i>EXMC_NAND_PCCAR D_FLAG_FALL</i>  | interrupt falling edge status                       |
| <i>EXMC_NAND_PCCAR D_FLAG_FIFOE</i> | FIFO empty flag                                     |
| <i>EXMC_SDRAM_FLAG _REFRESH</i>     | refresh error interrupt flag                        |
| <i>EXMC_SDRAM_FLAG _NREADY</i>      | not ready status                                    |
| <b>Output parameter{out}</b>        |   |
| -                                   | -   |
| <b>Return value</b>                 |   |
| -                                   | -   |

Example:

```
/* clear EXMC flag */
exmc_flag_clear(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_FLAG_RISE);
```

### **exmc\_interrupt\_enable**

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

**Table 3-394. Function exmc\_interrupt\_enable**

|                                    |   |
|------------------------------------|---|
| <b>Function name</b>               | exmc_interrupt_enable   |
| <b>Function prototype</b>          | void exmc_interrupt_enable(uint32_t bank, uint32_t interrupt_source); |
| <b>Function descriptions</b>       | enable EXMC interrupt   |
| <b>Precondition</b>                | -   |
| <b>The called functions</b>        | -   |
| <b>Input parameter{in}</b>         |   |
| <b>bank</b>                        | specifies the NAND bank,PC card bank                                  |
| <i>EXMC_BANK1_NAND</i>             | the NAND bank1  |
| <i>EXMC_BANK2_NAND</i>             | the NAND bank2  |
| <i>EXMC_BANK3_PCCA RD</i>          | the PC card bank  |
| <i>EXMC_SDRAM_DEVIC E0</i>         | the SDRAM device0   |
| <i>EXMC_SDRAM_DEVIC E1</i>         | the SDRAM device1   |
| <b>Input parameter{in}</b>         |   |
| <b>interrupt_source</b>            | specifies get which interrupt flag                                    |
| <i>EXMC_NAND_PCCAR D_INT_RISE</i>  | interrupt source of rising edge                                       |
| <i>EXMC_NAND_PCCAR D_INT_LEVEL</i> | interrupt source of high-level  |
| <i>EXMC_NAND_PCCAR D_INT_FALL</i>  | interrupt source of falling edge                                      |
| <i>EXMC_SDRAM_INT_REFRESH</i>      | interrupt source of refresh error                                     |
| <b>Output parameter{out}</b>       |   |
| -                                  | -   |
| <b>Return value</b>                |   |
| -                                  | -   |

Example:

```
/* enable EXMC interrupt */

exmc_interrupt_enable(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_INT_RISE);
```

### **exmc\_interrupt\_disable**

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

**Table 3-395. Function exmc\_interrupt\_disable**

|                                    |  |
|------------------------------------|--|
| <b>Function name</b>               | exmc_interrupt_disable   |
| <b>Function prototype</b>          | void exmc_interrupt_disable(uint32_t bank, uint32_t interrupt_source); |
| <b>Function descriptions</b>       | disable EXMC interrupt   |
| <b>Precondition</b>                | -  |
| <b>The called functions</b>        | -  |
| <b>Input parameter{in}</b>         |  |
| <b>bank</b>                        | specifies the NAND bank,PC card bank                                   |
| <i>EXMC_BANK1_NAND</i>             | the NAND bank1   |
| <i>EXMC_BANK2_NAND</i>             | the NAND bank2   |
| <i>EXMC_BANK3_PCCA RD</i>          | the PC card bank   |
| <i>EXMC_SDRAM_DEVIC E0</i>         | the SDRAM device0  |
| <i>EXMC_SDRAM_DEVIC E1</i>         | the SDRAM device1  |
| <b>Input parameter{in}</b>         |  |
| <b>interrupt_source</b>            | specifies get which interrupt flag                                     |
| <i>EXMC_NAND_PCCAR D_INT_RISE</i>  | interrupt source of rising edge  |
| <i>EXMC_NAND_PCCAR D_INT_LEVEL</i> | interrupt source of high-level   |
| <i>EXMC_NAND_PCCAR D_INT_FALL</i>  | interrupt source of falling edge                                       |
| <i>EXMC_SDRAM_INT_REFRESH</i>      | interrupt source of refresh error                                      |
| <b>Output parameter{out}</b>       |  |
| -                                  | -  |
| <b>Return value</b>                |  |
| -                                  | -  |

Example:

```
/* disable EXMC interrupt */

exmc_interrupt_disable(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_INT_RISE);

exmc_interrupt_flag_get
```

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

**Table 3-396. Function exmc\_interrupt\_flag\_get**

|                           |   |
|---------------------------|---|
| <b>Function name</b>      | exmc_interrupt_flag_get   |
| <b>Function prototype</b> | FlagStatus exmc_interrupt_flag_get(uint32_t bank, uint32_t interrupt_source); |

|                                     |  |
|-------------------------------------|--|
| <b>Function descriptions</b>        | get EXMC interrupt flag                |
| <b>Precondition</b>                 | -                                      |
| <b>The called functions</b>         | -                                      |
| <b>Input parameter{in}</b>          |  |
| <b>bank</b>                         | specifies the NAND bank , PC card bank |
| <i>EXMC_BANK1_NAND</i>              | the NAND bank1                         |
| <i>EXMC_BANK2_NAND</i>              | the NAND bank2                         |
| <i>EXMC_BANK3_PCCA RD</i>           | the PC Card bank                       |
| <i>EXMC_SDRAM_DEVIC E0</i>          | the SDRAM device0                      |
| <i>EXMC_SDRAM_DEVIC E1</i>          | the SDRAM device1                      |
| <b>Input parameter{in}</b>          |  |
| <b>interrupt_source</b>             | Interrupt flag                         |
| <i>EXMC_NAND_PCCAR D_INT_RISE</i>   | interrupt source of rising edge        |
| <i>EXMC_NAND_PCCAR D_INT_LEVEL</i>  | interrupt source of high-level         |
| <i>EXMC_NAND_PCCAR D_INT_FALL</i>   | interrupt source of falling edge       |
| <i>EXMC_SDRAM_INT_F LAG_REFRESH</i> | interrupt source of refresh error      |
| <b>Output parameter{out}</b>        |  |
| -                                   | -                                      |
| <b>Return value</b>                 |  |
| <b>FlagStatus</b>                   | SET or RESET                           |

Example:

```
/* check EXMC interrupt flag is set or not */

FlagStatus status;

status = exmc_interrupt_flag_get(EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_INT_RISE);
```

### **exmc\_interrupt\_flag\_clear**

The description of `exmc_interrupt_flag_clear` is shown as below:

**Table 3-397. Function `exmc_interrupt_flag_clear`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | <code>exmc_interrupt_flag_clear</code>   |
| <b>Function prototype</b>    | <code>void exmc_interrupt_flag_clear(uint32_t bank, uint32_t interrupt_source);</code> |
| <b>Function descriptions</b> | clear EXMC flag status   |
| <b>Precondition</b>          | -  |

| The called functions                    | -                                      |
|---|--|
| <b>Input parameter{in}</b>              |  |
| <b>bank</b>                             | specifies the NAND bank , PC card bank |
| <i>EXMC_BANK1_NAND</i>                  | the NAND bank1                         |
| <i>EXMC_BANK2_NAND</i>                  | the NAND bank2                         |
| <i>EXMC_BANK3_PCCA<br/>RD</i>           | the PC Card bank                       |
| <i>EXMC_SDRAM_DEVIC<br/>E0</i>          | the SDRAM device0                      |
| <i>EXMC_SDRAM_DEVIC<br/>E1</i>          | the SDRAM device1                      |
| <b>Input parameter{in}</b>              |  |
| <b>interrupt_source</b>                 | Interrupt flag                         |
| <i>EXMC_NAND_PCCAR<br/>D_INT_RISE</i>   | interrupt source of rising edge        |
| <i>EXMC_NAND_PCCAR<br/>D_INT_LEVEL</i>  | interrupt source of high-level         |
| <i>EXMC_NAND_PCCAR<br/>D_INT_FALL</i>   | interrupt source of falling edge       |
| <i>EXMC_SDRAM_INT_F<br/>LAG_REFRESH</i> | interrupt source of refresh error      |
| <b>Output parameter{out}</b>            |  |
| -                                       | -                                      |
| <b>Return value</b>                     |  |
| -                                       | -                                      |

Example:

```
/* clear EXMC interrupt flag */
exmc_interrupt_flag_clear(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_INT_RISE);
```

## 3.13. EXTI

EXTI is the interrupt/event controller in the MCU. It contains up to 20 independent edge detectors and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter [3.13.1](#), the EXTI firmware functions are introduced in chapter [3.13.2](#).

### 3.13.1. Descriptions of Peripheral registers

EXTI registers are listed in the table shown as below:

**Table 3-398. EXTI Registers**

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

### 3.13.2. Descriptions of Peripheral functions

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

**Table 3-399. EXTI firmware function**

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

#### Enum exti\_line\_enum

**Table 3-400. 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         |

| Member name | Function description |
|-------------|----------------------|
| EXTI_11     | EXTI line 11         |
| EXTI_12     | EXTI line 12         |
| EXTI_13     | EXTI line 13         |
| EXTI_14     | EXTI line 14         |
| EXTI_15     | EXTI line 15         |
| EXTI_16     | EXTI line 16         |
| EXTI_17     | EXTI line 17         |
| EXTI_18     | EXTI line 18         |
| EXTI_19     | EXTI line 19         |

### Enum exti\_mode\_enum

Table 3-401. Enum exti\_mode\_enum

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

### Enum exti\_trig\_type\_enum

Table 3-402. 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 3-403. Function exti\_deinit

|                              |   |
|------------------------------|---|
| Function name                | exti_deinit   |
| Function prototype           | void exti_deinit(void);                                   |
| Function descriptions        | reset the value of all EXTI registers into initial values |
| Precondition                 | -   |
| The called functions         | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

---

```
/* deinitialize the EXTI */
```

```
exti_deinit();
```

### **exti\_init**

The description of exti\_init is shown as below:

**Table 3-404. Function exti\_init**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | exti_init  |
| <b>Function prototype</b>    | void exti_init(exti_line_enum linex, exti_mode_enum mode,<br>exti_trig_type_enum trig_type); |
| <b>Function descriptions</b> | initialize EXTI line x   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| linex                        | EXTI line x, refer to <a href="#">Table 3-400. Enum exti_line_enum</a>                       |
| EXTI_x                       | x=0,1,2..19  |
| <b>Input parameter{in}</b>   |  |
| mode                         | EXTI mode, refer to <a href="#">Table 3-401. Enum exti_mode_enum</a>                         |
| EXTI_INTERRUPT               | interrupt mode   |
| EXTI_EVENT                   | event mode   |
| <b>Input parameter{in}</b>   |  |
| trig_type                    | interrupt trigger type, refer to <a href="#">Table 3-402. Enum exti_trig_type_enum</a>       |
| EXTI_TRIG_RISING             | rising edge trigger  |
| EXTI_TRIG_FALLING            | falling edge trigger   |
| EXTI_TRIG_BOTH               | rising edge and falling edge trigger   |
| EXTI_TRIG_NONE               | without rising edge or falling edge trigger  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

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

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

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

**Table 3-405. 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 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 3-400. Enum exti_line_enum</a> |
| <b>EXTI_x</b>                | x=0,1,2..19  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>FlagStatus</b>            | SET or RESET   |

Example:

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

FlagStatus state = 0;

state = exti_flag_get(EXTI_0);
```

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

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

**Table 3-406. 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 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 3-400. Enum exti_line_enum</a> |
| <b>EXTI_x</b>                | x=0,1,2..19  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

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

exti_flag_clear(EXTI_0);
```

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

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

**Table 3-407. 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 EXTI line x interrupt   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| linex                        | EXTI line x, refer to <a href="#">Table 3-400. Enum exti_line_enum</a> |
| EXTI_x                       | x=0,1,2..19  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

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

exti_interrupt_enable(EXTI_0);
```

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

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

**Table 3-408. Function exti\_interrupt\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | exti_interrupt_disable   |
| <b>Function prototype</b>    | void exti_interrupt_disable(exti_line_enum linex);                     |
| <b>Function descriptions</b> | disable EXTI line x interrupt  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| linex                        | EXTI line x, refer to <a href="#">Table 3-400. Enum exti_line_enum</a> |
| EXTI_x                       | x=0,1,2..19  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

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

exti_interrupt_disable(EXTI_0);
```

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

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

**Table 3-409. Function exti\_event\_enable**

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

|                              |  |
|------------------------------|--|
| <b>Function prototype</b>    | void exti_event_enable(exti_line_enum linex);                          |
| <b>Function descriptions</b> | enable EXTI line x event   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| linex                        | EXTI line x, refer to <a href="#">Table 3-400. Enum exti_line_enum</a> |
| EXTI_x                       | x=0,1,2..19  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

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

exti_event_enable(EXTI_0);
```

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

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

**Table 3-410. Function exti\_event\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | exti_event_disable   |
| <b>Function prototype</b>    | void exti_event_disable(exti_line_enum linex);                         |
| <b>Function descriptions</b> | disable EXTI line x event  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| linex                        | EXTI line x, refer to <a href="#">Table 3-400. Enum exti_line_enum</a> |
| EXTI_x                       | x=0,1,2..19  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable the event 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 3-411. Function exti\_software\_interrupt\_enable**

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

|                              |  |
|------------------------------|--|
| <b>Function prototype</b>    | void exti_software_interrupt_enable(exti_line_enum linex);             |
| <b>Function descriptions</b> | enable EXTI line x software interrupt                                  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| linex                        | EXTI line x, refer to <a href="#">Table 3-400. Enum exti_line_enum</a> |
| EXTI_x                       | x=0,1,2..19  |
| <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 3-412. Function exti\_software\_interrupt\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | exti_software_interrupt_disable  |
| <b>Function prototype</b>    | void exti_software_interrupt_disable(exti_line_enum linex);            |
| <b>Function descriptions</b> | disable EXTI line x software interrupt                                 |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| linex                        | EXTI line x, refer to <a href="#">Table 3-400. Enum exti_line_enum</a> |
| EXTI_x                       | x=0,1,2..19  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable EXTI line 0 software interrupt */
exti_software_interrupt_disable(EXTI_0);
```

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

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

**Table 3-413. Function exti\_interrupt\_flag\_get**

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

|                              |  |
|------------------------------|--|
| <b>Function prototype</b>    | FlagStatus exti_interrupt_flag_get(exti_line_enum linex);              |
| <b>Function descriptions</b> | get EXTI line x interrupt flag   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| linex                        | EXTI line x, refer to <a href="#">Table 3-400. Enum exti_line_enum</a> |
| EXTI_x                       | x=0,1,2..19  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| FlagStatus                   | SET or RESET   |

Example:

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

FlagStatus state = 0;

state = exti_interrupt_flag_get(EXTI_0);
```

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

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

**Table 3-414. Function exti\_interrupt\_flag\_clear**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | exti_interrupt_flag_clear  |
| <b>Function prototype</b>    | void exti_interrupt_flag_clear(exti_line_enum linex);                  |
| <b>Function descriptions</b> | clear EXTI line x interrupt flag                                       |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| linex                        | EXTI line x, refer to <a href="#">Table 3-400. Enum exti_line_enum</a> |
| EXTI_x                       | x=0,1,2..19  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

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

exti_interrupt_flag_clear(EXTI_0);
```

## **3.14. FMC**

There is flash controller and option byte for GD32F20x series. The FMC registers are listed

in chapter [3.14.1](#) the FMC firmware functions are introduced in chapter [3.14.2](#).

### 3.14.1. Descriptions of Peripheral registers

FMC registers are listed in the table shown as below:

**Table 3-415. FMC Registers**

| Registers  | Descriptions                      |
|------------|-----------------------------------|
| FMC_WS     | wait state register               |
| FMC_KEY0   | unlock key register 0             |
| FMC_OBKEY  | option byte unlock key register   |
| FMC_STAT0  | status register 0                 |
| FMC_CTL0   | control register 0                |
| FMC_ADDR0  | address register 0                |
| FMC_OBSTAT | option byte status register       |
| FMC_WP     | erase/program protection register |
| FMC_KEY1   | unlock key register 1             |
| FMC_STAT1  | status register 1                 |
| FMC_CTL1   | control register 1                |
| FMC_ADDR1  | address register 1                |
| FMC_WSEN   | wait state enable register        |
| FMC_PID    | product ID register               |

### 3.14.2. Descriptions of Peripheral functions

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

**Table 3-416. FMC firmware function**

| Function name        | Function description                             |
|----------------------|--|
| fmc_wscnt_set        | set the FMC wait state counter                   |
| fmc_unlock           | unlock the main FMC operation                    |
| fmc_bank0_unlock     | unlock the FMC bank0 operation                   |
| fmc_bank1_unlock     | unlock the FMC bank1 operation                   |
| fmc_lock             | lock the main FMC operation                      |
| fmc_bank0_lock       | lock the FMC bank0 operation                     |
| fmc_bank1_lock       | lock the FMC bank1 operation                     |
| fmc_page_erase       | erase page                                       |
| fmc_mass_erase       | erase whole chip                                 |
| fmc_bank0_erase      | erase whole bank0                                |
| fmc_bank1_erase      | erase whole bank1                                |
| fmc_word_program     | program a word at the corresponding address      |
| fmc_halfword_program | program a half word at the corresponding address |
| ob_unlock            | unlock the option byte operation                 |
| ob_lock              | lock the option byte operation                   |

| Function name                 | Function description                           |
|-------------------------------|--|
| 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                 | program option byte user                       |
| ob_data_program               | program option bytes data                      |
| ob_user_get                   | get the FMC option bytes user                  |
| ob_data_get                   | get the FMC option bytes data                  |
| ob_write_protection_get       | get the FMC option byte write protection       |
| ob_spc_get                    | get option byte security protection code value |
| fmc_flag_get                  | check flag is set or not                       |
| fmc_flag_clear                | clear the FMC flag                             |
| fmc_interrupt_enable          | enable FMC interrupt                           |
| fmc_interrupt_disable         | disable FMC interrupt                          |
| fmc_interrupt_flag_get        | get FMC interrupt flag state                   |
| fmc_interrupt_flag_clear      | clear FMC interrupt flag                       |

### Enum fmc\_state\_enum

Table 3-417. fmc\_state\_enum

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

### Enum fmc\_interrupt\_enum

Table 3-418. fmc\_interrupt\_enum

| Member name       | Function description               |
|-------------------|------------------------------------|
| FMC_INT_BANK0-END | FMC bank0 end of program interrupt |
| FMC_INT_BANK0-ERR | FMC bank0 error interrupt          |
| FMC_INT_BANK1-END | FMC bank1 end of program interrupt |
| FMC_INT_BANK1-ERR | FMC bank1 error interrupt          |

**Enum fmc\_flag\_enum**
**Table 3-419. fmc\_flag\_enum**

| Member name          | Function description                          |
|----------------------|---|
| FMC_FLAG_BANK0_BUSY  | FMC bank0 busy flag                           |
| FMC_FLAG_BANK0_PGERR | FMC bank0 operation error flag                |
| FMC_FLAG_BANK0_WPERR | FMC bank0 erase/program protection error flag |
| FMC_FLAG_BANK0_END   | FMC bank0 end of operation flag               |
| FMC_FLAG_OBER_R      | FMC option bytes read error flag              |
| FMC_FLAG_BANK1_BUSY  | FMC bank1 busy flag                           |
| FMC_FLAG_BANK1_PGERR | FMC bank1 operation error flag                |
| FMC_FLAG_BANK1_WPERR | FMC bank1 erase/program protection error flag |
| FMC_FLAG_BANK1_END   | FMC bank1 end of operation flag               |

**Enum fmc\_interrupt\_flag\_enum**
**Table 3-420. fmc\_interrupt\_flag\_enum**

| Member name              | Function description                                    |
|--------------------------|---|
| FMC_INT_FLAG_BANK0_PGERR | FMC bank0 operation error interrupt flag                |
| FMC_INT_FLAG_BANK0_WPERR | FMC bank0 erase/program protection error interrupt flag |
| FMC_INT_FLAG_BANK0_END   | FMC bank0 end of operation interrupt flag               |
| FMC_INT_FLAG_BANK1_PGERR | FMC bank1 operation error interrupt flag                |
| FMC_INT_FLAG_BANK1_WPERR | FMC bank1 erase/program protection error interrupt flag |
| FMC_INT_FLAG_BANK1_END   | FMC bank1 end of operation interrupt flag               |

**fmc\_wscnt\_set**

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

**Table 3-421. Function fmc\_wscnt\_set**

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

Example:

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

fmc_wscnt_set(WS_WSCNT_1);
```

### **fmc\_unlock**

The description of fmc\_unlock is shown as below:

**Table 3-422. Function fmc\_unlock**

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

Example:

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

fmc_unlock();
```

### **fmc\_bank0\_unlock**

The description of fmc\_bank0\_unlock is shown as below:

**Table 3-423. Function fmc\_bank0\_unlock**

|                              |                                     |
|------------------------------|-------------------------------------|
| <b>Function name</b>         | fmc_bank0_unlock                    |
| <b>Function prototype</b>    | void fmc_bank0_unlock(void);        |
| <b>Function descriptions</b> | unlock the main FMC bank0 operation |
| <b>Precondition</b>          | -                                   |
| <b>The called functions</b>  | -                                   |
| <b>Input parameter{in}</b>   |                                     |
| -                            | -                                   |
| <b>Output parameter{out}</b> |                                     |
| -                            | -                                   |
| <b>Return value</b>          |                                     |
| -                            | -                                   |

Example:

```
/* unlock the main FMC bank0 operation */

fmc_bank0_unlock();
```

### **fmc\_bank1\_unlock**

The description of fmc\_bank1\_unlock is shown as below:

**Table 3-424. Function fmc\_bank1\_unlock**

|                              |                                     |
|------------------------------|-------------------------------------|
| <b>Function name</b>         | fmc_bank1_unlock                    |
| <b>Function prototype</b>    | void fmc_bank1_unlock(void);        |
| <b>Function descriptions</b> | unlock the main FMC bank1 operation |
| <b>Precondition</b>          | -                                   |
| <b>The called functions</b>  | -                                   |
| <b>Input parameter{in}</b>   |                                     |
| -                            | -                                   |
| <b>Output parameter{out}</b> |                                     |
| -                            | -                                   |
| <b>Return value</b>          |                                     |
| -                            | -                                   |

Example:

```
/* unlock the main FMC bank1 operation */

fmc_bank1_unlock();
```

### **fmc\_lock**

The description of fmc\_lock is shown as below:

**Table 3-425. Function fmc\_lock**

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

Example:

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

fmc_lock();
```

### **fmc\_bank0\_lock**

The description of fmc\_bank0\_lock is shown as below:

**Table 3-426. Function fmc\_bank0\_lock**

|                              |                                   |
|------------------------------|-----------------------------------|
| <b>Function name</b>         | fmc_bank0_lock                    |
| <b>Function prototype</b>    | void fmc_bank0_lock(void);        |
| <b>Function descriptions</b> | lock the main FMC bank0 operation |
| <b>Precondition</b>          | -                                 |
| <b>The called functions</b>  | -                                 |
| <b>Input parameter{in}</b>   |                                   |
| -                            | -                                 |
| <b>Output parameter{out}</b> |                                   |
| -                            | -                                 |
| <b>Return value</b>          |                                   |
| -                            | -                                 |

Example:

```
/* lock the main FMC bank0 operation */

fmc_bank0_lock();
```

### fmc\_bank1\_lock

The description of fmc\_bank1\_lock is shown as below:

**Table 3-427. Function fmc\_bank1\_lock**

|                              |                                   |
|------------------------------|-----------------------------------|
| <b>Function name</b>         | fmc_bank1_lock                    |
| <b>Function prototype</b>    | void fmc_bank1_lock(void);        |
| <b>Function descriptions</b> | lock the main FMC bank1 operation |
| <b>Precondition</b>          | -                                 |
| <b>The called functions</b>  | -                                 |
| <b>Input parameter{in}</b>   |                                   |
| -                            | -                                 |
| <b>Output parameter{out}</b> |                                   |
| -                            | -                                 |
| <b>Return value</b>          |                                   |
| -                            | -                                 |

Example:

```
/* lock the main FMC bank1 operation */

fmc_bank1_lock();
```

### fmc\_page\_erase

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

**Table 3-428. Function fmc\_page\_erase**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | fmc_page_erase   |
| <b>Function prototype</b>    | fmc_state_enum fmc_page_erase(uint32_t page_address);                |
| <b>Function descriptions</b> | erase page   |
| <b>Precondition</b>          | fmc_unlock   |
| <b>The called functions</b>  | fmc_bank0_ready_wait / fmc_bank1_ready_wait                          |
| <b>Input parameter{in}</b>   |  |
| page_address                 | the page address to be erased  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>fmc_state_enum</b>        | state of FMC, refer to <a href="#">Table 3-417. fmc_state_enum</a> . |

Example:

```
/* erase page */

fmc_state_enum state;
state = fmc_page_erase(0x08004000);
```

### fmc\_mass\_erase

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

**Table 3-429. 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_bank0_ready_wait / fmc_bank1_ready_wait                          |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>fmc_state_enum</b>        | state of FMC, refer to <a href="#">Table 3-417. fmc_state_enum</a> . |

Example:

```
/* erase whole chip */

fmc_state_enum state;

state = fmc_mass_erase();
```

### fmc\_bank0\_erase

The description of fmc\_bank0\_erase is shown as below:

**Table 3-430. Function fmc\_bank0\_erase**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | fmc_bank0_erase  |
| <b>Function prototype</b>    | fmc_state_enum fmc_bank0_erase(void);                                |
| <b>Function descriptions</b> | erase whole bank0  |
| <b>Precondition</b>          | fmc_unlock   |
| <b>The called functions</b>  | fmc_bank0_ready_wait   |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>fmc_state_enum</b>        | state of FMC, refer to <a href="#">Table 3-417. fmc_state_enum</a> . |

Example:

```
/* erase bank0 whole chip */

fmc_state_enum state;

state = fmc_bank0_erase();
```

### fmc\_bank1\_erase

The description of fmc\_bank1\_erase is shown as below:

**Table 3-431. Function fmc\_bank1\_erase**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | fmc_bank1_erase  |
| <b>Function prototype</b>    | fmc_state_enum fmc_bank1_erase();                                    |
| <b>Function descriptions</b> | erase whole bank1  |
| <b>Precondition</b>          | fmc_unlock   |
| <b>The called functions</b>  | fmc_bank1_ready_wait   |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>fmc_state_enum</b>        | state of FMC, refer to <a href="#">Table 3-417. fmc_state_enum</a> . |

Example:

```
/* erase bank1 whole chip */
fmc_state_enum state;
state = fmc_bank1_erase();
```

### fmc\_word\_program

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

**Table 3-432. 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_bank0_ready_wait / fmc_bank1_ready_wait                          |
| <b>Input parameter{in}</b>   |  |
| address                      | the address to program   |
| <b>Input parameter{in}</b>   |  |
| data                         | the data to program  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>fmc_state_enum</b>        | state of FMC, refer to <a href="#">Table 3-417. fmc_state_enum</a> . |

Example:

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

```
fmc_state_enum state;
state = fmc_word_program ( 0x08004000,0xaabbccdd);
```

### **fmc\_halfword\_program**

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

**Table 3-433. Function fmc\_halfword\_program**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | fmc_halfword_program  |
| <b>Function prototype</b>    | fmc_state_enum fmc_halfword_program(uint32_t address, uint16_t data); |
| <b>Function descriptions</b> | program a halfword at the corresponding address                       |
| <b>Precondition</b>          | fmc_unlock  |
| <b>The called functions</b>  | fmc_bank0_ready_wait / fmc_bank1_ready_wait                           |
| <b>Input parameter{in}</b>   |   |
| address                      | the address to program  |
| <b>Input parameter{in}</b>   |   |
| data                         | the data to program   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>fmc_state_enum</b>        | state of FMC, refer to <a href="#">Table 3-417. fmc_state_enum.</a>   |

Example:

```
/* program a half word at the corresponding address */
fmc_state_enum state;
state = fmc_halfword_program(0x08004000,0xaabb);
```

### **ob\_unlock**

The description of ob\_unlock is shown as below:

**Table 3-434. 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 3-435. 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 3-436. Function ob\_erase**

|                              |                       |
|------------------------------|-----------------------|
| <b>Function name</b>         | ob_erase              |
| <b>Function prototype</b>    | void ob_erase(void);  |
| <b>Function descriptions</b> | erase the option byte |
| <b>Precondition</b>          | ob_unlock             |
| <b>The called functions</b>  | fmc_bank0_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 3-437. 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_bank0_ready_wait   |
| <b>Input parameter{in}</b>   |  |
| <b>ob_wp</b>                 | enable write protection  |
| <b>OB_WPx</b>                | write protect specify sector x                                       |
| <b>OB_WP_ALL</b>             | write protect all sector   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>fmc_state_enum</b>        | state of FMC, refer to <a href="#">Table 3-417. fmc_state_enum</a> . |

Example:

```
/* enable write protection */

fmc_state_enum state;

state = ob_write_protection_enable(OB_WP7);
```

### **ob\_security\_protection\_config**

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

**Table 3-438. 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 the option byte security protection                  |
| <b>Precondition</b>          | ob_unlock  |
| <b>The called functions</b>  | fmc_bank0_ready_wait   |
| <b>Input parameter{in}</b>   |  |
| <b>ob_spc</b>                | specify security protection                                    |
| <b>FMC_NSPC</b>              | no security protection   |
| <b>FMC_USPC</b>              | under security protection                                      |
| <b>Output parameter{out}</b> |  |
| -                            | -  |

| Return value   |  |
|----------------|--|
| fmc_state_enum | state of FMC, refer to <a href="#">Table 3-417. fmc_state_enum</a> . |

Example:

```
/* enable security protection */

fmc_state_enum state;

state = ob_security_protection_config(FMC_USPC);
```

### ob\_user\_write

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

**Table 3-439. Function ob\_user\_write**

|                              |  |
|------------------------------|--|
| Function name                | ob_user_write  |
| Function prototype           | fmc_state_enum ob_user_write(uint8_t ob_fwdgt, uint8_t ob_deepsleep,<br>uint8_t ob_stby, uint8_t ob_boot); |
| Function descriptions        | program option byte user   |
| Precondition                 | ob_unlock  |
| The called functions         | fmc_bank0_ready_wait   |
| Input parameter{in}          |  |
| ob_fwdgt                     | option byte watchdog value   |
| OB_FWDGT_SW                  | software free watchdog   |
| OB_FWDGT_HW                  | hardware free watchdog   |
| Input parameter{in}          |  |
| ob_deepsleep                 | option byte deepsleep reset value  |
| OB_DEEPSLEEP_NRS<br><i>T</i> | no reset when entering deepsleep mode  |
| OB_DEEPSLEEP_RST             | generate a reset instead of entering deepsleep mode  |
| Input parameter{in}          |  |
| ob_stby                      | option byte standby reset value  |
| OB_STDBY_NRST                | no reset when entering standby mode  |
| OB_STDBY_RST                 | generate a reset instead of entering standby mode  |
| Input parameter{in}          |  |
| ob_boot                      | specifies the option byte boot bank value  |
| OB_BOOT_B0                   | boot from bank0  |
| OB_BOOT_B1                   | boot from bank1 or bank0 if bank1 is void  |
| Output parameter{out}        |  |
| -                            | -  |
| Return value                 |  |
| fmc_state_enum               | state of FMC, refer to <a href="#">Table 3-417. fmc_state_enum</a> .                                       |

Example:

```
/* configure user option byte */
```

```
fmc_state_enum state;
state = ob_user_write(OB_FWDGT_HW, OB_DEEPSLEEP_RST, OB_STDBY_RST,
OB_BOOT_B1);
```

### ob\_data\_program

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

**Table 3-440. 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_bank0_ready_wait   |
| <b>Input parameter{in}</b>   |  |
| address                      | 0x1fff804 / 0x1fff806  |
| <b>Input parameter{in}</b>   |  |
| data                         | the byte to be programmed  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>fmc_state_enum</b>        | state of FMC, refer to <a href="#">Table 3-417. fmc_state_enum</a> . |

Example:

```
/* program option bytes data */
fmc_state_enum state;
state = ob_data_program(0x1fff804, 0x56);
```

### ob\_user\_get

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

**Table 3-441. 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 option bytes user |
| <b>Precondition</b>          | -                             |
| <b>The called functions</b>  | -                             |
| <b>Input parameter{in}</b>   |                               |
| -                            | -                             |
| <b>Output parameter{out}</b> |                               |
| -                            | -                             |
| <b>Return value</b>          |                               |

|                |  |
|----------------|--|
| <b>uint8_t</b> | the FMC user option byte values(0x00 – 0xFF) |
|----------------|--|

Example:

```
/* get the FMC user option byte */

uint8_t user;

user = ob_user_get();
```

### **ob\_data\_get**

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

**Table 3-442. Function ob\_data\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | ob_data_get                                 |
| <b>Function prototype</b>    | uint8_t ob_data_get(void);                  |
| <b>Function descriptions</b> | get the FMC option bytes data               |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>uint8_t</b>               | the FMC data option byte values(0x0 – 0xFF) |

Example:

```
/* get the FMC data option byte */

uint8_t data;

data = ob_data_get();
```

### **ob\_write\_protection\_get**

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

**Table 3-443. 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> |  |
| -                            | -  |

| Return value |  |
|--------------|--|
| uint32_t     | the FMC write protection option byte value(0x0 – 0xFFFFFFFF) |

Example:

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

uint32_t wp;

wp = ob_write_protection_get();
```

### ob\_spc\_get

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

**Table 3-444. Function ob\_spc\_get**

|                       |  |
|-----------------------|--|
| Function name         | ob_spc_get                                     |
| Function prototype    | FlagStatus ob_spc_get(void);                   |
| Function descriptions | get option byte security protection code value |
| Precondition          | -  |
| The called functions  | -  |
| Input parameter{in}   |  |
| -                     | -  |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| FlagStatus            | SET or RESET                                   |

Example:

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

FlagStatus spc;

spc = ob_spc_get();
```

### fmc\_flag\_get

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

**Table 3-445. Function fmc\_flag\_get**

|                       |   |
|-----------------------|---|
| Function name         | fmc_flag_get  |
| Function prototype    | FlagStatus fmc_flag_get(uint32_t flag);                         |
| Function descriptions | check flag is set or not  |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |
| flag                  | FMC flag, refer to <a href="#">Table 3-419. fmc_flag_enum</a> . |
| Output parameter{out} |   |

|              |              |
|--------------|--------------|
| -            | -            |
| Return value |              |
| FlagStatus   | SET or RESET |

Example:

```
/* get FMC flag */

FlagStatus flag;

flag = fmc_flag_get(FMC_FLAG_BANK0_END);
```

### fmc\_flag\_clear

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

**Table 3-446. 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>     | check flag is set or not  |
| <b>Precondition</b>              | -   |
| <b>The called functions</b>      | -   |
| <b>Input parameter{in}</b>       |   |
| <b>flag</b>                      | FMC flag, refer to <a href="#">Table 3-419. fmc_flag_enum</a> . |
| <i>FMC_FLAG_BANK0_P<br/>GERR</i> | FMC bank0 operation error flag                                  |
| <i>FMC_FLAG_BANK0_W<br/>PERR</i> | FMC bank0 erase/program protection error flag                   |
| <i>FMC_FLAG_BANK0_E<br/>ND</i>   | FMC bank0 end of operation flag                                 |
| <i>FMC_FLAG_BANK1_P<br/>GERR</i> | FMC bank1 operation error flag                                  |
| <i>FMC_FLAG_BANK1_W<br/>PERR</i> | FMC bank1 erase/program protection error flag                   |
| <i>FMC_FLAG_BANK1_E<br/>ND</i>   | FMC bank1 end of operation flag                                 |
| <b>Output parameter{out}</b>     |   |
| -                                | -   |
| <b>Return value</b>              |   |
| -                                | -   |

Example:

```
/* clear FMC flag */

fmc_flag_clear(FMC_FLAG_BANK0_END);
```

### fmc\_interrupt\_enable

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

**Table 3-447. Function fmc\_interrupt\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | fmc_interrupt_enable  |
| <b>Function prototype</b>    | void fmc_interrupt_enable(fmc_interrupt_enum interrupt);                  |
| <b>Function descriptions</b> | enable FMC interrupt  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| interrupt                    | FMC interrupt, refer to <a href="#">Table 3-418. fmc_interrupt_enum</a> . |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable FMC interrupt */
fmc_interrupt_enable(FMC_INT_BANK0_END);
```

### fmc\_interrupt\_disable

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

**Table 3-448. Function fmc\_interrupt\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | fmc_interrupt_disable   |
| <b>Function prototype</b>    | void fmc_interrupt_disable(fmc_interrupt_enum interrupt);                 |
| <b>Function descriptions</b> | disable FMC interrupt   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| interrupt                    | FMC interrupt, refer to <a href="#">Table 3-418. fmc_interrupt_enum</a> . |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable FMC interrupt */
fmc_interrupt_disable(FMC_INT_BANK0_END);
```

### fmc\_interrupt\_flag\_get

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

**Table 3-449. Function fmc\_interrupt\_flag\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | fmc_interrupt_flag_get  |
| <b>Function prototype</b>    | FlagStatus fmc_interrupt_flag_get(fmc_interrupt_flag_enum int_flag);                |
| <b>Function descriptions</b> | get FMC interrupt flag state  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| int_flag                     | FMC interrupt flag, refer to <a href="#">Table 3-420. fmc_interrupt_flag_enum</a> . |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| FlagStatus                   | SET or RESET  |

Example:

```
/* get FMC flag */

FlagStatus flag;

flag = fmc_interrupt_flag_get (FMC_INT_FLAG_BANK0_PGERR);
```

### fmc\_interrupt\_flag\_clear

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

**Table 3-450. 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 int_flag);             |
| <b>Function descriptions</b> | clear FMC interrupt flag  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| int_flag                     | FMC interrupt flag, refer to <a href="#">Table 3-420. fmc_interrupt_flag_enum</a> . |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* clear FMC flag */

fmc_interrupt_flag_get(FMC_INT_FLAG_BANK0_PGERR);
```

## 3.15. 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 [3.15.1](#), the FWDGT firmware functions are introduced in chapter [3.15.2](#).

### 3.15.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

**Table 3-451. FWDGT Registers**

| Registers  | Descriptions       |
|------------|--------------------|
| FWDGT_CTL  | control register   |
| FWDGT_PSC  | prescaler register |
| FWDGT_RLD  | reload register    |
| FWDGT_STAT | status register    |

### 3.15.2. Descriptions of Peripheral functions

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

**Table 3-452. FWDGT firmware function**

| Function name                | Function description  |
|------------------------------|---|
| fwdgt_write_enable           | enable write access to FWDGT_PSC and FWDGT_RLD              |
| fwdgt_write_disable          | disable write access to FWDGT_PSC and FWDGT_RLD             |
| fwdgt_enable                 | start the free watchdog timer counter                       |
| fwdgt_prescaler_value_config | configure the free watchdog timer counter prescaler value   |
| fwdgt_reload_value_config    | configure the free watchdog timer counter reload value      |
| fwdgt_counter_reload         | reload the counter of FWDGT                                 |
| fwdgt_config                 | configure counter reload value, and prescaler divider value |
| fwdgt_flag_get               | get flag state of FWDGT                                     |

#### fwdgt\_write\_enable

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

**Table 3-453. Function fwdgt\_write\_enable**

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

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

Example:

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

fwdgt_write_enable();
```

### **fwdgt\_write\_disable**

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

**Table 3-454. 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 3-455. 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> |                                       |

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

Example:

```
/* start the free watchdog timer counter */

fwdgt_enable();
```

### fwdgt\_prescaler\_value\_config

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

**Table 3-456. 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 3-457. 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 0xFFFF */
ErrStatus flag;
flag = fwdgt_reload_value_config (0xFFFF);
```

### fwdgt\_counter\_reload

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

**Table 3-458. 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 3-459. Function fwdgt\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | fwdgt_config  |
| <b>Function prototype</b>    | ErrStatus fwdgt_config(uint16_t reload_value, uint8_t prescaler_div); |
| <b>Function descriptions</b> | configure counter reload value, and prescaler divider value           |

|                              |                                       |
|------------------------------|---------------------------------------|
| <b>Precondition</b>          | -                                     |
| <b>The called functions</b>  | -                                     |
| <b>Input parameter{in}</b>   |                                       |
| <b>reload_value</b>          | specify reload value(0x0000 - 0x0FFF) |
| <b>Input parameter{in}</b>   |                                       |
| <b>prescaler_div</b>         | FWDGT prescaler value-                |
| <i>FWDGT_PSC_DIV4</i>        | FWDGT prescaler set to 4              |
| <i>FWDGT_PSC_DIV8</i>        | FWDGT prescaler set to 8              |
| <i>FWDGT_PSC_DIV16</i>       | FWDGT prescaler set to 16             |
| <i>FWDGT_PSC_DIV32</i>       | FWDGT prescaler set to 32             |
| <i>FWDGT_PSC_DIV64</i>       | FWDGT prescaler set to 64             |
| <i>FWDGT_PSC_DIV128</i>      | FWDGT prescaler set to 128            |
| <i>FWDGT_PSC_DIV256</i>      | FWDGT prescaler set to 256            |
| <b>Output parameter{out}</b> |                                       |
| -                            | -                                     |
| <b>Return value</b>          |                                       |
| <b>ErrStatus</b>             | ERROR or SUCCESS                      |

Example:

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

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

### fwdgt\_flag\_get

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

**Table 3-460. 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);

```

## 3.16. GPIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter [3.16.1](#), the GPIO firmware functions are introduced in chapter [3.16.2](#).

### 3.16.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

**Table 3-461. GPIO Registers**

| Registers   | Descriptions                          |
|-------------|---------------------------------------|
| GPIOx_CTL0  | GPIO port control register 0          |
| GPIOx_CTL1  | GPIO port control register 1          |
| GPIOx_ISTAT | GPIO port input status register       |
| GPIOx_OCTL  | GPIO port output control register     |
| GPIOx_BOP   | GPIO port bit operate register        |
| GPIOx_BC    | GPIO port bit clear register          |
| GPIOx_LOCK  | GPIO port configuration lock register |
| AFIO_EC     | AFIO event control register           |
| AFIO_PCF0   | AFIO port configuration register 0    |
| AFIO_EXTI0  | EXTI sources selection register 0     |
| AFIO_EXTI1  | EXTI sources selection register 1     |
| AFIO_EXTI2  | EXTI sources selection register 2     |
| AFIO_EXTI3  | EXTI sources selection register 3     |
| AFIO_PCF1   | AFIO port configuration register 1    |
| AFIO_PCF2   | AFIO port configuration register 2    |
| AFIO_PCF3   | AFIO port configuration register 3    |
| AFIO_PCF4   | AFIO port configuration register 4    |
| AFIO_PCF5   | AFIO port configuration register 5    |

### 3.16.2. Descriptions of Peripheral functions

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

**Table 3-462. 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                       |

| Function name             | Function description                  |
|---------------------------|---------------------------------------|
| 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            |
| 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_pin_remap1_config    | configure GPIO pin remap1             |
| gpio_ethernet_phy_select  | select ethernet MII or RMII PHY       |
| 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 bit                     |

### gpio\_deinit

The description of gpio\_deinit is shown as below:

**Table 3-463. 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 |
| <b>Input parameter{in}</b>   |  |
| gpio_periph                  | GPIO port  |
| GPIOx                        | GPIOx(x = A, B, C, D, E, F, G, H, I)               |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* reset GPIOA */
gpio_deinit(GPIOA);
```

### gpio\_afio\_deinit

The description of gpio\_afio\_deinit is shown as below:

**Table 3-464. Function gpio\_afio\_deinit**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | gpio_afio_deinit                                   |
| <b>Function prototype</b>    | void gpio_afio_deinit(void);                       |
| <b>Function descriptions</b> | reset alternate function I/O(AFIO)                 |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | rcu_periph_reset_enable / rcu_periph_reset_disable |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* reset alternate function */
gpio_afio_deinit();
```

### gpio\_init

The description of gpio\_init is shown as below:

**Table 3-465. Function gpio\_init**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | gpio_init   |
| <b>Function prototype</b>    | void gpio_init(uint32_t gpio_periph,uint32_t mode,uint32_t speed,uint32_t pin); |
| <b>Function descriptions</b> | GPIO parameter initialization   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| gpio_periph                  | GPIO port   |
| GPIOx                        | GPIOx(x = A, B, C, D, E, F, G, H, I)  |
| <b>Input parameter{in}</b>   |   |
| mode                         | gpio pin mode   |
| GPIO_MODE_AIN                | analog input mode   |
| GPIO_MODE_IN_FLOATING        | floating input mode   |
| GPIO_MODE_IPD                | pull-down input mode  |
| GPIO_MODE_IPU                | pull-up input mode  |
| GPIO_MODE_OUT_OD             | GPIO output with open-drain   |
| GPIO_MODE_OUT_PP             | GPIO output with push-pull  |
| GPIO_MODE_AF_OD              | AFIO output with open-drain   |
| GPIO_MODE_AF_PP              | 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      |
| Input parameter{in}      |                             |
| <b>pin</b>               | GPIO pin                    |
| <i>GPIO_PIN_x</i>        | <i>GPIO_PIN_x(x=0..15)</i>  |
| <i>GPIO_PIN_ALL</i>      | all pins                    |
| Output parameter{out}    |                             |
| -                        | -                           |
| Return value             |                             |
| -                        | -                           |

Example:

```
/* configure PA0 as analog input mode */
gpio_init(GPIOA, GPIO_MODEAIN, GPIO_OSPEED_50MHZ, GPIO_PIN_0);
```

### gpio\_bit\_set

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

**Table 3-466. 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>  | -   |
| Input parameter{in}          |   |
| <b>gpio_periph</b>           | GPIO port   |
| <i>GPIOx</i>                 | <i>GPIOx(x = A, B, C, D, E, F, G, H, I)</i>           |
| Input parameter{in}          |   |
| <b>pin</b>                   | GPIO pin  |
| <i>GPIO_PIN_x</i>            | <i>GPIO_PIN_x(x=0..15)</i>                            |
| <i>GPIO_PIN_ALL</i>          | all pins  |
| Output parameter{out}        |   |
| -                            | -   |
| Return value                 |   |
| -                            | -   |

Example:

```
/* set PA0 */

gpio_bit_set(GPIOA, GPIO_PIN_0);
```

### gpio\_bit\_reset

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

**Table 3-467. 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 bit                                      |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| gpio_periph                  | GPIO port   |
| GPIOx                        | GPIOx(x = A, B, C, D, E, F, G, H, I)                    |
| <b>Input parameter{in}</b>   |   |
| pin                          | GPIO pin  |
| GPIO_PIN_x                   | GPIO_PIN_x(x=0..15)                                     |
| GPIO_PIN_ALL                 | all pins  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* reset PA0 */

gpio_bit_reset(GPIOA, GPIO_PIN_0);
```

### gpio\_bit\_write

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

**Table 3-468. 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>   |  |
| gpio_periph                  | GPIO port  |
| GPIOx                        | GPIOx(x = A, B, C, D, E, F, G, H, I)   |
| <b>Input parameter{in}</b>   |  |
| pin                          | GPIO pin   |

|                              |                                  |
|------------------------------|----------------------------------|
| <code>GPIO_PIN_x</code>      | <code>GPIO_PIN_x(x=0..15)</code> |
| <code>GPIO_PIN_ALL</code>    | all pins                         |
| <b>Input parameter{in}</b>   |                                  |
| <code>bit_value</code>       | SET or RESET                     |
| <code>RESET</code>           | reset the port pin               |
| <code>SET</code>             | 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 3-469. Function `gpio_port_write`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | <code>gpio_port_write</code>   |
| <b>Function prototype</b>    | <code>void gpio_port_write(uint32_t gpio_periph,uint16_t data);</code> |
| <b>Function descriptions</b> | write data to the specified GPIO port                                  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <code>gpio_periph</code>     | GPIO port  |
| <code>GPIOx</code>           | <code>GPIOx(x = A, B, C, D, E, F, G, H, I)</code>                      |
| <b>Input parameter{in}</b>   |  |
| <code>data</code>            | 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 3-470. Function gpio\_input\_bit\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | gpio_input_bit_get  |
| <b>Function prototype</b>    | FlagStatus gpio_input_bit_get(uint32_t gpio_periph,uint32_t pin); |
| <b>Function descriptions</b> | get GPIO pin input status   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| gpio_periph                  | GPIO port   |
| GPIOx                        | GPIOx(x = A, B, C, D, E, F, G, H, I)                              |
| <b>Input parameter{in}</b>   |   |
| pin                          | GPIO pin  |
| GPIO_PIN_x                   | GPIO_PIN_x(x=0..15)   |
| GPIO_PIN_ALL                 | all pins  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| FlagStatus                   | 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 3-471. 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>   |   |
| gpio_periph                  | GPIO port   |
| GPIOx                        | GPIOx(x = A, B, C, D, E, F, G, H, I)                |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| uint16_t                     | 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 3-472. Function gpio\_output\_bit\_get**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | gpio_output_bit_get  |
| <b>Function prototype</b>    | FlagStatus gpio_output_bit_get(uint32_t gpio_periph,uint32_t pin); |
| <b>Function descriptions</b> | get GPIO pin output status   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| gpio_periph                  | GPIO port  |
| GPIOx                        | GPIOx(x = A, B, C, D, E, F, G, H, I)                               |
| <b>Input parameter{in}</b>   |  |
| pin                          | GPIO pin   |
| GPIO_PIN_x                   | GPIO_PIN_x(x=0..15)  |
| GPIO_PIN_ALL                 | all pins   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| FlagStatus                   | 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 3-473. 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>   |  |
| gpio_periph                  | GPIO port  |
| GPIOx                        | GPIOx(x = A, B, C, D, E, F, G, H, I)                 |

| Output parameter{out} |           |
|-----------------------|-----------|
| -                     | -         |
| Return value          |           |
| <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 3-474. Function gpio\_pin\_remap\_config**

| Function name                     | gpio_pin_remap_config  |
|-----------------------------------|--|
| Function prototype                | void gpio_pin_remap_config(uint32_t gpio_remap, ControlStatus newvalue); |
| Function descriptions             | configure GPIO pin remap   |
| Precondition                      | -  |
| The called functions              | -  |
| Input parameter{in}               |  |
| <b>gpio_remap</b>                 | select the pin to remap  |
| <b>GPIO_SPI0_REMAP</b>            | SPI0 remapping   |
| <b>GPIO_I2C0_REMAP</b>            | I2C0 remapping   |
| <b>GPIO_USART0_REMAP</b>          | USART0 remapping   |
| <b>GPIO_USART1_REMAP</b>          | USART1 remapping   |
| <b>GPIO_USART2_PARTIAL_REMAP</b>  | USART2 partial remapping   |
| <b>GPIO_USART2_FULL_REMAP</b>     | USART2 full remapping  |
| <b>GPIO_TIMER0_PARTIAL_REMAP</b>  | TIMER0 partial remapping   |
| <b>GPIO_TIMER0_FULL_REMAP</b>     | TIMER0 full remapping  |
| <b>GPIO_TIMER1_PARTIAL_REMAP0</b> | TIMER1 partial remapping   |
| <b>GPIO_TIMER1_PARTIAL_REMAP1</b> | TIMER1 partial remapping   |
| <b>GPIO_TIMER1_FULL_REMAP</b>     | TIMER1 full remapping  |
| <b>GPIO_TIMER2_PARTIAL</b>        | TIMER2 partial remapping   |

|  |   |
|--|---|
| <code>AL_REMAP</code>                  |   |
| <code>GPIO_TIMER2_FULL_REMAP</code>    | TIMER2 full remapping                               |
| <code>GPIO_TIMER3_REMAP</code>         | TIMER3 remapping                                    |
| <code>GPIO_CAN0_PARTIAL_REMAP</code>   | CAN0 partial remapping                              |
| <code>GPIO_CAN0_FULL_REMAP</code>      | CAN0 full remapping                                 |
| <code>GPIO_PD01_REMAP</code>           | PD01 remapping                                      |
| <code>GPIO_TIMER4CH3_IR_EMAP</code>    | TIMER4 channel3 internal remapping                  |
| <code>GPIO_ADC0_ETRGIN_S_REMAP</code>  | ADC0 external trigger inserted conversion remapping |
| <code>GPIO_ADC0_ETRGRE_G_REMAP</code>  | ADC0 external trigger regular conversion remapping  |
| <code>GPIO_ADC1_ETRGIN_S_REMAP</code>  | ADC1 external trigger inserted conversion remapping |
| <code>GPIO_ADC1_ETRGRE_G_REMAP</code>  | ADC1 external trigger regular conversion remapping  |
| <code>GPIO_ENET_REMAP</code>           | ENET remapping                                      |
| <code>GPIO_CAN1_REMAP</code>           | CAN1 remapping                                      |
| <code>GPIO_SWJ_NONJTRS_T_REMAP</code>  | full SWJ(JTAG-DP + SW-DP), but without NJTRST       |
| <code>GPIO_SWJ_SWDPENABLE_REMAP</code> | JTAG-DP disabled and SW-DP enabled                  |
| <code>GPIO_SWJ_DISABLE_REMAP</code>    | JTAG-DP disabled and SW-DP disabled                 |
| <code>GPIO_SPI2_REMAP</code>           | SPI2 remapping                                      |
| <code>GPIO_TIMER1IT1_REMAP</code>      | TIMER1 internal trigger 1 remapping                 |
| <code>GPIO_PTP_PPS_REMAP</code>        | ethernet PTP PPS remapping                          |
| <code>GPIO_TIMER8_REMAP</code>         | TIMER8 remapping                                    |
| <code>GPIO_TIMER9_REMAP</code>         | TIMER9 remapping                                    |
| <code>GPIO_TIMER10_REMAP</code>        | TIMER10 remapping                                   |
| <code>GPIO_TIMER12_REMAP</code>        | TIMER12 remapping                                   |
| <code>GPIO_TIMER13_REMAP</code>        | TIMER13 remapping                                   |

|  |                              |
|--|------------------------------|
| <i>GPIO_EXMC_NADV_R</i><br><i>EMAP</i> | EXMC_NADV connect/disconnect |
| <b>Input parameter{in}</b>             |                              |
| <i>newvalue</i>                        | 是否使能                         |
| <i>ENABLE</i>                          | 使能                           |
| <i>DISABLE</i>                         | 禁能                           |
| <b>Output parameter{out}</b>           |                              |
| -                                      | -                            |
| <b>Return value</b>                    |                              |
| -                                      | -                            |

Example:

```
/*enable SPI0 remapping*/
gpio_pin_remap_config(GPIO_SPI0_REMAP, ENABLE);
```

### gpio\_pin\_remap1\_config

The description of gpio\_pin\_remap1\_config is shown as below:

**Table 3-475. Function gpio\_pin\_remap1\_config**

|   |   |
|---|---|
| <b>Function name</b>                            | gpio_pin_remap1_config  |
| <b>Function prototype</b>                       | void gpio_pin_remap1_config(uint8_t remap_reg, uint32_t remap, ControlStatus newvalue); |
| <b>Function descriptions</b>                    | configure GPIO pin remap1   |
| <b>Precondition</b>                             | -   |
| <b>The called functions</b>                     | -   |
| <b>Input parameter{in}</b>                      |   |
| <i>remap_reg</i>                                | AFIO port configuration register  |
| <i>GPIO_PCF2</i>                                | AFIO port configuration register 2  |
| <i>GPIO_PCF3</i>                                | AFIO port configuration register 3  |
| <i>GPIO_PCF4</i>                                | AFIO port configuration register 4  |
| <i>GPIO_PCF5</i>                                | AFIO port configuration register 5  |
| <b>Input parameter{in}</b>                      |   |
| <i>remap</i>                                    | select the pin to remap   |
| <i>GPIO_PCF2_DCI_VSY</i><br><i>NC_PG9_REMAP</i> | DCI VSYNC remapped to PG9   |
| <i>GPIO_PCF2_DCI_VSY</i><br><i>NC_PI5_REMAP</i> | DCI VSYNC remapped to PI5   |
| <i>GPIO_PCF2_DCI_D0_</i><br><i>PC6_REMAP</i>    | DCI D0 remapped to PC6  |
| <i>GPIO_PCF2_DCI_D0_</i><br><i>PH9_REMAP</i>    | DCI D0 remapped to PH9  |
| <i>GPIO_PCF2_DCI_D1_</i>                        | DCI D1 remapped to PC7  |

|                                    |                         |
|------------------------------------|-------------------------|
| <i>PC7_REMAP</i>                   |                         |
| <i>GPIO_PCF2_DCI_D1_PH10_REMAP</i> | DCI D1 remapped to PH10 |
| <i>GPIO_PCF2_DCI_D2_PE0_REMAP</i>  | DCI D2 remapped to PE0  |
| <i>GPIO_PCF2_DCI_D2_PG10_REMAP</i> | DCI D2 remapped to PG10 |
| <i>GPIO_PCF2_DCI_D2_PH11_REMAP</i> | DCI D2 remapped to PH11 |
| <i>GPIO_PCF2_DCI_D3_PE1_REMAP</i>  | DCI D3 remapped to PE1  |
| <i>GPIO_PCF2_DCI_D3_PG11_REMAP</i> | DCI D3 remapped to PG11 |
| <i>GPIO_PCF2_DCI_D3_PH12_REMAP</i> | DCI D3 remapped to PH12 |
| <i>GPIO_PCF2_DCI_D4_PE4_REMAP</i>  | DCI D4 remapped to PE4  |
| <i>GPIO_PCF2_DCI_D4_PH14_REMAP</i> | DCI D4 remapped to PH14 |
| <i>GPIO_PCF2_DCI_D5_PD3_REMAP</i>  | DCI D5 remapped to PD3  |
| <i>GPIO_PCF2_DCI_D5_PI4_REMAP</i>  | DCI D5 remapped to PI4  |
| <i>GPIO_PCF2_DCI_D6_PE5_REMAP</i>  | DCI D6 remapped to PE5  |
| <i>GPIO_PCF2_DCI_D6_PI6_REMAP</i>  | DCI D6 remapped to PI6  |
| <i>GPIO_PCF2_DCI_D7_PE6_REMAP</i>  | DCI D7 remapped to PE6  |
| <i>GPIO_PCF2_DCI_D7_PI7_REMAP</i>  | D7 remapped to PI7      |
| <i>GPIO_PCF2_DCI_D8_PH6_REMAP</i>  | D8 remapped to PH6      |
| <i>GPIO_PCF2_DCI_D8_PI1_REMAP</i>  | DCI D8 remapped to PI1  |
| <i>GPIO_PCF2_DCI_D9_PH7_REMAP</i>  | DCI D9 remapped to PH7  |
| <i>GPIO_PCF2_DCI_D9_PI2_REMAP</i>  | DCI D9 remapped to PI2  |
| <i>GPIO_PCF2_DCI_D10_PD6_REMAP</i> | DCI D10 remapped to PD6 |
| <i>GPIO_PCF2_DCI_D10_PI3_REMAP</i> | DCI D10 remapped to PI3 |

|   |                           |
|---|---------------------------|
| <code>GPIO_PCF2_DCI_D11_PF10_REMAP</code>   | DCI D11 remapped to PF10  |
| <code>GPIO_PCF2_DCI_D11_PH15_REMAP</code>   | DCI D11 remapped to PH15  |
| <code>GPIO_PCF2_DCI_D12_PG6_REMAP</code>    | DCI D12 remapped to PG6   |
| <code>GPIO_PCF2_DCI_D13_PG15_REMAP</code>   | DCI D12 remapped to PG15  |
| <code>GPIO_PCF2_DCI_D13_PI0_REMAP</code>    | DCI D13 remapped to PI0   |
| <code>GPIO_PCF2_DCI_HSY_NC_PH8_REMAP</code> | DCI HSYNC to PH8          |
| <code>GPIO_PCF2_PH01_REMAP</code>           | PH0/PH1 remapping         |
| <code>GPIO_PCF3_TLI_B5_PA3_REMAP</code>     | TLI B5 remapped to PA3    |
| <code>GPIO_PCF3_TLI_VSY_NC_PA4_REMAP</code> | TLI VSYNC remapped to PA4 |
| <code>GPIO_PCF3_TLI_G2_PA6_REMAP</code>     | TLI G2 remapped to PA6    |
| <code>GPIO_PCF3_TLI_R6_PA8_REMAP</code>     | TLI R6 remapped to PA8    |
| <code>GPIO_PCF3_TLI_R4_PA11_REMAP</code>    | TLI R4 remapped to PA11   |
| <code>GPIO_PCF3_TLI_R5_PA12_REMAP</code>    | TLI R5 remapped to PA12   |
| <code>GPIO_PCF3_TLI_R3_PB0_REMAP</code>     | TLI R3 remapped to PB0    |
| <code>GPIO_PCF3_TLI_R6_PB1_REMAP</code>     | TLI R6 remapped to PB1    |
| <code>GPIO_PCF3_TLI_B6_PB8_REMAP</code>     | TLI B6 remapped to PB8    |
| <code>GPIO_PCF3_TLI_B7_PB9_REMAP</code>     | TLI B7 remapped to PB9    |
| <code>GPIO_PCF3_TLI_G4_PB10_REMAP</code>    | TLI G4 remapped to PB10   |
| <code>GPIO_PCF3_TLI_G5_PB11_REMAP</code>    | TLI G5 remapped to PB11   |
| <code>GPIO_PCF3_TLI_HSY_NC_PC6_REMAP</code> | TLI HSYNC remapped to PC6 |
| <code>GPIO_PCF3_TLI_G6_PC7_REMAP</code>     | TLI G6 remapped to PC7    |
| <code>GPIO_PCF3_TLI_R2_PC10</code>          | TLI R2 remapped to PC10   |

|                                     |                          |
|-------------------------------------|--------------------------|
| <i>C10_REMAP</i>                    |                          |
| <i>GPIO_PCF3_TLI_G7_PD3_REMAP</i>   | TLI G7 remapped to PD3   |
| <i>GPIO_PCF3_TLI_B2_P_D6_REMAP</i>  | TLI B2 remapped to PD6   |
| <i>GPIO_PCF3_TLI_B3_P_D10_REMAP</i> | TLI B3 remapped to PD10  |
| <i>GPIO_PCF3_TLI_B0_P_E4_REMAP</i>  | TLI B0 remapped to PE4   |
| <i>GPIO_PCF3_TLI_G0_PE5_REMAP</i>   | TLI G0 remapped to PE5   |
| <i>GPIO_PCF3_TLI_G1_PE6_REMAP</i>   | TLI G1 remapped to PE6   |
| <i>GPIO_PCF3_TLI_G3_PE11_REMAP</i>  | TLI G3 remapped to PE11  |
| <i>GPIO_PCF3_TLI_B4_P_E12_REMAP</i> | TLI B4 remapped to PE12  |
| <i>GPIO_PCF3_TLI_DE_PE13_REMAP</i>  | TLI DE remapped to PE13  |
| <i>GPIO_PCF3_TLI_CLK_PE14_REMAP</i> | TLI CLK remapped to PE14 |
| <i>GPIO_PCF3_TLI_R7_P_E15_REMAP</i> | TLI R7 remapped to PE15  |
| <i>GPIO_PCF3_TLI_DE_PF10_REMAP</i>  | TLI DE remapped to PF10  |
| <i>GPIO_PCF3_TLI_R7_P_G6_REMAP</i>  | TLI R7 remapped to PG6   |
| <i>GPIO_PCF3_TLI_CLK_PG7_REMAP</i>  | TLI CLK remapped to PG7  |
| <i>GPIO_PCF3_TLI_G3_PG10_REMAP</i>  | TLI G3 remapped to PG10  |
| <i>GPIO_PCF3_TLI_B2_P_G10_REMAP</i> | TLI B2 remapped to PG10  |
| <i>GPIO_PCF3_TLI_B3_P_G11_REMAP</i> | TLI B3 remapped to PG11  |
| <i>GPIO_PCF4_TLI_B4_P_G12_REMAP</i> | B4 remapped to PG12      |
| <i>GPIO_PCF4_TLI_B1_P_G12_REMAP</i> | B1 remapped to PG12      |
| <i>GPIO_PCF4_TLI_R0_P_H2_REMAP2</i> | R0 remapped to PH2       |
| <i>GPIO_PCF4_TLI_R1_P_H3_REMAP</i>  | TLI R1 remapped to PH3   |

|  |                            |
|--|----------------------------|
| <code>GPIO_PCF4_TLI_R2_P_H8_REMAP</code>     | TLI R2 remapped to PH8     |
| <code>GPIO_PCF4_TLI_R3_P_H9_REMAP</code>     | TLI R3 remapped to PH9     |
| <code>GPIO_PCF4_TLI_R4_P_H10_REMAP</code>    | TLI R4 remapped to PH10    |
| <code>GPIO_PCF4_TLI_R5_P_H11_REMAP</code>    | TLI R5 remapped to PH11    |
| <code>GPIO_PCF4_TLI_R6_P_H12_REMAP</code>    | TLI R6 remapped to PH12    |
| <code>GPIO_PCF4_TLI_G2_PH13_REMAP</code>     | TLI G2 remapped to PH13    |
| <code>GPIO_PCF4_TLI_G3_PH14_REMAP</code>     | TLI G3 remapped to PH14    |
| <code>GPIO_PCF4_TLI_G4_PH15_REMAP</code>     | TLI G4 remapped to PH15    |
| <code>GPIO_PCF4_TLI_G5_PI0_REMAP</code>      | TLI G5 remapped to PI0     |
| <code>GPIO_PCF4_TLI_G6_PI1_REMAP</code>      | TLI G6 remapped to PI1     |
| <code>GPIO_PCF4_TLI_G7_PI2_REMAP</code>      | TLI G7 remapped to PI2     |
| <code>GPIO_PCF4_TLI_B4_P_I4_REMAP</code>     | TLI B4 remapped to PI4     |
| <code>GPIO_PCF4_TLI_B5_P_I5_REMAP</code>     | TLI B5 remapped to PI5     |
| <code>GPIO_PCF4_TLI_B6_P_I6_REMAP</code>     | TLI B6 remapped to PI6     |
| <code>GPIO_PCF4_TLI_B7_P_I7_REMAP</code>     | TLI B7 remapped to PI7     |
| <code>GPIO_PCF4_TLI_VSY_NC_PI9_REMAP</code>  | TLI VSYNC remapped to PI9  |
| <code>GPIO_PCF4_TLI_HSY_NC_PI10_REMAP</code> | TLI HSYNC remapped to PI10 |
| <code>GPIO_PCF4_TLI_R0_P_H4_REMAP</code>     | TLI R0 remapped to PH4     |
| <code>GPIO_PCF4_TLI_R1_P_I3_REMAP</code>     | TLI R1 remapped to PI3     |
| <code>GPIO_PCF4_SPI1_SC_K_PD3_REMAP</code>   | SPI1 SCK remapped to PD3   |
| <code>GPIO_PCF4_SPI2_MO_SI_PD6_REMAP</code>  | SPI2 MOSI remapped to PD6  |
| <code>GPIO_PCF5_I2C2_RE</code>               | I2C2 remapping 0           |

| <i>MAP0</i>                              |                                       |
|--|---------------------------------------|
| <i>GPIO_PCF5_I2C2_RE<br/>MAP1</i>        | I2C2 remapping 1                      |
| <i>GPIO_PCF5_TIMER1_<br/>CH0_REMAP</i>   | TIMER1 CH0 remapped to PA5            |
| <i>GPIO_PCF5_TIMER4_<br/>REMAP</i>       | TIMER4 CH0 remapping                  |
| <i>GPIO_PCF5_TIMER7_<br/>CHON_REMAP0</i> | TIMER7 CHON remapping 0               |
| <i>GPIO_PCF5_TIMER7_<br/>CHON_REMAP1</i> | TIMER7 CHON remapping 1               |
| <i>GPIO_PCF5_TIMER7_<br/>CH_REMAP</i>    | TIMER7 CH remapping                   |
| <i>GPIO_PCF5_I2C1_RE<br/>MAP0</i>        | I2C1 remapping 0                      |
| <i>GPIO_PCF5_I2C1_RE<br/>MAP1</i>        | I2C1 remapping 1                      |
| <i>GPIO_PCF5_SPI1_NS<br/>CK_REMAP0</i>   | SPI1 NSS/SCK remapping 0              |
| <i>GPIO_PCF5_SPI1_NS<br/>CK_REMAP1</i>   | SPI1 NSS/SCK remapping 1              |
| <i>GPIO_PCF5_SPI1_IO_<br/>REMAP0</i>     | SPI1 MISO/MOSI remapping 0            |
| <i>GPIO_PCF5_SPI1_IO_<br/>REMAP1</i>     | SPI1 MISO/MOSI remapping 1            |
| <i>GPIO_PCF5_UART3_<br/>REMAP</i>        | UART3 remapping                       |
| <i>GPIO_PCF5_TIMER11_<br/>_REMAP</i>     | TIMER11 remapping                     |
| <i>GPIO_PCF5_CAN0_A<br/>DD_REMAP</i>     | CAN0 addition remapping               |
| <i>GPIO_PCF5_ENET_TX<br/>D3_REMAP</i>    | ETH_RXD3 remapped to PE2              |
| <i>GPIO_PCF5_PPS_HI_<br/>REMAP</i>       | ETH_PPS_OUT remapped to PG8           |
| <i>GPIO_PCF5_ENET_TX<br/>D01_REMAP</i>   | ETH_TX_EN/ETH_RXD0/ETH_RXD1 remapping |
| <i>GPIO_PCF5_ENET_C<br/>RSCOL_REMAP</i>  | ETH_MII_CRS/ETH_MII_COL remapping     |
| <i>GPIO_PCF5_ENET_R<br/>X_HI_REMAP</i>   | ETH_RXD2/ETH_RXD3/ETH_RX_ER remapping |
| <i>GPIO_PCF5_UART6_<br/>REMAP</i>        | UART6 remapping                       |

|   |                             |
|---|-----------------------------|
| <code>GPIO_PCF5_USART5_CK_PG7_REMAP</code>    | USART5 CK remapped to PG7   |
| <code>GPIO_PCF5_USART5_RTS_PG12_REMAP</code>  | USART5 RTS remapped to PG12 |
| <code>GPIO_PCF5_USART5_CTS_PG13_REMAP</code>  | USART5 CTS remapped to PG13 |
| <code>GPIO_PCF5_USART5_TX_PG14_REMAP</code>   | USART5 TX remapped to PG14  |
| <code>GPIO_PCF5_USART5_RX_PG9_REMAP</code>    | USART5 RX remapped to PG9   |
| <code>GPIO_PCF5_EXMC_S_DNWE_PC0_REMAP</code>  | EXMC SDNWE remapped to PC0  |
| <code>GPIO_PCF5_EXMC_S_DCKE0_PC3_REMAP</code> | EXMC SDCKE0 remapped to PC3 |
| <code>GPIO_PCF5_EXMC_S_DCKE1_PB5_REMAP</code> | EXMC SDCKE1 remapped to PB5 |
| <code>GPIO_PCF5_EXMC_S_DNE0_PC2_REMAP</code>  | EXMC SDNE0 remapped to PC2  |
| <code>GPIO_PCF5_EXMC_S_DNE1_PB6_REMAP</code>  | EXMC SDNE1 remapped to PB6  |
| <b>Input parameter{in}</b>                    |                             |
| <code>newvalue</code>                         | 是否使能                        |
| <code>ENABLE</code>                           | 使能                          |
| <code>DISABLE</code>                          | 禁能                          |
| <b>Output parameter{out}</b>                  |                             |
| -   | -                           |
| <b>Return value</b>                           |                             |
| -   | -                           |

Example:

```
/*enable GPIO_PCF2 UART3 remapping */
gpio_pin_remap_config(GPIO_PCF2, GPIO_PCF5_UART3_REMAP, ENABLE);
```

### gpio\_ethernet\_phy\_select

The description of gpio\_ethernet\_phy\_select is shown as below:

**Table 3-476. Function gpio\_ethernet\_phy\_select**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | gpio_ethernet_phy_select   |
| <b>Function prototype</b>    | <code>void gpio_ethernet_phy_select(uint32_t gpio_enetsel);</code> |
| <b>Function descriptions</b> | select ethernet MII or RMII PHY                                    |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |

| Input parameter{in}       |  |
|---------------------------|--|
| <b>enet_sel</b>           | ethernet MII or RMII PHY selection                     |
| <b>GPIO_ENET_PHY_MII</b>  | configure ethernet MAC for connection with an MII PHY  |
| <b>GPIO_ENET_PHY_RMII</b> | configure ethernet MAC for connection with an RMII PHY |
| Output parameter{out}     |  |
| -                         | -  |
| Return value              |  |
| -                         | -  |

Example:

```
/* configure ethernet MAC for connection with an RMII PHY */
gpio_ethernet_phy_select(GPIO_ENET_PHY_RMII);
```

### **gpio\_exti\_source\_select**

The description of **gpio\_exti\_source\_select** is shown as below:

**Table 3-477. Function gpio\_exti\_source\_select**

| <b>Function name</b>          | gpio_exti_source_select   |
|-------------------------------|---|
| <b>Function prototype</b>     | void gpio_exti_source_select(uint8_t gpio_outputport,uint8_t gpio_outputpin); |
| <b>Function descriptions</b>  | select GPIO pin exti sources  |
| <b>Precondition</b>           | -   |
| <b>The called functions</b>   | -   |
| Input parameter{in}           |   |
| <b>gpio_outputport</b>        | gpio event output port  |
| <b>GPIO_PORT_SOURCE_GPIOn</b> | output port source(x = A, B, C, D, E, F, G, H, I)                             |
| Input parameter{in}           |   |
| <b>gpio_outputpin</b>         | gpio event output pin   |
| <b>GPIO_PIN_SOURCE_x</b>      | pin number(x=0..15)   |
| Output parameter{out}         |   |
| -                             | -   |
| Return value                  |   |
| -                             | -   |

Example:

```
/* configure 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 3-478. Function gpio\_event\_output\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | gpio_event_output_config   |
| <b>Function prototype</b>    | void gpio_event_output_config(uint8_t gpio_outputport,uint8_t gpio_outputpin); |
| <b>Function descriptions</b> | configure GPIO pin event output  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| gpio_outputport              | gpio event output port   |
| GPIO_EVENT_PORT_GPIOx        | event output port x(x = A, B, C, D, E)   |
| <b>Input parameter{in}</b>   |  |
| gpio_outputpin               | gpio event output pin  |
| GPIO_EVENT_PIN_x             | pin number(x=0..15)  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure 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 3-479. 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 3-480. 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 3-481. 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  |
| <b>GPIOx</b>                 | GPIOx(x = A, B, C, D, E, F, G, H, I)                   |
| <b>Input parameter{in}</b>   |  |
| <b>pin</b>                   | GPIO pin   |
| <b>GPIO_PIN_x</b>            | GPIO_PIN_x(x=0..15)                                    |
| <b>GPIO_PIN_ALL</b>          | all pins   |
| <b>Output parameter{out}</b> |  |

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

Example:

```
/* lock PA0 */
gpio_pin_lock(GPIOA, GPIO_PIN_0);
```

## 3.17. HAU

The HASH Acceleration Unit supports acceleration of SHA-1, SHA-224, SHA-256, MD5 algorithm and the HMAC (keyed-hash message authentication code) algorithm. The HAU registers are listed in chapter [3.17.1](#). the HAU firmware functions are introduced in chapter [3.17.2](#).

### 3.17.1. Descriptions of Peripheral registers

HAU registers are listed in the table shown as below:

**Table 3-482. HAU Registers**

| Registers | Descriptions                       |
|-----------|------------------------------------|
| HAU_CTL   | control register                   |
| HAU_DI    | data input register                |
| HAU_CFG   | configuration register             |
| HAU_DO0   | data output register 0             |
| HAU_DO1   | data output register 1             |
| HAU_DO2   | data output register 2             |
| HAU_DO3   | data output register 3             |
| HAU_DO4   | data output register 4             |
| HAU_DO5   | data output register 5             |
| HAU_DO6   | data output register 6             |
| HAU_DO7   | data output register 7             |
| HAU_INTEN | interrupt enable register          |
| HAU_STAT  | status and interrupt flag register |

### 3.17.2. Descriptions of Peripheral functions

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

**Table 3-483. HAU firmware function**

| Function name | Function description                     |
|---------------|--|
| hau_deinit    | reset the HAU peripheral                 |
| hau_init      | initialize the HAU peripheral parameters |

| Function name                      | Function description   |
|------------------------------------|--|
| hau_init_struct_para_init          | initialize the sturct hau_initpara   |
| hau_reset                          | reset the HAU processor core   |
| hau_last_word_validbits_num_config | configure the number of valid bits in last word of the message                                       |
| hau_data_write                     | write data to the IN FIFO  |
| hau_infifo_words_num_get           | return the number of words already written into the IN FIFO  |
| hau_digest_read                    | read the message digest result   |
| hau_digest_calculation_enable      | enable digest calculation  |
| hau_multiple_single_dma_config     | configure multiple or single DMA is used, and digest calculation at the end of a DMA transfer or not |
| hau_dma_enable                     | enable the HAU DMA interface   |
| hau_dma_disable                    | disable the HAU DMA interface  |
| hau_hash_sha_1                     | calculate digest using SHA1 in HASH mode   |
| hau_hmac_sha_1                     | calculate digest using SHA1 in HMAC mode   |
| hau_hash_sha_224                   | calculate digest using SHA224 in HASH mode   |
| hau_hmac_sha_224                   | calculate digest using SHA224 in HMAC mode   |
| hau_hash_sha_256                   | calculate digest using SHA256 in HASH mode   |
| hau_hmac_sha_256                   | calculate digest using SHA256 in HMAC mode   |
| hau_hash_md5                       | calculate digest using MD5 in HASH mode  |
| hau_hmac_md5                       | calculate digest using MD5 in HMAC mode  |
| hau_flag_get                       | get the HAU flag status  |
| hau_flag_clear                     | clear the HAU flag status  |
| hau_interrupt_enable               | enable the HAU interrupts  |
| hau_interrupt_disable              | disable the HAU interrupts   |
| hau_interrupt_flag_get             | get the HAU interrupt flag status  |
| hau_interrupt_flag_clear           | clear the HAU interrupt flag status  |

### Structure hau\_init\_parameter\_struct

Table 3-484. Structure hau\_init\_parameter\_struct

| Member name | Function description   |
|-------------|--|
| algo        | algorithm selection: HAU_ALGO_SHA1, HAU_ALGO_SHA224, HAU_ALGO_SHA256, HAU_ALGO_MD5           |
| mode        | HAU mode selection: HAU_MODE_HASH, HAU_MODE_HMAC   |
| datatype    | data type mode: HAU_SWAPPING_32BIT, HAU_SWAPPING_16BIT, HAU_SWAPPING_8BIT, HAU_SWAPPING_1BIT |
| keytype     | key length mode: HAU_KEY_SHORTER_64, HAU_KEY_LONGGER_64                                      |

### Structure hau\_digest\_parameter\_struct

**Table 3-485. Structure hau\_digest\_parameter\_struct**

| Member name | Function description      |
|-------------|---------------------------|
| out[8]      | message digest result 0-7 |

### hau\_deinit

The description of hau\_deinit is shown as below:

**Table 3-486. Function hau\_deinit**

|                       |                          |
|-----------------------|--------------------------|
| Function name         | hau_deinit               |
| Function prototype    | void hau_deinit(void);   |
| Function descriptions | reset the HAU peripheral |
| Precondition          | -                        |
| The called functions  | -                        |
| Input parameter{in}   |                          |
| -                     | -                        |
| Output parameter{out} |                          |
| -                     | -                        |
| Return value          |                          |
| -                     | -                        |

Example:

```
/* reset the HAU peripheral */
hau_deinit();
```

### hau\_init

The description of hau\_init is shown as below:

**Table 3-487. Function hau\_init**

|                       |  |
|-----------------------|--|
| Function name         | hau_init   |
| Function prototype    | void hau_init(hau_init_parameter_struct* initpara);  |
| Function descriptions | initialize the HAU peripheral parameters   |
| Precondition          | -  |
| The called functions  | -  |
| Input parameter{in}   |  |
| initpara              | HAU init parameter struct, refer to structure <a href="#">Table 3-484. Structure hau_init_parameter_struct</a> |
| Output parameter{out} |  |
| -                     | -  |
| Return value          |  |
| -                     | -  |

Example:

```

/* initialize the HAU peripheral parameters */

hau_init_parameter_struct initpara;

hau_initpara.algo = HAU_ALGO_MD5;

hau_initpara.mode = HAU_MODE_HASH;

hau_initpara.datatype = HAU_SWAPPING_8BIT;

hau_initpara.keytype = HAU_KEY_SHORTER_64;

hau_init(&initpara);
  
```

### **hau\_init\_struct\_para\_init**

The description of hau\_init\_parameter\_init is shown as below:

**Table 3-488. Function hau\_init\_parameter\_init**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | hau_init_struct_para_init  |
| <b>Function prototype</b>    | void hau_init_struct_para_init (hau_init_parameter_struct* initpara);  |
| <b>Function descriptions</b> | initialize the sturct hau_init_parameter_struct  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| initpara                     | HAU init parameter struct, refer to structure <a href="#">Table 3-484. Structure hau_init_parameter_struct</a> |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```

/* initialize the HAU peripheral parameters */

hau_init_parameter_struct initpara;

hau_init_struct_para_init(&initpara);
  
```

### **hau\_reset**

The description of hau\_reset is shown as below:

**Table 3-489. Function hau\_reset**

|                              |                              |
|------------------------------|------------------------------|
| <b>Function name</b>         | hau_reset                    |
| <b>Function prototype</b>    | void hau_reset(void);        |
| <b>Function descriptions</b> | reset the HAU processor core |

|                              |   |
|------------------------------|---|
| <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 HAU processor core */

hau_reset();
```

### **hau\_last\_word\_validbits\_num\_config**

The description of hau\_last\_word\_validbits\_num\_config is shown as below:

**Table 3-490. Function hau\_last\_word\_validbits\_num\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | hau_last_word_validbits_num_config                             |
| <b>Function prototype</b>    | void hau_last_word_validbits_num_config(uint32_t valid_num);   |
| <b>Function descriptions</b> | configure the number of valid bits in last word of the message |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>valid_num</b>             | number of valid bits in last word of the message(0x00 – 0x1F)  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure the number of valid bits in last word of the message */

hau_last_word_validbits_num_config(0x10);
```

### **hau\_data\_write**

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

**Table 3-491. Function hau\_data\_write**

|                              |                                     |
|------------------------------|-------------------------------------|
| <b>Function name</b>         | hau_data_write                      |
| <b>Function prototype</b>    | void hau_data_write(uint32_t data); |
| <b>Function descriptions</b> | write data to the IN FIFO           |
| <b>Precondition</b>          | -                                   |
| <b>The called functions</b>  | -                                   |

| Input parameter{in}   |                                 |
|-----------------------|---------------------------------|
| <b>data</b>           | data to write(0x0 – 0xFFFFFFFF) |
| Output parameter{out} |                                 |
| -                     | -                               |
| Return value          |                                 |
| -                     | -                               |

Example:

```
/* write data to the IN FIFO */
hau_data_write(0x10);
```

### **hau\_infifo\_words\_num\_get**

The description of hau\_infifo\_words\_num\_get is shown as below:

**Table 3-492. Function hau\_infifo\_words\_num\_get**

| <b>Function name</b>         | hau_infifo_words_num_get                                    |
|------------------------------|---|
| <b>Function prototype</b>    | uint32_t hau_infifo_words_num_get(void);                    |
| <b>Function descriptions</b> | return the number of words already written into the IN FIFO |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| Input parameter{in}          |   |
| -                            | -   |
| Output parameter{out}        |   |
| -                            | -   |
| Return value                 |   |
| <b>uint32_t</b>              | 0x0 – 0xFFFFFFFF  |

Example:

```
/* return the number of words already written into the IN FIFO */
uint32_t num;
num = hau_infifo_words_num_get();
```

### **hau\_digest\_read**

The description of hau\_digest\_read is shown as below:

**Table 3-493. Function hau\_digest\_read**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | hau_digest_read  |
| <b>Function prototype</b>    | void hau_digest_read(hau_digest_parameter_struct* digestpara); |
| <b>Function descriptions</b> | read the message digest result                                 |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |

| Input parameter{in}   |  |
|-----------------------|--|
| -                     | -  |
| Output parameter{out} |  |
| digestpara            | HAU digest parameter struct, refer to structure <a href="#">Table 3-485. Structure hau_digest_parameter_struct</a> |
| Return value          |  |
| -                     | -  |

Example:

```
/* read the message digest result */

hau_digest_parameter_struct digestpara;

hau_digest_read(&digestpara);
```

### **hau\_digest\_calculation\_enable**

The description of hau\_digest\_calculation\_enable is shown as below:

**Table 3-494. Function hau\_digest\_calculation\_enable**

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

Example:

```
/* enable digest calculation */

hau_digest_calculation_enable();
```

### **hau\_multiple\_single\_dma\_config**

The description of hau\_multiple\_single\_dma\_config is shown as below:

**Table 3-495. Function hau\_multiple\_single\_dma\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | hau_multiple_single_dma_config   |
| <b>Function prototype</b>    | void hau_multiple_single_dma_config(uint32_t multi_single);  |
| <b>Function descriptions</b> | configure single or multiple DMA is used, and digest calculation at the end of a DMA transfer or not |

|                               |   |
|-------------------------------|---|
| <b>Precondition</b>           | -   |
| <b>The called functions</b>   | -   |
| <b>Input parameter{in}</b>    |   |
| <b>multi_single</b>           | Multiple or single  |
| <b>SINGLE_DMA_AUTO_DIGEST</b> | message padding and message digest calculation at the end of a DMA transfer                       |
| <b>MULTIPLE_DMA_NO_DIGEST</b> | multiple DMA transfers needed and CALEN bit is not automatically set at the end of a DMA transfer |
| <b>Output parameter{out}</b>  |   |
| -                             | -   |
| <b>Return value</b>           |   |
| -                             | -   |

Example:

```
/* configure multiple or single DMA is used, and digest calculation at the end of a DMA transfer
or not */

hau_multiple_single_dma_config(SINGLE_DMA_AUTO_DIGEST);
```

### **hau\_dma\_enable**

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

**Table 3-496. Function hau\_dma\_enable**

|                              |                              |
|------------------------------|------------------------------|
| <b>Function name</b>         | hau_dma_enable               |
| <b>Function prototype</b>    | void hau_dma_enable(void);   |
| <b>Function descriptions</b> | enable the HAU DMA interface |
| <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 HAU DMA interface */

hau_dma_enable();
```

### **hau\_dma\_disable**

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

**Table 3-497. Function hau\_dma\_disable**

|                              |                               |
|------------------------------|-------------------------------|
| <b>Function name</b>         | hau_dma_disable               |
| <b>Function prototype</b>    | void hau_dma_disable(void);   |
| <b>Function descriptions</b> | disable the HAU DMA interface |
| <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 HAU DMA interface */

hau_dma_disable();
```

### **hau\_hash\_sha\_1**

The description of hau\_hash\_sha\_1 is shown as below:

**Table 3-498. Function hau\_hash\_sha\_1**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | hau_hash_sha_1  |
| <b>Function prototype</b>    | ErrStatus hau_hash_sha_1(uint8_t *input, uint32_t in_length, uint8_t output[20]); |
| <b>Function descriptions</b> | calculate digest using SHA1 in HASH mode  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>input</b>                 | pointer to the input buffer   |
| <b>Input parameter{in}</b>   |   |
| <b>in_length</b>             | length of the input buffer  |
| <b>Output parameter{out}</b> |   |
| <b>output</b>                | the result digest   |
| <b>Return value</b>          |   |
| <b>ErrStatus</b>             | SUCCESS or ERROR  |

Example:

```
/* calculate digest using SHA1 in HASH mode */

ErrStatus status;

const uint8_t input[16] = {0x01};

static uint8_t output[20];
```

```
status = hau_hash_sha_1((uint8_t *)input, 0x10, output);
```

### **hau\_hmac\_sha\_1**

The description of hau\_hmac\_sha\_1 is shown as below:

**Table 3-499. Function hau\_hmac\_sha\_1**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | hau_hmac_sha_1  |
| <b>Function prototype</b>    | ErrStatus hau_hmac_sha_1(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[20]); |
| <b>Function descriptions</b> | calculate digest using SHA1 in HMAC mode  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>key</b>                   | pointer to the key used for HMAC  |
| <b>Input parameter{in}</b>   |   |
| <b>keysize</b>               | length of the key used for HMAC   |
| <b>Input parameter{in}</b>   |   |
| <b>input</b>                 | pointer to the input buffer   |
| <b>Input parameter{in}</b>   |   |
| <b>in_length</b>             | length of the input buffer  |
| <b>Output parameter{out}</b> |   |
| <b>output</b>                | the result digest   |
| <b>Return value</b>          |   |
| <b>ErrStatus</b>             | SUCCESS or ERROR  |

Example:

```
/* calculate digest using SHA1 in HMAC mode */

ErrStatus status;

const uint8_t key[16] ={0x01};

const uint8_t input[16] ={0x01};

static uint8_t output[20];

status = hau_hmac_sha_1((uint8_t *)key, 0x10, (uint8_t *)input, 0x10, output);
```

### **hau\_hash\_sha\_224**

The description of hau\_hash\_sha\_224 is shown as below:

**Table 3-500. Function hau\_hash\_sha\_224**

|                           |   |
|---------------------------|---|
| <b>Function name</b>      | hau_hash_sha_224  |
| <b>Function prototype</b> | ErrStatus hau_hash_sha_224(uint8_t *input, uint32_t in_length, uint8_t output[28]); |

|                              |  |
|------------------------------|--|
| <b>Function descriptions</b> | calculate digest using SHA224 in HASH mode |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>input</b>                 | pointer to the input buffer                |
| <b>Input parameter{in}</b>   |  |
| <b>in_length</b>             | length of the input buffer                 |
| <b>Output parameter{out}</b> |  |
| <b>output</b>                | the result digest                          |
| <b>Return value</b>          |  |
| <b>ErrStatus</b>             | SUCCESS or ERROR                           |

Example:

```
/* calculate digest using SHA224 in HASH mode */

ErrStatus status;

const uint8_t input[16] ={0x01};

static uint8_t output[20];

status = hau_hash_sha_224((uint8_t *)input, 0x10, output);
```

### **hau\_hmac\_sha\_224**

The description of hau\_hmac\_sha\_224 is shown as below:

**Table 3-501. Function hau\_hmac\_sha\_224**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | hau_hmac_sha_224  |
| <b>Function prototype</b>    | ErrStatus hau_hmac_sha_224(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[28]); |
| <b>Function descriptions</b> | calculate digest using SHA224 in HMAC mode  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>key</b>                   | pointer to the key used for HMAC  |
| <b>Input parameter{in}</b>   |   |
| <b>keysize</b>               | length of the key used for HMAC   |
| <b>Input parameter{in}</b>   |   |
| <b>input</b>                 | pointer to the input buffer   |
| <b>Input parameter{in}</b>   |   |
| <b>in_length</b>             | length of the input buffer  |
| <b>Output parameter{out}</b> |   |
| <b>output</b>                | the result digest   |
| <b>Return value</b>          |   |
| <b>ErrStatus</b>             | SUCCESS or ERROR  |

Example:

```

/* calculate digest using SHA224 in HMAC mode */

ErrStatus status;

const uint8_t key[16] ={0x01};

const uint8_t input[16] ={0x01};

static uint8_t output[20];

status = hau_hmac_sha_224((uint8_t *)key, 0x10, (uint8_t *)input, 0x10, output);
  
```

### **hau\_hash\_sha\_256**

The description of hau\_hash\_sha\_256 is shown as below:

**Table 3-502. Function hau\_hash\_sha\_256**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | hau_hash_sha_256  |
| <b>Function prototype</b>    | ErrStatus hau_hash_sha_256(uint8_t *input, uint32_t in_length, uint8_t output[32]); |
| <b>Function descriptions</b> | calculate digest using SHA256 in HASH mode  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>input</b>                 | pointer to the input buffer   |
| <b>Input parameter{in}</b>   |   |
| <b>in_length</b>             | length of the input buffer  |
| <b>Output parameter{out}</b> |   |
| <b>output</b>                | the result digest   |
| <b>Return value</b>          |   |
| <b>ErrStatus</b>             | SUCCESS or ERROR  |

Example:

```

/* calculate digest using SHA256 in HASH mode */

ErrStatus status;

const uint8_t input[16] ={0x01};

static uint8_t output[20];

status = hau_hash_sha_256((uint8_t *)input, 0x10, output);
  
```

### **hau\_hmac\_sha\_256**

The description of hau\_hmac\_sha\_256 is shown as below:

**Table 3-503. Function hau\_hmac\_sha\_256**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | hau_hmac_sha_256  |
| <b>Function prototype</b>    | ErrStatus hau_hmac_sha_256(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[32]); |
| <b>Function descriptions</b> | calculate digest using SHA256 in HMAC mode  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>key</b>                   | pointer to the key used for HMAC  |
| <b>Input parameter{in}</b>   |   |
| <b>keysize</b>               | length of the key used for HMAC   |
| <b>Input parameter{in}</b>   |   |
| <b>input</b>                 | pointer to the input buffer   |
| <b>Input parameter{in}</b>   |   |
| <b>in_length</b>             | length of the input buffer  |
| <b>Output parameter{out}</b> |   |
| <b>output</b>                | the result digest   |
| <b>Return value</b>          |   |
| <b>ErrStatus</b>             | SUCCESS or ERROR  |

Example:

```
/* calculate digest using SHA256 in HMAC mode */

ErrStatus status;

const uint8_t key[16] ={0x01};

const uint8_t input[16] ={0x01};

static uint8_t output[20];

status = hau_hmac_sha_256((uint8_t *)key, 0x10, (uint8_t *)input, 0x10, output);
```

### **hau\_hash\_md5**

The description of hau\_hash\_md5 is shown as below:

**Table 3-504. Function hau\_hash\_md5**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | hau_hash_md5  |
| <b>Function prototype</b>    | ErrStatus hau_hash_md5(uint8_t *input, uint32_t in_length, uint8_t output[16]); |
| <b>Function descriptions</b> | calculate digest using MD5 in HASH mode   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>input</b>                 | pointer to the input buffer   |

| Input parameter{in}   |                            |
|-----------------------|----------------------------|
| in_length             | length of the input buffer |
| Output parameter{out} |                            |
| output                | the result digest          |
| Return value          |                            |
| ErrStatus             | SUCCESS or ERROR           |

Example:

```
/* calculate digest using MD5 in HASH mode */

ErrStatus status;

const uint8_t input[16] ={0x01};

static uint8_t output[20];

status = hau_hash_md5((uint8_t *)input, 0x10, output);
```

### **hau\_hmac\_md5**

The description of hau\_hmac\_md5 is shown as below:

**Table 3-505. Function hau\_hmac\_md5**

| <b>Function name</b>         | hau_hmac_md5  |
|------------------------------|---|
| <b>Function prototype</b>    | ErrStatus hau_hmac_md5(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[16]); |
| <b>Function descriptions</b> | calculate digest using MD5 in HMAC mode   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| Input parameter{in}          |   |
| key                          | pointer to the key used for HMAC  |
| Input parameter{in}          |   |
| keysize                      | length of the key used for HMAC   |
| Input parameter{in}          |   |
| input                        | pointer to the input buffer   |
| Input parameter{in}          |   |
| in_length                    | length of the input buffer  |
| Output parameter{out}        |   |
| output                       | the result digest   |
| Return value                 |   |
| ErrStatus                    | SUCCESS or ERROR  |

Example:

```
/* calculate digest using MD5 in HMAC mode */

ErrStatus status;
```

```

const uint8_t key[16] ={0x01};

const uint8_t input[16] ={0x01};

static uint8_t output[20];

status = hau_hmac_md5((uint8_t *)key, 0x10, (uint8_t *)input, 0x10, output);
  
```

### **hau\_flag\_get**

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

**Table 3-506. Function hau\_flag\_get**

|   |  |
|---|--|
| <b>Function name</b>                      | hau_flag_get   |
| <b>Function prototype</b>                 | FlagStatus hau_flag_get(uint32_t flag);              |
| <b>Function descriptions</b>              | get the HAU flag status                              |
| <b>Precondition</b>                       | -  |
| <b>The called functions</b>               | -  |
| <b>Input parameter{in}</b>                |  |
| <b>flag</b>                               | HAU flag status                                      |
| <i>HAU_FLAG_DATA_INP<br/>UT</i>           | there is enough space (16 bytes) in the input FIFO   |
| <i>HAU_FLAG_CALCULA<br/>TION_COMPLETE</i> | digest calculation is completed                      |
| <i>HAU_FLAG_DMA</i>                       | DMA is enabled (DMAE =1) or a transfer is processing |
| <i>HAU_FLAG_BUSY</i>                      | data block is in process                             |
| <i>HAU_FLAG_INFIFO_N<br/>O_EMPTY</i>      | the input FIFO is not empty                          |
| <b>Output parameter{out}</b>              |  |
| -   | -  |
| <b>Return value</b>                       |  |
| <b>FlagStatus</b>                         | SET or RESET   |

Example:

```

/* get the HAU flag status */

FlagStatus status;

status = hau_flag_get(HAU_FLAG_DMA);
  
```

### **hau\_flag\_clear**

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

**Table 3-507. Function hau\_flag\_clear**

|                           |                                     |
|---------------------------|-------------------------------------|
| <b>Function name</b>      | hau_flag_clear                      |
| <b>Function prototype</b> | void hau_flag_clear(uint32_t flag); |

|                                      |  |
|--------------------------------------|--|
| <b>Function descriptions</b>         | clear the HAU flag status                          |
| <b>Precondition</b>                  | -  |
| <b>The called functions</b>          | -  |
| <b>Input parameter{in}</b>           |  |
| <b>flag</b>                          | HAU flag status                                    |
| <i>HAU_FLAG_DATA_INPUT</i>           | there is enough space (16 bytes) in the input FIFO |
| <i>UT</i>                            |  |
| <i>HAU_FLAG_CALCULATION_COMPLETE</i> | digest calculation is completed                    |
| <b>Output parameter{out}</b>         |  |
| -                                    | -  |
| <b>Return value</b>                  |  |
| -                                    | -  |

Example:

```
/* clear the HAU flag status */

hau_flag_clear(HAU_FLAG_DATA_INPUT);
```

### **hau\_interrupt\_enable**

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

**Table 3-508. Function hau\_interrupt\_enable**

|                                     |  |
|-------------------------------------|--|
| <b>Function name</b>                | hau_interrupt_enable                           |
| <b>Function prototype</b>           | void hau_interrupt_enable(uint32_t interrupt); |
| <b>Function descriptions</b>        | enable the HAU interrupts                      |
| <b>Precondition</b>                 | -  |
| <b>The called functions</b>         | -  |
| <b>Input parameter{in}</b>          |  |
| <b>flag</b>                         | HAU flag status                                |
| <i>HAU_INT_DATA_INPUT</i>           | a new block can be entered into the IN buffer  |
| <i>T</i>                            |  |
| <i>HAU_INT_CALCULATION_COMPLETE</i> | calculation complete                           |
| <b>Output parameter{out}</b>        |  |
| -                                   | -  |
| <b>Return value</b>                 |  |
| -                                   | -  |

Example:

```
/* enable hau interrupt */

hau_interrupt_enable(HAU_INT_DATA_INPUT);
```

### **hau\_interrupt\_disable**

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

**Table 3-509. Function hau\_interrupt\_disable**

|                                     |   |
|-------------------------------------|---|
| <b>Function name</b>                | hau_interrupt_disable                           |
| <b>Function prototype</b>           | void hau_interrupt_disable(uint32_t interrupt); |
| <b>Function descriptions</b>        | disable the HAU interrupts                      |
| <b>Precondition</b>                 | -   |
| <b>The called functions</b>         | -   |
| <b>Input parameter{in}</b>          |   |
| <b>flag</b>                         | HAU flag status                                 |
| <i>HAU_INT_DATA_INPUT_T</i>         | a new block can be entered into the IN buffer   |
| <i>HAU_INT_CALCULATION_COMPLETE</i> | calculation complete                            |
| <b>Output parameter{out}</b>        |   |
| -                                   | -   |
| <b>Return value</b>                 |   |
| -                                   | -   |

Example:

```
/* disable hau interrupt */
hau_interrupt_disable(HAU_INT_DATA_INPUT);
```

### **hau\_interrupt\_flag\_get**

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

**Table 3-510. Function hau\_interrupt\_flag\_get**

|  |   |
|--|---|
| <b>Function name</b>                     | hau_interrupt_flag_get                                |
| <b>Function prototype</b>                | FlagStatus hau_interrupt_flag_get(uint32_t int_flag); |
| <b>Function descriptions</b>             | get the HAU interrupt flag status                     |
| <b>Precondition</b>                      | -   |
| <b>The called functions</b>              | -   |
| <b>Input parameter{in}</b>               |   |
| <b>int_flag</b>                          | HAU interrupt flag status                             |
| <i>HAU_INT_FLAG_DATA_INPUT</i>           | there is enough space (16 bytes) in the input FIFO    |
| <i>HAU_INT_FLAG_CALCULATION_COMPLETE</i> | digest calculation is completed                       |
| <b>Output parameter{out}</b>             |   |
| -  | -   |
| <b>Return value</b>                      |   |

|            |              |
|------------|--------------|
| FlagStatus | SET or RESET |
|------------|--------------|

Example:

```

/* get the HAU interrupt flag status */

FlagStatus status;

status = hau_interrupt_flag_get(HAU_INT_FLAG_DATA_INPUT);
  
```

### **hau\_interrupt\_flag\_clear**

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

**Table 3-511. Function hau\_interrupt\_flag\_clear**

|  |  |  |
|--|--|--|
| <b>Function name</b>                     | hau_interrupt_flag_clear                           |  |
| <b>Function prototype</b>                | void hau_interrupt_flag_clear(uint32_t int_flag);  |  |
| <b>Function descriptions</b>             | clear the HAU interrupt flag status                |  |
| <b>Precondition</b>                      | -  |  |
| <b>The called functions</b>              | -  |  |
| <b>Input parameter{in}</b>               |  |  |
| <i>int_flag</i>                          | HAU interrupt flag status                          |  |
| <i>HAU_INT_FLAG_DATA_INPUT</i>           | there is enough space (16 bytes) in the input FIFO |  |
| <i>HAU_INT_FLAG_CALCULATION_COMPLETE</i> | digest calculation is completed                    |  |
| <b>Output parameter{out}</b>             |  |  |
| -  | -  |  |
| <b>Return value</b>                      |  |  |
| -  | -  |  |

Example:

```

/* clear the HAU interrupt flag status */

hau_interrupt_flag_clear(HAU_INT_FLAG_DATA_INPUT);
  
```

## **3.18. 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 [3.18.1](#), the I2C firmware functions are introduced in chapter [3.18.2](#).

### **3.18.1. Descriptions of Peripheral registers**

I2C registers are listed in the table shown as below:

**Table 3-512. 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         |

### 3.18.2. Descriptions of Peripheral functions

I2C firmware functions are listed in the table shown as below:

**Table 3-513. I2C firmware function**

| Function name                      | Function description  |
|------------------------------------|---|
| i2c_deinit                         | reset I2C   |
| i2c_clock_config                   | configure I2C clock   |
| i2c_mode_addr_config               | configure I2C address   |
| i2c_smbus_type_config              | SMBus type selection  |
| i2c_ack_config                     | whether or not to send an ACK                                   |
| i2c_ackpos_config                  | configure I2C POAP position                                     |
| i2c_master_addressing              | master send slave address                                       |
| i2c_dualaddr_enable                | enable dual-address mode  |
| i2c_dualaddr_disable               | disable dual-address mode                                       |
| i2c_enable                         | enable I2C  |
| i2c_disable                        | disable I2C   |
| i2c_start_on_bus                   | generate a START condition on I2C bus                           |
| i2c_stop_on_bus                    | generate a STOP condition on I2C bus                            |
| i2c_data_transmit                  | I2C transmit data function                                      |
| i2c_data_receive                   | I2C receive data function                                       |
| i2c_dma_enable                     | I2C DMA mode enable   |
| i2c_dma_last_transfer_config       | configure whether next DMA EOT is DMA last transfer or not      |
| i2c_stretch_scl_low_config         | whether to stretch SCL low when data is not ready in slave mode |
| i2c_slave_response_to_gcall_config | whether or not to response to a general call                    |
| i2c_software_reset_config          | software reset I2C  |
| i2c_pec_enable                     | I2C PEC calculation on or off                                   |
| i2c_pec_transfer_enable            | I2C whether to transfer PEC value                               |
| i2c_pec_value_get                  | packet error checking value                                     |
| i2c_smbus_issue_alert              | I2C issue alert through SMBA pin                                |

| Function name            | Function description             |
|--------------------------|----------------------------------|
| i2c_smbus_arp_enable     | I2C ARP protocol in SMBus switch |
| i2c_flag_get             | check I2C flag is set or not     |
| i2c_flag_clear           | clear I2C flag                   |
| i2c_interrupt_enable     | enable I2C interrupt             |
| i2c_interrupt_disable    | disable I2C interrupt            |
| i2c_interrupt_flag_get   | check I2C interrupt flag         |
| i2c_interrupt_flag_clear | clear I2C interrupt flag         |

### i2c\_deinit

The description of i2c\_deinit is shown as below:

**Table 3-514. Function i2c\_deinit**

|                       |  |  |
|-----------------------|--|--|
| Function name         | i2c_deinit   |  |
| Function prototype    | void i2c_deinit(uint32_t i2c_periph);              |  |
| Function descriptions | reset I2C  |  |
| Precondition          | -  |  |
| The called functions  | rcu_periph_reset_enable / rcu_periph_reset_disable |  |
| Input parameter{in}   |  |  |
| i2c_periph            | I2C peripheral                                     |  |
| I2Cx                  | (x=0,1,2)  |  |
| Output parameter{out} |  |  |
| -                     | -  |  |
| Return value          |  |  |
| -                     | -  |  |

Example:

```
/* reset I2C0 */
i2c_deinit(I2C0);
```

### i2c\_clock\_config

The description of i2c\_clock\_config is shown as below:

**Table 3-515. Function i2c\_clock\_config**

|                       |  |  |
|-----------------------|--|--|
| Function name         | i2c_clock_config   |  |
| Function prototype    | void i2c_clock_config(uint32_t i2c_periph, uint32_t clkspeed, uint32_t dutycyc); |  |
| Function descriptions | I2C clock configure  |  |
| Precondition          | -  |  |
| The called functions  | rcu_clock_freq_get   |  |
| Input parameter{in}   |  |  |
| i2c_periph            | I2C peripheral   |  |

|                              |   |
|------------------------------|---|
| <i>I2Cx</i>                  | (x=0,1,2)                                 |
| <b>Input parameter{in}</b>   |   |
| <i>clkspeed</i>              | i2c clock speed                           |
| <b>Input parameter{in}</b>   |   |
| <i>dutycyc</i>               | duty cycle in fast mode or fast mode plus |
| <i>I2C_DTCY_2</i>            | $T_{low}/T_{high}=2$                      |
| <i>I2C_DTCY_16_9</i>         | $T_{low}/T_{high}=16/9$                   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* configure I2C0 clock speed as 100KHz */
i2c_clock_config(I2C0, 100000, I2C_DTCY_2);
```

### **i2c\_mode\_addr\_config**

The description of *i2c\_mode\_addr\_config* is shown as below:

**Table 3-516. Function i2c\_mode\_addr\_config**

|                                  |   |
|----------------------------------|---|
| <b>Function name</b>             | i2c_mode_addr_config  |
| <b>Function prototype</b>        | void i2c_mode_addr_config(uint32_t i2c_periph, uint32_t mode, uint32_t addformat, uint32_t addr); |
| <b>Function descriptions</b>     | configure I2C address   |
| <b>Precondition</b>              | -   |
| <b>The called functions</b>      | -   |
| <b>Input parameter{in}</b>       |   |
| <i>i2c_periph</i>                | I2C peripheral  |
| <i>I2Cx</i>                      | (x=0,1,2)   |
| <b>Input parameter{in}</b>       |   |
| <i>mode</i>                      | I2C mode select   |
| <i>I2C_I2CMODE_ENABL<br/>E</i>   | I2C mode  |
| <i>I2C_SMBUSMODE_EN<br/>ABLE</i> | SMBus mode  |
| <b>Input parameter{in}</b>       |   |
| <i>addformat</i>                 | 7bits or 10bits   |
| <i>I2C_ADDFORMAT_7BITS</i>       | 7bits   |
| <i>I2C_ADDFORMAT_10BITS</i>      | 10bits  |
| <b>Input parameter{in}</b>       |   |

|                              |             |
|------------------------------|-------------|
| <b>addr</b>                  | I2C address |
| <b>Output parameter{out}</b> |             |
| -                            | -           |
| <b>Return value</b>          |             |
| -                            | -           |

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 3-517. Function i2c\_smbus\_type\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | i2c_smbus_type_config   |
| <b>Function prototype</b>    | void i2c_smbus_type_config(uint32_t i2c_periph, uint32_t type); |
| <b>Function descriptions</b> | SMBus type selection  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <i>i2c_periph</i>            | I2C peripheral  |
| <i>I2Cx</i>                  | (x=0,1,2)   |
| <b>Input parameter{in}</b>   |   |
| <b>type</b>                  | Device or host  |
| <i>I2C_SMBUS_DEVICE</i>      | device  |
| <i>I2C_SMBUS_HOST</i>        | host  |
| <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 3-518. 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>   |                      |
| <i>i2c_periph</i>            | I2C peripheral       |
| <i>I2Cx</i>                  | (x=0,1,2)            |
| <b>Input parameter{in}</b>   |                      |
| <b>ack</b>                   | I2C peripheral       |
| <i>I2C_ACK_ENABLE</i>        | ACK will be sent     |
| <i>I2C_ACK_DISABLE</i>       | ACK will not be sent |
| <b>Output parameter{out}</b> |                      |
| -                            | -                    |
| <b>Return value</b>          |                      |
| -                            | -                    |

Example:

```
/* I2C0 will send ACK */
i2c_ack_config(I2C0, I2C_ACK_ENABLE);
```

### **i2c\_ackpos\_config**

The description of i2c\_ackpos\_config is shown as below:

**Table 3-519. 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>   |  |
| <i>i2c_periph</i>            | I2C peripheral   |
| <i>I2Cx</i>                  | (x=0,1,2)  |
| <b>Input parameter{in}</b>   |  |
| <b>pos</b>                   | ACK position   |
| <i>I2C_ACKPOS_CURRENT</i>    | whether to send ACK or not for the current                 |
| <i>I2C_ACKPOS_NEXT</i>       | whether to send ACK or not 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 3-520. Function i2c\_master\_addressing**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | i2c_master_addressing   |
| <b>Function prototype</b>    | void i2c_master_addressing(uint32_t i2c_periph, uint32_t addr, uint32_t trandirection); |
| <b>Function descriptions</b> | master sends slave address  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>i2c_periph</b>            | I2C peripheral  |
| <i>I2Cx</i>                  | (x=0,1,2)   |
| <b>Input parameter{in}</b>   |   |
| <b>addr</b>                  | slave address   |
| <b>Input parameter{in}</b>   |   |
| <b>trandirection</b>         | transmitter or receiver   |
| <i>I2C_TRANSMITTER</i>       | transmitter   |
| <i>I2C_RECEIVER</i>          | receiver  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* send slave address to I2C bus and I2C0 act as receiver */
i2c_master_addressing(I2C0, 0x82, I2C_RECEIVER);
```

### **i2c\_dualaddr\_enable**

The description of i2c\_dualaddr\_enable is shown as below:

**Table 3-521. Function i2c\_dualaddr\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | i2c_dualaddr_enable   |
| <b>Function prototype</b>    | void i2c_dualaddr_enable(uint32_t i2c_periph, uint32_t addr); |
| <b>Function descriptions</b> | enable dual-address mode                                      |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>i2c_periph</b>            | I2C peripheral  |
| <i>I2Cx</i>                  | (x=0,1,2)   |

| Input parameter{in}   |   |
|-----------------------|---|
| <b>addr</b>           | the second address in dual-address mode |
| Output parameter{out} |   |
| -                     | -                                       |
| Return value          |   |
| -                     | -                                       |

Example:

```
/* configure I2C second slave address */

i2c_dualaddr_enable(I2C0, I2C_DUADEN_ENABLE);
```

### **i2c\_dualaddr\_disable**

The description of i2c\_dualaddr\_disable is shown as below:

**Table 3-522. 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>  | -   |
| Input parameter{in}          |   |
| <b>i2c_periph</b>            | I2C peripheral                                  |
| <b>I2Cx</b>                  | (x=0,1,2)                                       |
| Output parameter{out}        |   |
| -                            | -   |
| Return value                 |   |
| -                            | -   |

Example:

```
/* disable I2C0 dual-address */

i2c_dualaddr_disable(I2C0);
```

### **i2c\_enable**

The description of i2c\_enable is shown as below:

**Table 3-523. 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>  | -                                     |
| Input parameter{in}          |                                       |

|                              |                |
|------------------------------|----------------|
| <b>i2c_periph</b>            | I2C peripheral |
| <i>I2Cx</i>                  | (x=0,1,2)      |
| <b>Output parameter{out}</b> |                |
| -                            | -              |
| <b>Return value</b>          |                |
| -                            | -              |

Example:

```
/* enable I2C0 */

i2c_enable(I2C0);
```

### **i2c\_disable**

The description of **i2c\_disable** is shown as below:

**Table 3-524. 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>   |  |
| <i>i2c_periph</i>            | I2C peripheral                         |
| <i>I2Cx</i>                  | (x=0,1,2)                              |
| <b>Output parameter{out}</b> |  |
| -                            | -                                      |
| <b>Return value</b>          |  |
| -                            | -                                      |

Example:

```
/* disable I2C0 */

i2c_disable(I2C0);
```

### **i2c\_start\_on\_bus**

The description of **i2c\_start\_on\_bus** is shown as below:

**Table 3-525. Function i2c\_start\_on\_bus**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | i2c_start_on_bus                            |
| <b>Function prototype</b>    | void i2c_start_on_bus(uint32_t i2c_periph); |
| <b>Function descriptions</b> | generate a START condition on I2C bus       |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |

|                              |                |
|------------------------------|----------------|
| <b>i2c_periph</b>            | I2C peripheral |
| <i>I2Cx</i>                  | (x=0,1,2)      |
| <b>Output parameter{out}</b> |                |
| -                            | -              |
| <b>Return value</b>          |                |
| -                            | -              |

Example:

```
/* I2C0 send a start condition to I2C bus */

i2c_start_on_bus(I2C0);
```

### **i2c\_stop\_on\_bus**

The description of i2c\_stop\_on\_bus is shown as below:

**Table 3-526. 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>   |  |
| <i>i2c_periph</i>            | I2C peripheral                             |
| <i>I2Cx</i>                  | (x=0,1,2)                                  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* I2C0 generate a STOP condition to I2C bus */

i2c_stop_on_bus(I2C0);
```

### **i2c\_data\_transmit**

The description of i2c\_data\_transmit is shown as below:

**Table 3-527. Function i2c\_data\_transmit**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_data_transmit  |
| <b>Function prototype</b>    | void i2c_data_transmit(uint32_t i2c_periph, uint8_t data); |
| <b>Function descriptions</b> | I2C transmit data function                                 |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |

|                              |                |
|------------------------------|----------------|
| <b>i2c_periph</b>            | I2C peripheral |
| <i>I2Cx</i>                  | (x=0,1,2)      |
| <b>Input parameter{in}</b>   |                |
| <b>data</b>                  | transmit data  |
| <b>Output parameter{out}</b> |                |
| -                            | -              |
| <b>Return value</b>          |                |
| -                            | -              |

Example:

```
/* I2C0 transmit data */

i2c_data_transmit(I2C0);
```

### i2c\_data\_receive

The description of i2c\_data\_receive is shown as below:

**Table 3-528. Function i2c\_data\_receive**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_data_receive                               |
| <b>Function prototype</b>    | uint8_t i2c_data_receive(uint32_t i2c_periph); |
| <b>Function descriptions</b> | I2C receive data function                      |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>i2c_periph</b>            | I2C peripheral                                 |
| <i>I2Cx</i>                  | (x=0,1,2)                                      |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>uint8_t</b>               | 0x00..0xFF                                     |

Example:

```
/* I2C0 receive data */

uint8_t i2c_receiver;

i2c_receiver = i2c_data_receive(I2C0);
```

### i2c\_dma\_enable

The description of i2c\_dma\_enable is shown as below:

**Table 3-529. Function i2c\_dma\_enable**

|                           |  |
|---------------------------|--|
| <b>Function name</b>      | i2c_dma_enable   |
| <b>Function prototype</b> | void i2c_dma_enable(uint32_t i2c_periph, uint32_t dmastate); |

|                              |                     |
|------------------------------|---------------------|
| <b>Function descriptions</b> | enable I2C DMA mode |
| <b>Precondition</b>          | -                   |
| <b>The called functions</b>  | -                   |
| <b>Input parameter{in}</b>   |                     |
| <i>i2c_periph</i>            | I2C peripheral      |
| <i>I2Cx</i>                  | (x=0,1,2)           |
| <b>Input parameter{in}</b>   |                     |
| <b>dmastate</b>              | On or off           |
| <i>I2C_DMA_ON</i>            | DMA mode enable     |
| <i>I2C_DMA_OFF</i>           | DMA mode disable    |
| <b>Output parameter{out}</b> |                     |
| -                            | -                   |
| <b>Return value</b>          |                     |
| -                            | -                   |

Example:

```
/* I2C0 DMA mode enable */
i2c_dma_enable(I2C0, I2C_DMA_ON);
```

### **i2c\_dma\_last\_transfer\_config**

The description of *i2c\_dma\_last\_transfer\_config* is shown as below:

**Table 3-530. Function i2c\_dma\_last\_transfer\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | i2c_dma_last_transfer_config  |
| <b>Function prototype</b>    | void i2c_dma_last_transfer_config(uint32_t i2c_periph, uint32_t dmalast); |
| <b>Function descriptions</b> | configure whether next DMA EOT is DMA last transfer or not                |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <i>i2c_periph</i>            | I2C peripheral  |
| <i>I2Cx</i>                  | (x=0,1,2)   |
| <b>Input parameter{in}</b>   |   |
| <b>dmalast</b>               | next DMA EOT is the last transfer or not                                  |
| <i>I2C_DMALST_ON</i>         | next DMA EOT is the last transfer   |
| <i>I2C_DMALST_OFF</i>        | next DMA EOT is not the last transfer                                     |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

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 3-531. Function i2c\_stretch\_scl\_low\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | i2c_stretch_scl_low_config  |
| <b>Function prototype</b>    | void i2c_stretch_scl_low_config(uint32_t i2c_periph, uint32_t stretchpara); |
| <b>Function descriptions</b> | whether to stretch SCL low when data is not ready in slave mode             |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| i2c_periph                   | I2C peripheral  |
| I2Cx                         | (x=0,1,2)   |
| <b>Input parameter{in}</b>   |   |
| stretchpara                  | SCL stretching enable or disable  |
| I2C_SCLSTRETCH_ENABLE        | SCL stretching is enabled   |
| I2C_SCLSTRETCH_DISABLE       | SCL stretching is disabled  |
| <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 3-532. 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>   |   |
| i2c_periph                   | I2C peripheral  |
| I2Cx                         | (x=0,1,2)   |

| Input parameter{in}     |   |
|-------------------------|---|
| <i>gcallpara</i>        | 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 |
| Output parameter{out}   |   |
| -                       | -   |
| Return value            |   |
| -                       | -   |

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 3-533. Function i2c\_software\_reset\_config**

| <b>Function name</b>         | i2c_software_reset_config   |
|------------------------------|---|
| <b>Function prototype</b>    | void i2c_software_reset_config(uint32_t i2c_periph, uint32_t sreset); |
| <b>Function descriptions</b> | software reset I2C  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| Input parameter{in}          |   |
| <i>i2c_periph</i>            | I2C peripheral  |
| <i>I2Cx</i>                  | (x=0,1,2)   |
| Input parameter{in}          |   |
| <i>sreset</i>                | under reset or not  |
| <i>I2C_SRESET_SET</i>        | I2C is under reset  |
| <i>I2C_SRESET_RESET</i>      | I2C is not under reset  |
| Output parameter{out}        |   |
| -                            | -   |
| Return value                 |   |
| -                            | -   |

Example:

```
/* software reset I2C0 */
i2c_software_reset_config(I2C0, I2C_SRESET_SET);
```

### i2c\_pec\_enable

The description of i2c\_pec\_enable is shown as below:

**Table 3-534. Function i2c\_pec\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_pec_enable   |
| <b>Function prototype</b>    | void i2c_pec_enable(uint32_t i2c_periph, uint32_t pecstate); |
| <b>Function descriptions</b> | I2C PEC calculation on or off                                |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| i2c_periph                   | I2C peripheral   |
| I2Cx                         | (x=0,1,2)  |
| <b>Input parameter{in}</b>   |  |
| pecpara                      | On or off  |
| I2C_PEC_ENABLE               | PEC calculation on   |
| I2C_PEC_DISABLE              | PEC calculation off  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* Enable I2C PEC calculation */

i2c_pec_enable(I2C0, I2C_PEC_ENABLE);
```

### i2c\_pec\_transfer\_enable

The description of i2c\_pec\_transfer\_enable is shown as below:

**Table 3-535. Function i2c\_pec\_transfer\_enable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_pec_transfer_enable  |
| <b>Function prototype</b>    | void i2c_pec_transfer_enable(uint32_t i2c_periph, uint32_t pecpara); |
| <b>Function descriptions</b> | I2C whether to transfer PEC value                                    |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| i2c_periph                   | I2C peripheral   |
| I2Cx                         | (x=0,1,2)  |
| <b>Input parameter{in}</b>   |  |
| pecpara                      | transfer PEC or not  |
| I2C_PECTRANS_ENA<br>BLE      | transfer PEC   |
| I2C_PECTRANS_DISA<br>BLE     | not transfer PEC   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |

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

Example:

```
/* I2C0 transfer PEC */

i2c_pec_transfer_enable(I2C0, I2C_PECTRANS_ENABLE);
```

### **i2c\_pec\_value\_get**

The description of i2c\_pec\_value\_get is shown as below:

**Table 3-536. 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>   |   |
| i2c_periph                   | I2C peripheral                                  |
| I2Cx                         | (x=0,1,2)                                       |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| uint8_t                      | 0..255  |

Example:

```
/* I2C0 get packet error checking value */

uint8_t pec_value;

pec_value = i2c_pec_value_get(I2C0);
```

### **i2c\_smbus\_issue\_alert**

The description of i2c\_smbus\_issue\_alert is shown as below:

**Table 3-537. Function i2c\_smbus\_issue\_alert**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_smbus_issue_alert  |
| <b>Function prototype</b>    | void i2c_smbus_issue_alert(uint32_t i2c_periph, uint32_t smbuspara); |
| <b>Function descriptions</b> | I2C issue alert through SMBA pin                                     |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| i2c_periph                   | I2C peripheral   |
| I2Cx                         | (x=0,1,2)  |

| Input parameter{in}              |                                     |
|----------------------------------|-------------------------------------|
| <b>smbuspara</b>                 | issue alert through SMBA pin or not |
| <i>I2C_SALTSEND_ENAB<br/>LE</i>  | issue alert through SMBA pin        |
| <i>I2C_SALTSEND_DISA<br/>BLE</i> | not issue alert through SMBA pin    |
| Output parameter{out}            |                                     |
| -                                | -                                   |
| Return value                     |                                     |
| -                                | -                                   |

Example:

```
/* I2C0 issue alert through SMBA pin enable*/
i2c_smbus_issue_alert(I2C0, I2C_SALTSEND_ENABLE);
```

### i2c\_smbus\_arp\_enable

The description of i2c\_smbus\_arp\_enable is shown as below:

**Table 3-538. Function i2c\_smbus\_arp\_enable**

| <b>Function name</b>         | i2c_smbus_arp_enable   |
|------------------------------|--|
| <b>Function prototype</b>    | void i2c_smbus_arp_enable(uint32_t i2c_periph, uint32_t arpstate); |
| <b>Function descriptions</b> | enable or disable I2C ARP protocol in SMBus switch                 |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| Input parameter{in}          |  |
| <b>i2c_periph</b>            | I2C peripheral   |
| <i>I2Cx</i>                  | (x=0,1,2)  |
| Input parameter{in}          |  |
| <b>arpstate</b>              | ARP protocol in SMBus switch                                       |
| <i>I2C_ARP_ENABLE</i>        | enable ARP   |
| <i>I2C_ARP_DISABLE</i>       | disable ARP  |
| Output parameter{out}        |  |
| -                            | -  |
| Return value                 |  |
| -                            | -  |

Example:

```
/* enable I2C0 ARP protocol in SMBus switch */
i2c_smbus_arp_enable(I2C0, I2C_ARP_ENABLE);
```

## i2c\_flag\_get

The description of i2c\_flag\_get is shown as below:

**Table 3-539. Function i2c\_flag\_get**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_flag_get   |
| <b>Function prototype</b>    | FlagStatus i2c_flag_get(uint32_t i2c_periph,uint32_t flag );                     |
| <b>Function descriptions</b> | check I2C flag is set or not   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| i2c_periph                   | I2C peripheral   |
| I2Cx                         | (x=0,1,2)  |
| <b>Input parameter{in}</b>   |  |
| flag                         | specify get which flag   |
| I2C_FLAG_SBSEND              | start condition send out   |
| I2C_FLAG_ADDSEND             | address is sent in master mode or received and matches in slave mode             |
| I2C_FLAG_BTC                 | byte transmission finishes   |
| I2C_FLAG_ADD10SEN<br>D       | header of 10-bit address is sent in master mode                                  |
| I2C_FLAG_STPDET              | stop condition detected in slave mode  |
| I2C_FLAG_RBNE                | I2C_DATA is not empty during receiving   |
| I2C_FLAG_TBE                 | I2C_DATA is empty during transmitting  |
| I2C_FLAG_BERR                | a bus error occurs indication a unexpected start or stop condition on I2C bus    |
| I2C_FLAG_LOSTARB             | arbitration lost in master mode  |
| I2C_FLAG_AERR                | acknowledge error  |
| I2C_FLAG_OVERRR              | overrun or underrun 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_TRS                 | 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 |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| FlagStatus                   | SET / RESET  |

Example:

```

/* check whether start condition send out */

FlagStatus flag_state = RESET;

flag_state = i2c_flag_get(I2C0, I2C_FLAG_SBSEND);
  
```

### i2c\_flag\_clear

The description of i2c\_flag\_clear is shown as below:

**Table 3-540. Function i2c\_flag\_clear**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_flag_clear   |
| <b>Function prototype</b>    | void i2c_flag_clear(uint32_t i2c_periph, uint32_t flag); |
| <b>Function descriptions</b> | clear I2C flag   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| i2c_periph                   | I2C peripheral   |
| I2Cx                         | (x=0,1,2)  |
| <b>Input parameter{in}</b>   |  |
| flag                         | flag type  |
| I2C_FLAG_SMBALT              | SMBus Alert status                                       |
| I2C_FLAG_SMBTO               | timeout signal in SMBus mode                             |
| I2C_FLAG_PECERR              | PEC error when receiving data                            |
| I2C_FLAG_OUERR               | over-run or under-run situation occurs in slave mode     |
| I2C_FLAG_AERR                | acknowledge error  |
| I2C_FLAG_LOSTARB             | arbitration lost in master mode                          |
| I2C_FLAG_BERR                | a bus error  |
| I2C_FLAG_ADDSEND             | cleared by reading I2C_STAT0 and reading I2C_STAT1       |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```

/* clear a bus error flag*/

i2c_flag_clear(I2C0, I2C_FLAG_BERR);
  
```

### i2c\_interrupt\_enable

The description of i2c\_interrupt\_enable is shown as below:

**Table 3-541. Function i2c\_interrupt\_enable**

|                      |                      |
|----------------------|----------------------|
| <b>Function name</b> | i2c_interrupt_enable |
|----------------------|----------------------|

|                              |   |
|------------------------------|---|
| <b>Function prototype</b>    | void i2c_interrupt_enable(uint32_t i2c_periph, i2c_interrupt_enum interrupt); |
| <b>Function descriptions</b> | enable I2C interrupt  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| i2c_periph                   | I2C peripheral  |
| I2Cx                         | (x=0,1,2)   |
| <b>Input parameter{in}</b>   |   |
| interrupt                    | interrupt type  |
| I2C_INT_ERR                  | error interrupt enable  |
| I2C_INT_EV                   | event interrupt enable  |
| I2C_INT_BUF                  | buffer interrupt enable   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable I2C0 error interrupt */
i2c_interrupt_enable(I2C0, I2C_INT_EV);
```

### i2c\_interrupt\_disable

The description of i2c\_interrupt\_disable is shown as below:

**Table 3-542. Function i2c\_interrupt\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | i2c_interrupt_disable  |
| <b>Function prototype</b>    | void i2c_interrupt_disable(uint32_t i2c_periph, i2c_interrupt_enum interrupt); |
| <b>Function descriptions</b> | disable I2C interrupt  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| i2c_periph                   | I2C peripheral   |
| I2Cx                         | (x=0,1,2)  |
| <b>Input parameter{in}</b>   |  |
| interrupt                    | interrupt type   |
| I2C_INT_ERR                  | error interrupt disable  |
| I2C_INT_EV                   | event interrupt disable  |
| I2C_INT_BUF                  | buffer interrupt disable   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |

|   |   |
|---|---|
| - | - |
|---|---|

Example:

```
/* disable I2C0 error interrupt */

i2c_interrupt_disable(I2C0, I2C_INT_EV);
```

### i2c\_interrupt\_flag\_get

The description of i2c\_interrupt\_flag\_get is shown as below:

**Table 3-543. Function i2c\_interrupt\_flag\_get**

|                                |  |
|--------------------------------|--|
| <b>Function name</b>           | i2c_interrupt_flag_get   |
| <b>Function prototype</b>      | FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph,<br>i2c_interrupt_flag_enum int_flag); |
| <b>Function descriptions</b>   | check I2C interrupt flag   |
| <b>Precondition</b>            | -  |
| <b>The called functions</b>    | -  |
| <b>Input parameter{in}</b>     |  |
| <i>i2c_periph</i>              | I2C peripheral   |
| <i>I2Cx</i>                    | (x=0,1,2)  |
| <b>Input parameter{in}</b>     |  |
| <i>int_flag</i>                | interrupt flag   |
| <i>I2C_INT_FLAG_SBSE_ND</i>    | start condition sent out in master mode interrupt flag                                       |
| <i>I2C_INT_FLAG_ADDS_END</i>   | address is sent in master mode or received and matches in slave mode interrupt flag          |
| <i>I2C_INT_FLAG_BTC</i>        | byte transmission finishes   |
| <i>I2C_INT_FLAG_ADD10_SEND</i> | header of 10-bit address is sent in master mode interrupt flag                               |
| <i>I2C_INT_FLAG_STPD_ET</i>    | stop condition detected in slave mode interrupt flag   |
| <i>I2C_INT_FLAG_RBNE</i>       | I2C_DATA is not empty during receiving interrupt flag  |
| <i>I2C_INT_FLAG_TBE</i>        | I2C_DATA is empty during transmitting interrupt flag   |
| <i>I2C_INT_FLAG_BERR</i>       | a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag |
| <i>I2C_INT_FLAG_LOSTA_RB</i>   | arbitration lost in master mode interrupt flag   |
| <i>I2C_INT_FLAG_AERR</i>       | acknowledge error interrupt flag   |
| <i>I2C_INT_FLAG_OUER_R</i>     | over-run or under-run situation occurs in slave mode interrupt flag                          |
| <i>I2C_INT_FLAG_PCEC_RR</i>    | PEC error when receiving data interrupt flag   |
| <i>I2C_INT_FLAG_SMBT</i>       | timeout signal in SMBus mode interrupt flag  |

|                              |                                   |
|------------------------------|-----------------------------------|
| O                            |                                   |
| I2C_INT_FLAG_SMBA<br>LT      | SMBus alert status interrupt flag |
| <b>Output parameter{out}</b> |                                   |
| -                            | -                                 |
| <b>Return value</b>          |                                   |
| -                            | -                                 |

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 3-544. Function i2c\_interrupt\_flag\_clear**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | i2c_interrupt_flag_clear  |
| <b>Function prototype</b>    | void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);           |
| <b>Function descriptions</b> | clear I2C interrupt flag  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| i2c_periph                   | I2C peripheral  |
| I2Cx                         | (x=0,1,2)   |
| <b>Input parameter{in}</b>   |   |
| int_flag                     | interrupt flag  |
| I2C_INT_FLAG_ADDS<br>END     | address is sent in master mode or received and matches in slave mode<br>interrupt flag          |
| I2C_INT_FLAG_BERR            | a bus error occurs indication a unexpected start or stop condition on I2C<br>bus interrupt flag |
| I2C_INT_FLAG_LOSTA<br>RB     | arbitration lost in master mode interrupt flag  |
| I2C_INT_FLAG_AERR            | acknowledge error interrupt flag  |
| I2C_INT_FLAG_OUER<br>R       | over-run or under-run situation occurs in slave mode interrupt flag                             |
| I2C_INT_FLAG_PECCE<br>RR     | PEC error when receiving data interrupt flag  |
| I2C_INT_FLAG_SMBTO<br>O      | timeout signal in SMBus mode interrupt flag   |
| I2C_INT_FLAG_SMBA            | SMBus alert status interrupt flag   |

|                              |   |
|------------------------------|---|
| <i>LT</i>                    |   |
| <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);
```

## 3.19. MISC

MISC is a software package that provide the interfaces for NVIC and SysTick. The NVIC and SysTick registers are listed in chapter [3.19.1](#), the MISC firmware functions are introduced in chapter [3.19.2](#).

### 3.19.1. Descriptions of Peripheral registers

**Table 3-545. NVIC Registers**

| Registers            | Descriptions                                     |
|----------------------|--|
| ISER <sup>(1)</sup>  | interrupt Set Enable Register                    |
| ICER <sup>(1)</sup>  | interrupt Clear Enable Register                  |
| ISPR <sup>(1)</sup>  | interrupt Set Pending Register                   |
| ICPR <sup>(1)</sup>  | interrupt Clear Pending Register                 |
| IABR <sup>(1)</sup>  | interrupt Active bit Register                    |
| IP <sup>(1)</sup>    | interrupt Priority Register                      |
| STIR <sup>(1)</sup>  | software Trigger Interrupt Register              |
| CPUID <sup>(2)</sup> | CPUID Base Register                              |
| ICSR <sup>(2)</sup>  | interrupt Control and State Register             |
| VTOR <sup>(2)</sup>  | vector Table Offset Register                     |
| AIRCR <sup>(2)</sup> | application Interrupt and Reset Control Register |
| SCR <sup>(2)</sup>   | system Control Register                          |
| CCR <sup>(2)</sup>   | configuration Control Register                   |
| SHP <sup>(2)</sup>   | system Handlers Priority Registers               |
| SHCSR <sup>(2)</sup> | system Handler Control and State Register        |
| CFSR <sup>(2)</sup>  | configurable Fault Status Register               |
| HFSR <sup>(2)</sup>  | hardFault Status Register                        |
| DFSR <sup>(2)</sup>  | debug Fault Status Register                      |
| MMFAR <sup>(2)</sup> | memManage Fault Address Register                 |
| BFAR <sup>(2)</sup>  | busFault Address Register                        |
| AFSR <sup>(2)</sup>  | auxiliary Fault Status Register                  |
| PFR <sup>(2)</sup>   | processor Feature Register                       |

| Registers            | Descriptions                        |
|----------------------|-------------------------------------|
| DFR <sup>(2)</sup>   | debug Feature Register              |
| ADR <sup>(2)</sup>   | auxiliary Feature Register          |
| MMFR <sup>(2)</sup>  | memory Model Feature Register       |
| ISAR <sup>(2)</sup>  | instruction Set Attributes Register |
| CPACR <sup>(2)</sup> | coprocessor Access Control Register |

1. refer to the structure NVIC\_Type, is defined in the core\_cm3.h file
2. refer to the structure SCB\_Type, is defined in the core\_cm3.h file

**Table 3-546. SysTick Registers**

| Registers            | Descriptions                        |
|----------------------|-------------------------------------|
| CTRL <sup>(1)</sup>  | sysTick Control and Status Register |
| LOAD <sup>(1)</sup>  | sysTick Reload Value Register       |
| VAL <sup>(1)</sup>   | sysTick Current Value Register      |
| CALIB <sup>(1)</sup> | sysTick Calibration Register        |

1. refer to the structure SysTick\_Type, is defined in the core\_cm3.h file

### 3.19.2. Descriptions of Peripheral functions

#### Enum IRQn\_Type

**Table 3-547. IRQn\_Type**

| Member name              | Function description            |
|--------------------------|---------------------------------|
| WWDGT_IRQHandler         | WWDGT interrupt                 |
| LVD_IRQHandler           | LVD from EXTI line interrupt    |
| TAMPER_IRQHandler        | tamper interrupt                |
| RTC_IRQHandler           | RTC global interrupt            |
| FMC_IRQHandler           | FMC global interrupt            |
| RCU_IRQHandler           | RCU global interrupt            |
| EXTI0_IRQHandler         | EXTI line0 interrupt            |
| EXTI1_IRQHandler         | EXTI line1 interrupt            |
| EXTI2_IRQHandler         | EXTI line2 interrupt            |
| EXTI3_IRQHandler         | EXTI line3 interrupt            |
| EXTI4_IRQHandler         | EXTI line4 interrupt            |
| DMA0_Channel0_IRQHandler | DMA0 channel 0 global interrupt |
| DMA0_Channel1_IRQHandler | DMA0 channel 1 global interrupt |
| DMA0_Channel2_IRQHandler | DMA0 channel 2 global interrupt |
| DMA0_Channel3_IRQHandler | DMA0 channel 3 global interrupt |
| DMA0_Channel4_IRQHandler | DMA0 channel 4 global interrupt |
| DMA0_Channel5_IRQHandler | DMA0 channel 5 global interrupt |
| DMA0_Channel6_IRQHandler | DMA0 channel 6 global interrupt |
| ADC0_1_IRQHandler        | ADC0 and ADC1 global interrupts |
| CAN0_TX_IRQHandler       | CAN0 TX interrupt               |

|                               |   |
|-------------------------------|---|
| CAN0_RX0_IRQHandler           | CAN0 RX0 interrupt  |
| CAN0_RX1_IRQHandler           | CAN0 RX1 interrupt  |
| CAN0_EWMC_IRQHandler          | CAN0 EWMC interrupt   |
| EXTI5_9_IRQHandler            | EXTI[9:5] interrupts  |
| TIMER0_BRK_TIMER8_IRQHandler  | TIMER0 break interrupt and TIMER8 global interrupt                    |
| TIMER0_UP_TIMER9_IRQHandler   | TIMER0 update Interrupt and TIMER9 global interrupt                   |
| TIMER0_TRG_CMT_TIMER10_IRQn   | TIMER0 trigger and commutation interrupt and TIMER10 global interrupt |
| TIMER0_Channel_IRQHandler     | TIMER0 channel capture compare interrupt                              |
| TIMER1_IRQHandler             | TIMER1 global interrupt   |
| TIMER2_IRQHandler             | TIMER2 global interrupt   |
| TIMER3_IRQHandler             | TIMER3 global interrupt   |
| I2C0_EV_IRQHandler            | I2C0 event interrupt  |
| I2C0_ER_IRQHandler            | I2C0 error interrupt  |
| I2C1_EV_IRQHandler            | I2C1 event interrupt  |
| I2C1_ER_IRQHandler            | I2C1 error interrupt  |
| SPI0_IRQHandler               | SPI0 global interrupt   |
| SPI1_IRQHandler               | SPI1 global interrupt   |
| USART0_IRQHandler             | USART0 global interrupt   |
| USART1_IRQHandler             | USART1 global interrupt   |
| USART2_IRQHandler             | USART2 global interrupt   |
| EXTI10_15_IRQHandler          | EXTI[15:10] interrupts  |
| RTC_Alarm_IRQHandler          | RTC alarm from EXTI line interrupt                                    |
| USBFS_WKUP_IRQHandler         | USBFS wakeUp from EXTI line interrupt                                 |
| TIMER7_BRK_TIMER11_IRQHandler | TIMER7 break interrupt and TIMER11 global interrupt                   |
| TIMER7_UP_TIMER12_IRQHandler  | TIMER7 update interrupt and TIMER12 global interrupt                  |
| TIMER7_TRG_CMT_TIMER13_IRQn   | TIMER7 Channel Capture Compare Interrupt                              |
| TIMER7_Channel_IRQHandler     | TIMER7 Channel Capture Compare Interrupt                              |
| ADC2_IRQHandler               | ADC2 global interrupt   |
| EXMC_IRQHandler               | EXMC global interrupt   |
| SDIO_IRQHandler               | SDIO global interrupt   |
| TIMER4_IRQHandler             | TIMER4 global interrupt   |
| SPI2_IRQHandler               | SPI2 global interrupt   |
| UART3_IRQHandler              | UART3 global interrupt  |
| UART4_IRQHandler              | UART4 global interrupt  |
| TIMER5_IRQHandler             | TIMER5 global interrupt   |
| TIMER6_IRQHandler             | TIMER6 global interrupt   |
| DMA1_Channel0_IRQHandler      | DMA1 Channel 0 global interrupt                                       |
| DMA1_Channel1_IRQHandler      | DMA1 Channel 1 global interrupt                                       |
| DMA1_Channel2_IRQHandler      | DMA1 Channel 2 global interrupt                                       |
| DMA1_Channel3_IRQHandler      | DMA1 Channel 3 global interrupt                                       |

|                          |   |
|--------------------------|---|
| DMA1_Channel4_IRQHandler | DMA1 Channel 4 global interrupt         |
| ENET_IRQHandler          | ENET global interrupt                   |
| ENET_WKUP_IRQHandler     | ENET wakeup through EXTI line interrupt |
| CAN1_TX_IRQHandler       | CAN1 TX interrupt                       |
| CAN1_RX0_IRQHandler      | CAN1 RX0 interrupt                      |
| CAN1_RX1_IRQHandler      | CAN1 RX1 interrupt                      |
| CAN1_EWMC_IRQHandler     | CAN1 EWMC interrupt                     |
| USBFS_IRQHandler         | USBFS global interrupt                  |
| DMA1_Channel5_IRQHandler | DMA1 Channel 5 global interrupt         |
| DMA1_Channel6_IRQHandler | DMA1 Channel 6 global interrupt         |
| USART5_IRQHandler        | USART5 global interrupt                 |
| I2C2_EV_IRQHandler       | I2C2 event interrupt                    |
| I2C2_ER_IRQHandler       | I2C2 error interrupt                    |
| DCI_IRQHandler           | DCI global interrupt                    |
| CAU_IRQHandler           | CAU global interrupt                    |
| HAU_TRNG_IRQHandler      | HAU and TRNG global interrupt           |
| UART6_IRQHandler         | UART6 global interrupt                  |
| UART7_IRQHandler         | UART7 global interrupt                  |
| TLI_IRQHandler           | TLI global interrupt                    |
| TLI_ER_IRQHandler        | TLI global error interrupt              |

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

**Table 3-548. MISC firmware function**

| Function name           | Function description                  |
|-------------------------|---------------------------------------|
| nvic_priority_group_set | set the priority group                |
| nvic_irq_enable         | enable NVIC interrupt request         |
| nvic_irq_disable        | disable NVIC interrupt request        |
| nvic_vector_table_set   | set the NVIC vector table address     |
| system_lowpower_set     | set the state of the low power mode   |
| system_lowpower_reset   | reset the state of the low power mode |
| systick_clksource_set   | set the systick clock source          |

### **nvic\_priority\_group\_set**

The description of nvic\_priority\_group\_set is shown as below:

**Table 3-549. Function nvic\_priority\_group\_set**

|                       |   |
|-----------------------|---|
| Function name         | nvic_priority_group_set                               |
| Function prototype    | void nvic_priority_group_set(uint32_t nvic_prigroup); |
| Function descriptions | configure bits length of the priority group           |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |

|                                 |  |
|---------------------------------|--|
| <b>nvic_prigroup</b>            | priority group   |
| <i>NVIC_PRIGROUP_PR_E0_SUB4</i> | 0 bits for pre-emption priority 4 bits for subpriority |
| <i>NVIC_PRIGROUP_PR_E1_SUB3</i> | 1 bits for pre-emption priority 3 bits for subpriority |
| <i>NVIC_PRIGROUP_PR_E2_SUB2</i> | 2 bits for pre-emption priority 2 bits for subpriority |
| <i>NVIC_PRIGROUP_PR_E3_SUB1</i> | 3 bits for pre-emption priority 1 bits for subpriority |
| <i>NVIC_PRIGROUP_PR_E4_SUB0</i> | 4 bits for pre-emption priority 0 bits for subpriority |
| <b>Output parameter{out}</b>    |  |
| -                               | -  |
| <b>Return value</b>             |  |
| -                               | -  |

Example:

```
/* priority group configuration , 0 bits for pre-emption priority 4 bits for subpriority */
nvic_priority_group_set(NVIC_PRIGROUP_PRE0_SUB4);
```

### **nvic\_irq\_enable**

The description of nvc\_irq\_enable is shown as below:

**Table 3-550. Function nvc\_irq\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | nvc_irq_enable  |
| <b>Function prototype</b>    | void nvc_irq_enable(uint8_t nvc_irq, uint8_t nvc_irq_pre_priority, uint8_t nvc_irq_sub_priority); |
| <b>Function descriptions</b> | enable NVIC request, configure the priority of interrupt  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | nvc_priority_group_set  |
| <b>Input parameter{in}</b>   |   |
| <b>nvc_irq</b>               | NVIC interrupt, refer to enum <a href="#">Table 3-547. IRQn_Type</a>                              |
| <b>Input parameter{in}</b>   |   |
| <b>nvc_irq_pre_priority</b>  | the pre-emption priority needed to set (0~4)  |
| <b>Input parameter{in}</b>   |   |
| <b>nvc_irq_sub_priority</b>  | the subpriority needed to set (0~4)   |
| <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 3-551. Function nvic\_irq\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | nvic_irq_disable   |
| <b>Function prototype</b>    | void nvic_irq_disable(uint8_t nvic_irq);                             |
| <b>Function descriptions</b> | disable NVIC request   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>nvic_irq</b>              | NVIC interrupt, refer to enum <a href="#">Table 3-547. 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 3-552. Function nvic\_vector\_table\_set**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | nvic_vector_table_set   |
| <b>Function prototype</b>    | void nvic_vector_table_set(uint32_t nvic_vict_tab, uint32_t offset);  |
| <b>Function descriptions</b> | set the NVIC vector table address                                     |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>nvic_vict_tab</b>         | the RAM or FLASH base address   |
| <b>NVIC_VECTTAB_RAM</b>      | RAM base address  |
| <b>NVIC_VECTTAB_FLASH</b>    | Flash base address  |
| <b>Input parameter{in}</b>   |   |
| <b>offset</b>                | Vector Table offset (vector table start address= base address+offset) |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |

Example:

```

/* set vector table address = NVIC_VECTTAB_FLASH + 0x200 */

#define NVIC_VECTTAB_FLASH ((uint32_t)0x08000000)

nvic_vector_table_set(NVIC_VECTTAB_FLASH, 0x200);
  
```

### **system\_lowpower\_set**

The description of system\_lowpower\_set is shown as below:

**Table 3-553. Function system\_lowpower\_set**

|   |   |
|---|---|
| <b>Function name</b>                      | system_lowpower_set   |
| <b>Function prototype</b>                 | void system_lowpower_set(uint8_t lowpower_mode);  |
| <b>Function descriptions</b>              | the state of the low power mode management  |
| <b>Precondition</b>                       | -   |
| <b>The called functions</b>               | -   |
| <b>Input parameter{in}</b>                |   |
| <b>lowpower_mode</b>                      | the low power mode state  |
| <i>SCB_LPM_SLEEP_EXI</i><br><i>T_ISR</i>  | if chose this para, the system always enter low power mode by exiting from ISR                |
| <i>SCB_LPM_DEEPSLEE</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 */

#define SCB_SCR_SLEEPONEXIT ((uint8_t)0x02)

#define SCB_LPM_SLEEP_EXIT_ISR SCB_SCR_SLEEPONEXIT

system_lowpower_set(SCB_LPM_SLEEP_EXIT_ISR);
  
```

### **system\_lowpower\_reset**

The description of system\_lowpower\_reset is shown as below:

**Table 3-554. Function system\_lowpower\_reset**

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

|                                 |  |
|---------------------------------|--|
| <b>Function prototype</b>       | void system_lowpower_reset(uint8_t lowpower_mode);                                 |
| <b>Function descriptions</b>    | the state of the low power mode management   |
| <b>Precondition</b>             | -  |
| <b>The called functions</b>     | -  |
| <b>Input parameter{in}</b>      |  |
| <b>lowpower_mode</b>            | the low power mode state   |
| <b>SCB_LPM_SLEEP_EXIT_T_ISR</b> | if chose this para, the system will exit low power mode by exiting from ISR        |
| <b>SCB_LPM_DEEPSLEEP_P</b>      | if chose this para, the system will enter the SLEEP mode                           |
| <b>SCB_LPM_WAKE_BY_ALL_INT</b>  | 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 */

#define SCB_SCR_SLEEPONEXIT ((uint8_t)0x02)

#define SCB_LPM_SLEEP_EXIT_ISR SCB_SCR_SLEEPONEXIT

system_lowpower_reset(SCB_LPM_SLEEP_EXIT_ISR);
```

### systick\_clksource\_set

The description of systick\_clksource\_set is shown as below:

**Table 3-555. 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               |
| <b>SYSTICK_CLKSOURCE_HCLK</b>      | systick clock source is from HCLK                       |
| <b>SYSTICK_CLKSOURCE_HCLK_DIV8</b> | systick clock source is from HCLK/8                     |
| <b>Output parameter{out}</b>       |   |
| -                                  | -   |
| <b>Return value</b>                |   |

Example:

```

/* systick clock source is HCLK/8 */

#define SYSTICK_CLKSOURCE_HCLK_DIV8 ((uint32_t)0xFFFFFFFFBU)

systick_clksource_set(SYSTICK_CLKSOURCE_HCLK_DIV8);
  
```

## 3.20. 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 [3.20.1](#), the PMU firmware functions are introduced in chapter [3.20.2](#).

### 3.20.1. Descriptions of Peripheral registers

PMU registers are listed in the table shown as below:

**Table 3-556. PMU Registers**

| Registers | Descriptions                |
|-----------|-----------------------------|
| PMU_CTL   | control register            |
| PMU_CS    | control and status register |

### 3.20.2. Descriptions of Peripheral functions

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

**Table 3-557. PMU firmware function**

| Function name            | Function description                                   |
|--------------------------|--|
| pmu_deinit               | reset PMU registers                                    |
| pmu_lvd_select           | select low voltage detector threshold                  |
| pmu_lvd_disable          | disable PMU lvd  |
| pmu_to_sleepmode         | PMU work at sleep mode                                 |
| pmu_to_deepsleepmode     | PMU work at deepsleep mode                             |
| pmu_to_standbymode       | PMU work at standby mode                               |
| pmu_wakeup_pin_enable    | enable PMU wakeup pin                                  |
| pmu_wakeup_pin_disable   | disable PMU wakeup pin                                 |
| pmu_backup_write_enable  | enable write access to the registers in backup domain  |
| pmu_backup_write_disable | disable write access to the registers in backup domain |
| pmu_flag_get             | get PMU flag   |
| pmu_flag_clear           | clear PMU flag   |

### **pmu\_deinit**

The description of pmu\_deinit is shown as below:

**Table 3-558. Function pmu\_deinit**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | pmu_deinit   |
| <b>Function prototype</b>    | void pmu_deinit(void);                             |
| <b>Function descriptions</b> | reset PMU registers                                |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | rcu_periph_reset_enable / rcu_periph_reset_disable |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* reset PMU */
pmu_deinit();
```

### **pmu\_lvd\_select**

The description of pmu\_lvd\_select is shown as below:

**Table 3-559. Function pmu\_lvd\_select**

|                              |                                       |
|------------------------------|---------------------------------------|
| <b>Function name</b>         | pmu_lvd_select                        |
| <b>Function prototype</b>    | void pmu_lvd_select(uint32_t lvdt_n); |
| <b>Function descriptions</b> | select low voltage detector threshold |
| <b>Precondition</b>          | -                                     |
| <b>The called functions</b>  | -                                     |
| <b>Input parameter{in}</b>   |                                       |
| <b>lvdt_n</b>                | voltage threshold value               |
| <i>PMU_LVDT_0</i>            | voltage threshold is 2.2V             |
| <i>PMU_LVDT_1</i>            | voltage threshold is 2.3V             |
| <i>PMU_LVDT_2</i>            | voltage threshold is 2.4V             |
| <i>PMU_LVDT_3</i>            | voltage threshold is 2.5V             |
| <i>PMU_LVDT_4</i>            | voltage threshold is 2.6V             |
| <i>PMU_LVDT_5</i>            | voltage threshold is 2.7V             |
| <i>PMU_LVDT_6</i>            | voltage threshold is 2.8V             |
| <i>PMU_LVDT_7</i>            | voltage threshold is 2.9V             |
| <b>Output parameter{out}</b> |                                       |
| -                            | -                                     |
| <b>Return value</b>          |                                       |

|   |   |   |
|---|---|---|
| - | - | - |
|---|---|---|

Example:

```
/* select low voltage detector threshold as 2.9V */
```

```
pmu_lvd_select (PMU_LVDT_7);
```

### **pmu\_lvd\_disable**

The description of pmu\_lvd\_disable is shown as below:

**Table 3-560. 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 3-561. 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                  |
| <b>WFI_CMD</b>               | use WFI command                              |
| <b>WFE_CMD</b>               | 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 3-562. 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 work at normal power mode when pmu enter deepsleep mode       |
| <i>PMU_LDO_LOWPOW<br/>ER</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 3-563. 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>  | -                              |

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

Example:

```
/* PMU work at standby mode */

pmu_to_standby();
```

### pmu\_wakeup\_pin\_enable

The description of pmu\_wakeup\_pin\_enable is shown as below:

**Table 3-564. 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 PMU wakeup pin             |
| <b>Precondition</b>          | -                                 |
| <b>The called functions</b>  | -                                 |
| Input parameter{in}          |                                   |
| -                            | -                                 |
| Output parameter{out}        |                                   |
| -                            | -                                 |
| Return value                 |                                   |
| -                            | -                                 |

Example:

```
/* enable wakeup pin */

pmu_wakeup_pin_enable();
```

### pmu\_wakeup\_pin\_disable

The description of pmu\_wakeup\_pin\_disable is shown as below:

**Table 3-565. 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 PMU wakeup pin              |
| <b>Precondition</b>          | -                                   |
| <b>The called functions</b>  | -                                   |
| Input parameter{in}          |                                     |
| -                            | -                                   |

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

Example:

```
/* disable wakeup pin */
pmu_wakeup_pin_disable();
```

### **pmu\_backup\_write\_enable**

The description of pmu\_backup\_write\_enable is shown as below:

**Table 3-566. 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 write access to the registers in backup domain |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| Input parameter{in}          |   |
| -                            | -   |
| Output parameter{out}        |   |
| -                            | -   |
| Return value                 |   |
| -                            | -   |

Example:

```
/* enable write access to the registers in backup domain */
pmu_backup_write_enable();
```

### **pmu\_backup\_write\_disable**

The description of pmu\_backup\_write\_disable is shown as below:

**Table 3-567. 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 write access to the registers in backup domain |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| Input parameter{in}          |  |
| -                            | -  |
| Output parameter{out}        |  |
| -                            | -  |

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

Example:

```
/* disable write access to the registers in backup domain */
pmu_backup_write_disable();
```

### pmu\_flag\_get

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

**Table 3-568. Function pmu\_flag\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | pmu_flag_get                            |
| <b>Function prototype</b>    | FlagStatus pmu_flag_get(uint32_t flag); |
| <b>Function descriptions</b> | get PMU flag                            |
| <b>Precondition</b>          | -                                       |
| <b>The called functions</b>  | -                                       |
| <b>Input parameter{in}</b>   |   |
| <b>flag</b>                  | PMU flag                                |
| <i>PMU_FLAG_WAKEUP</i>       | wakeup flag                             |
| <i>PMU_FLAG_STANDBY</i>      | standby flag                            |
| <i>PMU_FLAG_LVD</i>          | lvd flag                                |
| <b>Output parameter{out}</b> |   |
| -                            | -                                       |
| <b>Return value</b>          |   |
| <b>FlagStatus</b>            | SET or RESET                            |

Example:

```
/* get PMU wakeup flag */
FlagStatus status;
status = pmu_flag_get (PMU_FLAG_WAKEUP);
```

### pmu\_flag\_clear

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

**Table 3-569. 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 PMU flag                            |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |

| flag                          | PMU flag           |
|-------------------------------|--------------------|
| <i>PMU_FLAG_RESET_WAKEUP</i>  | reset wakeup flag  |
| <i>PMU_FLAG_RESET_STANDBY</i> | reset standby flag |
| <b>Output parameter{out}</b>  |                    |
| -                             | -                  |
| <b>Return value</b>           |                    |
| -                             | -                  |

Example:

```
/* clear PMU wakeup flag */

pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

## 3.21. 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 [3.21.1](#), the RCU firmware functions are introduced in chapter [3.21.2](#).

### 3.21.1. Descriptions of Peripheral registers

RCU registers are listed in the table shown as below:

**Table 3-570. 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_AHB1EN    | AHB1 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_AHB1RST   | AHB1 reset register              |
| RCU_CFG1      | configuration register 1         |
| RCU_DSV       | Deep-sleep mode voltage register |
| RCU_AHB2EN    | AHB2 enable register             |
| RCU_ADDAPB2EN | APB2 additional enable register  |

| Registers          | Descriptions                    |
|--------------------|---------------------------------|
| RCU_ADDAPB1EN      | APB1 additional enable register |
| RCU_AHB2RST        | AHB2 reset register             |
| RCU_ADDAPB2RS<br>T | APB2 additional reset register  |
| RCU_ADDAPB1RS<br>T | APB1 additional reset register  |
| RCU_CFG2           | configuration register 2        |
| RCU_PLLTCTL        | PLLT control register           |
| RCU_PLLTINT        | PLLT interrupt register         |
| RCU_PLLCFG         | PLLT configuration register     |

### 3.21.2. Descriptions of Peripheral functions

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

**Table 3-571. RCU firmware function**

| Function name                  | Function description                          |
|--------------------------------|---|
| rcu_deinit                     | deinitialize the RCU                          |
| rcu_periph_clock_enable        | enable the peripherals clock                  |
| rcu_periph_clock_disable       | disable the peripherals clock                 |
| rcu_periph_clock_sleep_enable  | enable the peripherals clock when sleep mode  |
| rcu_periph_clock_sleep_disable | disable the peripherals clock when sleep mode |
| rcu_periph_reset_enable        | 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_ckout1_config              | configure the CK_OUT1 clock source            |
| rcu_pll_config                 | configure the main PLL clock                  |
| rcu_pdrv0_config               | configure the PREDV0 division factor          |
| rcu_pdrv1_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_usbfs_trng_clock_config    | configure the USBFS/TRNG prescaler factor     |
| rcu_rtc_clock_config           | configure the RTC clock source selection      |
| rcu_i2s1_clock_config          | configure the I2S1 clock source selection     |

| <b>Function name</b>              | <b>Function description</b>   |
|-----------------------------------|---|
| rcu_i2s2_clock_config             | configure the I2S2 clock source selection                                       |
| rcu_pllt_config                   | configure the PLLT clock selection  |
| rcu_pllt_vco_config               | Configure the PLLT clock multiplication and division factors                    |
| rcu_tli_clock_config              | configure the TLI prescaler factor  |
| rcu_lxtal_drive_capability_config | configure the LXTAL drive capability  |
| rcu_osc_stab_wait                 | wait for oscillator stabilization flags is SET or oscillator startup is timeout |
| rcu_osc_on                        | turn on the oscillator  |
| rcu_osc_off                       | turn off the oscillator   |
| rcu_osc_bypass_mode_enable        | enable the oscillator bypass mode   |
| rcu_osc_bypass_mode_disable       | disable the oscillator bypass mode  |
| rcu_hxtal_clock_monitor_enable    | enable the HXTAL clock monitor  |
| rcu_hxtal_clock_monitor_disable   | disable the HXTAL clock monitor   |
| rcu_ir8m_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                                       |
| rcu_flag_get                      | get the clock stabilization and peripheral reset flags                          |
| rcu_all_reset_flag_clear          | clear all the reset flag  |
| rcu_interrupt_enable              | enable the stabilization interrupt  |
| rcu_interrupt_disable             | disable the stabilization interrupt   |
| rcu_interrupt_flag_get            | get the clock stabilization interrupt and ckm flags                             |
| rcu_interrupt_flag_clear          | clear the interrupt flags   |

### Enum rcu\_periph\_enum

**Table 3-572. rcu\_periph\_enum**

| <b>Member name</b> | <b>Function description</b>                      |
|--------------------|--|
| RCU_GPIOx          | GPIO ports clock (x=A,B,C,D,E,F,G ,H,I)          |
| RCU_AF             | alternate function clock                         |
| RCU_CRC            | CRC clock  |
| RCU_DMAx           | DMAx clock (x=0,1)                               |
| RCU_ENET           | ENET clock                                       |
| RCU_ENETTX         | ENETTX clock                                     |
| RCU_ENETRX         | ENETRX clock                                     |
| RCU_USBFS          | USBFS clock                                      |
| RCU_EXMC           | EXMC clock                                       |
| RCU_TIMERx         | TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13) |
| RCU_WWDGT          | WWDGT clock                                      |
| RCU_SPIx           | SPIx clock (x=0,1,2)                             |
| RCU_USARTx         | USARTx clock (x=0,1,2,5)                         |
| RCU_UARTx          | UARTx clock (x=3,4 ,6,7)                         |
| RCU_I2Cx           | I2Cx clock (x=0,1,2)                             |

| <b>Member name</b> | <b>Function description</b> |
|--------------------|-----------------------------|
| RCU_CANx           | CANx clock (x=0,1)          |
| RCU_PMU            | PMU clock                   |
| RCU_DAC            | DAC clock                   |
| RCU_RTC            | RTC clock                   |
| RCU_ADCx           | ADCx clock (x=0,1,2)        |
| RCU_SDIO           | SDIO clock                  |
| RCU_BKPI           | BKP interface clock         |
| RCU_TLI            | TLI clock                   |
| RCU_DCI            | DCI clock                   |
| RCU_CAU            | CAU clock                   |
| RCU_HAU            | HAU clock                   |
| RCU_TRNG           | TRNG clock                  |

**Enum rcu\_periph\_sleep\_enum**
**Table 3-573. rcu\_periph\_sleep\_enum**

| <b>Member name</b> | <b>Function description</b> |
|--------------------|-----------------------------|
| RCU_FMC_SLP        | FMC clock                   |
| RCU_SRAM_SLP       | SRAM clock                  |

**Enum rcu\_periph\_reset\_enum**
**Table 3-574. rcu\_periph\_reset\_enum**

| <b>Member name</b> | <b>Function description</b>                            |
|--------------------|--|
| RCU_GPIOxRST       | reset GPIO ports clock (x=A,B,C,D,E,F,G,H,I)           |
| RCU_AFRST          | reset alternate function clock                         |
| RCU_ENETRST        | reset ENET clock                                       |
| RCU_USBFSRST       | reset USBFS clock                                      |
| RCU_TIMERxRST      | reset TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13) |
| RCU_WWDGTRST       | reset WWDGT clock                                      |
| RCU_SPIxRST        | reset SPIx clock (x=0,1,2)                             |
| RCU_USARTxRST      | reset USARTx clock (x=0,1,2,5)                         |
| RCU_UARTxRST       | reset UARTx clock (x=3,4,6,7)                          |
| RCU_I2CxRST        | reset I2Cx clock (x=0,1,2)                             |
| RCU_CANxRST        | reset CANx clock (x=0,1)                               |
| RCU_PMURST         | reset PMU clock  |
| RCU_DACRST         | reset DAC clock  |
| RCU_ADCxRST        | reset ADCx clock (x=0,1,2)                             |
| RCU_BKPIRST        | reset BKPI clock                                       |
| RCU_TLIRST         | reset TLI  |
| RCU_DCIRST         | reset DCI  |
| RCU_CAURST         | reset CAU  |

| Member name | Function description |
|-------------|----------------------|
| RCU_HAURST  | reset HAU            |
| RCU_TRNGRST | reset TRNG           |

### Enum rcu\_flag\_enum

Table 3-575. rcu\_flag\_enum

| Member name         | Function description             |
|---------------------|----------------------------------|
| RCU_FLAG_IRC8M_STB  | IRC8M stabilization flag         |
| RCU_FLAG_HXTAL_STB  | HXTAL stabilization flag         |
| RCU_FLAG_PLLST_B    | PLL stabilization flag           |
| RCU_FLAG_PLL1S_TB   | PLL1 stabilization flag          |
| RCU_FLAG_PLL2S_TB   | PLL2 stabilization flag          |
| RCU_FLAG_PLLTS_TB   | PLLT stabilization flag          |
| RCU_FLAG_LXTAL_STB  | LXTAL stabilization flag         |
| RCU_FLAG_IRC40_KSTB | IRC40K stabilization flag        |
| RCU_FLAG_EPRS_T     | external PIN reset flag          |
| RCU_FLAG_PORR_ST    | power reset flag                 |
| RCU_FLAG_SWRS_T     | software reset flag              |
| RCU_FLAG_FWDG_TRST  | free watchdog timer reset flag   |
| RCU_FLAG_WWD_GTRST  | window watchdog timer reset flag |
| RCU_FLAG_LPRST      | low-power reset flag             |

### Enum rcu\_int\_flag\_enum

Table 3-576. rcu\_int\_flag\_enum

| Member name             | Function description                |
|-------------------------|-------------------------------------|
| RCU_INT_FLAG_IR_C40KSTB | IRC40K stabilization interrupt flag |
| RCU_INT_FLAG_L_XTALSTB  | LXTAL stabilization interrupt flag  |

| Member name            | Function description               |
|------------------------|------------------------------------|
| RCU_INT_FLAG_IR_C8MSTB | IRC8M stabilization interrupt flag |
| RCU_INT_FLAG_H_XTALSTB | HXTAL stabilization interrupt flag |
| RCU_INT_FLAG_P_LLSTB   | PLL stabilization interrupt flag   |
| RCU_INT_FLAG_P_LL1STB  | PLL1 stabilization interrupt flag  |
| RCU_INT_FLAG_P_LL2STB  | PLL2 stabilization interrupt flag  |
| RCU_INT_FLAG_C_KM      | HXTAL clock stuck interrupt flag   |
| RCU_INT_FLAG_P_LLTSTB  | PLLT stabilization interrupt flag  |

### Enum rcu\_int\_flag\_clear\_enum

Table 3-577. rcu\_int\_flag\_clear\_enum

| Member name                 | Function description                      |
|-----------------------------|---|
| RCU_INT_FLAG_IR_C40KSTB_CLR | IRC40K stabilization interrupt flag clear |
| RCU_INT_FLAG_L_XTALSTB_CLR  | LXTAL stabilization interrupt flag clear  |
| RCU_INT_FLAG_IR_C8MSTB_CLR  | IRC8M stabilization interrupt flag clear  |
| RCU_INT_FLAG_H_XTALSTB_CLR  | HXTAL stabilization interrupt flag clear  |
| RCU_INT_FLAG_P_LLSTB_CLR    | PLL stabilization interrupt flag clear    |
| RCU_INT_FLAG_P_LL1STB_CLR   | PLL1 stabilization interrupt flag clear   |
| RCU_INT_FLAG_P_LL2STB_CLR   | PLL2 stabilization interrupt flag clear   |
| RCU_INT_FLAG_C_KM_CLR       | clock stuck interrupt flag clear          |
| RCU_INT_FLAG_P_LLTSTB_CLR   | PLLT stabilization interrupt flag clear   |

### Enum rcu\_int\_enum

Table 3-578. rcu\_int\_enum

| Member name     | Function description                  |
|-----------------|---------------------------------------|
| RCU_INT_IRC40KS | IRC40K stabilization interrupt enable |

| Member name       | Function description                 |
|-------------------|--------------------------------------|
| TB                |                                      |
| RCU_INT_LXTALS_TB | LXTAL stabilization interrupt enable |
| RCU_INT_IRC8MS_TB | IRC8M stabilization interrupt enable |
| RCU_INT_HXTALS_TB | HXTAL stabilization interrupt enable |
| RCU_INT_PLLSTB    | PLL stabilization interrupt enable   |
| RCU_INT_PLL1STB   | PLL1 stabilization interrupt enable  |
| RCU_INT_PLL2STB   | PLL2 stabilization interrupt enable  |
| RCU_INT_PLLTSTB   | PLLT stabilization interrupt enable  |

### Enum rcu\_osc\_type\_enum

Table 3-579. rcu\_osc\_type\_enum

| Member name | Function description                 |
|-------------|--------------------------------------|
| RCU_IRC8M   | internal 8M RC oscillators(IRC8M)    |
| RCU_HXTAL   | high speed crystal oscillator(HXTAL) |
| RCU_PLL_CK  | phase locked loop(PLL)               |
| RCU_PLL1_CK | phase locked loop 1                  |
| RCU_PLL2_CK | phase locked loop 2                  |
| RCU_LXTAL   | low speed crystal oscillator(LXTAL)  |
| RCU_IRC40K  | internal 40K RC oscillator(IRC40K)   |
| RCU_PLLT_CK | TLI phase locked loop                |

### Enum rcu\_clock\_freq\_enum

Table 3-580. rcu\_clock\_freq\_enum

| Member name | Function description   |
|-------------|------------------------|
| CK_SYS      | system clock frequency |
| CK_AHB      | AHB clock frequency    |
| CK_APB1     | APB1 clock frequency   |
| CK_APB2     | APB2 clock frequency   |

### rcu\_deinit

The description of rcu\_deinit is shown as below:

Table 3-581. Function rcu\_deinit

|                       |                        |
|-----------------------|------------------------|
| Function name         | rcu_deinit             |
| Function prototype    | void rcu_deinit(void); |
| Function descriptions | deinitialize the RCU   |

|                              |                    |
|------------------------------|--------------------|
| <b>Precondition</b>          | -                  |
| <b>The called functions</b>  | rcu_osci_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 3-582. Function `rcu_periph_clock_enable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_periph_clock_enable  |
| <b>Function prototype</b>    | void rcu_periph_clock_enable(rcu_periph_enum periph);                                |
| <b>Function descriptions</b> | enable the peripherals clock   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>periph</b>                | RCU peripherals, refer to <a href="#">Table 3-572. <code>rcu_periph_enum</code>.</a> |
| <i>RCU_GPIOx</i>             | GPIO ports clock (x=A,B,C,D,E,F,G ,H,I)  |
| <i>RCU_AF</i>                | alternate function clock   |
| <i>RCU_CRC</i>               | CRC clock  |
| <i>RCU_DMAX</i>              | DMAX clock (x=0,1)   |
| <i>RCU_ENET</i>              | ENET clock   |
| <i>RCU_ENETTX</i>            | ENETTX clock   |
| <i>RCU_ENETRX</i>            | ENETRX clock   |
| <i>RCU_USBFS</i>             | USBFS clock  |
| <i>RCU_EXMC</i>              | EXMC clock   |
| <i>RCU_TIMERx</i>            | TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)                                     |
| <i>RCU_WWDGT</i>             | WWDGT clock  |
| <i>RCU_SPIx</i>              | SPIx clock (x=0,1,2)   |
| <i>RCU_USARTx</i>            | USARTx clock (x=0,1,2,5)   |
| <i>RCU_UARTx</i>             | UARTx clock (x=3,4 ,6,7)   |
| <i>RCU_I2Cx</i>              | I2Cx clock (x=0,1,2)   |
| <i>RCU_CANx</i>              | CANx clock (x=0,1)   |
| <i>RCU_PMU</i>               | PMU clock  |
| <i>RCU_DAC</i>               | DAC clock  |

|                              |                      |
|------------------------------|----------------------|
| <i>RCU_RTC</i>               | RTC clock            |
| <i>RCU_ADCx</i>              | ADCx clock (x=0,1,2) |
| <i>RCU_SDIO</i>              | SDIO clock           |
| <i>RCU_BKPI</i>              | BKP interface clock  |
| <i>RCU_TLI</i>               | TLI clock            |
| <i>RCU_DCI</i>               | DCI clock            |
| <i>RCU_CAU</i>               | CAU clock            |
| <i>RCU_HAU</i>               | HAU clock            |
| <i>RCU_TRNG</i>              | TRNG clock           |
| <b>Output parameter{out}</b> |                      |
| -                            | -                    |
| <b>Return value</b>          |                      |
| -                            | -                    |

Example:

```
/* enable the USART0 clock */
rcu_periph_clock_enable(RCU_USART0);
```

### **rcu\_periph\_clock\_disable**

The description of `rcu_periph_clock_disable` is shown as below:

**Table 3-583. Function `rcu_periph_clock_disable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | <code>rcu_periph_clock_disable</code>  |
| <b>Function prototype</b>    | <code>void rcu_periph_clock_disable(rcu_periph_enum periph);</code>                  |
| <b>Function descriptions</b> | disable the peripherals clock  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <i>periph</i>                | RCU peripherals, refer to <a href="#">Table 3-572. <code>rcu_periph_enum</code>.</a> |
| <i>RCU_GPIOx</i>             | GPIO ports clock (x=A,B,C,D,E,F,G ,H,I).   |
| <i>RCU_AF</i>                | alternate function clock   |
| <i>RCU_CRC</i>               | CRC clock  |
| <i>RCU_DMAx</i>              | DMAx clock (x=0,1)   |
| <i>RCU_ENET</i>              | ENET clock   |
| <i>RCU_ENETTX</i>            | ENETTX clock   |
| <i>RCU_ENETRX</i>            | ENETRX clock   |
| <i>RCU_USBFS</i>             | USBFS clock  |
| <i>RCU_EXMC</i>              | EXMC clock   |
| <i>RCU_TIMERx</i>            | TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)                                     |
| <i>RCU_WWDGT</i>             | WWDGT clock  |
| <i>RCU_SPIx</i>              | SPIx clock (x=0,1,2)   |
| <i>RCU_USARTx</i>            | USARTx clock (x=0,1,2,5)   |

|                              |                          |
|------------------------------|--------------------------|
| <i>RCU_UARTx</i>             | UARTx clock (x=3,4 ,6,7) |
| <i>RCU_I2Cx</i>              | I2Cx clock (x=0,1,2)     |
| <i>RCU_CANx</i>              | CANx clock (x=0,1)       |
| <i>RCU_PMU</i>               | PMU clock                |
| <i>RCU_DAC</i>               | DAC clock                |
| <i>RCU_RTC</i>               | RTC clock                |
| <i>RCU_ADCx</i>              | ADCx clock (x=0,1,2)     |
| <i>RCU_SDIO</i>              | SDIO clock               |
| <i>RCU_BKPI</i>              | BKP interface clock      |
| <i>RCU_TLI</i>               | TLI clock                |
| <i>RCU_DCI</i>               | DCI clock                |
| <i>RCU_CAU</i>               | CAU clock                |
| <i>RCU_HAU</i>               | HAU clock                |
| <i>RCU_TRNG</i>              | TRNG clock               |
| <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 3-584. Function `rcu_periph_clock_sleep_enable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | <code>rcu_periph_clock_sleep_enable</code>                                     |
| <b>Function prototype</b>    | <code>void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);</code> |
| <b>Function descriptions</b> | enable the peripherals clock when in sleep mode                                |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>periph</b>                | RCU peripherals, refer to <a href="#">Table 3-573. rcu_periph_sleep_enum</a> . |
| <i>RCU_FMC_SLP</i>           | FMC clock  |
| <i>RCU_SRAM_SLP</i>          | SRAM clock   |
| <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 3-585. 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 3-573. rcu_periph_sleep_enum</a> . |
| <i>RCU_FMC_SLP</i>           | FMC clock  |
| <i>RCU_SRAM_SLP</i>          | SRAM clock   |
| <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 3-586. 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 3-574. rcu_periph_reset_enum</a> . |
| <i>RCU_GPIOxRST</i>          | reset GPIO ports clock (x=A,B,C,D,E,F,G,H,I)   |
| <i>RCU_AFRST</i>             | reset alternate function clock   |
| <i>RCU_ENETRST</i>           | reset ENET clock   |
| <i>RCU_USBFSRST</i>          | reset USBFS clock  |
| <i>RCU_TIMERxRST</i>         | reset TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)                               |
| <i>RCU_WWDGTRST</i>          | reset WWDGT clock  |

|                              |                                |
|------------------------------|--------------------------------|
| <i>RCU_SPIxRST</i>           | reset SPIx clock (x=0,1,2)     |
| <i>RCU_USARTxRST</i>         | reset USARTx clock (x=0,1,2,5) |
| <i>RCU_UARTxRST</i>          | reset UARTx clock (x=3,4,6,7)  |
| <i>RCU_I2CxRST</i>           | reset I2Cx clock (x=0,1,2)     |
| <i>RCU_CANxRST</i>           | reset CANx clock (x=0,1)       |
| <i>RCU_PMURST</i>            | reset PMU clock                |
| <i>RCU_DACRST</i>            | reset DAC clock                |
| <i>RCU_ADCxRST</i>           | reset ADCx clock (x=0,1,2)     |
| <i>RCU_BKPIRST</i>           | reset BKPI clock               |
| <i>RCU_TLIRST</i>            | reset TLI                      |
| <i>RCU_DCIRST</i>            | reset DCI                      |
| <i>RCU_CAUrst</i>            | reset CAU                      |
| <i>RCU_HAURST</i>            | reset HAU                      |
| <i>RCU_TRNGRST</i>           | reset TRNG                     |
| <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 3-587. Function `rcu_periph_reset_disable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | <code>rcu_periph_reset_disable</code>  |
| <b>Function prototype</b>    | <code>void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);</code>      |
| <b>Function descriptions</b> | disable the peripheral reset   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <i>periph_reset</i>          | RCU peripherals reset, refer to <a href="#">Table 3-574. rcu_periph_reset_enum</a> . |
| <i>RCU_GPIOxRST</i>          | reset GPIO ports clock (x=A,B,C,D,E,F,G,H,I)   |
| <i>RCU_AFRST</i>             | reset alternate function clock   |
| <i>RCU_ENETRST</i>           | reset ENET clock   |
| <i>RCU_USBFSRST</i>          | reset USBFS clock  |
| <i>RCU_TIMERxRST</i>         | reset TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)                               |
| <i>RCU_WWDGTRST</i>          | reset WWDGT clock  |
| <i>RCU_SPIxRST</i>           | reset SPIx clock (x=0,1,2)   |
| <i>RCU_USARTxRST</i>         | reset USARTx clock (x=0,1,2,5)   |

|                              |                               |
|------------------------------|-------------------------------|
| <i>RCU_UARTxRST</i>          | reset UARTx clock (x=3,4,6,7) |
| <i>RCU_I2CxRST</i>           | reset I2Cx clock (x=0,1,2)    |
| <i>RCU_CANxRST</i>           | reset CANx clock (x=0,1)      |
| <i>RCU_PMURST</i>            | reset PMU clock               |
| <i>RCU_DACRST</i>            | reset DAC clock               |
| <i>RCU_ADCxRST</i>           | reset ADCx clock (x=0,1,2)    |
| <i>RCU_BKPIRST</i>           | reset BKPI clock              |
| <i>RCU_TLIRST</i>            | reset TLI                     |
| <i>RCU_DCIRST</i>            | reset DCI                     |
| <i>RCU_CAUrst</i>            | reset CAU                     |
| <i>RCU_HAURST</i>            | reset HAU                     |
| <i>RCU_TRNGRST</i>           | reset TRNG                    |
| <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 3-588. Function `rcu_bkp_reset_enable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | <code>rcu_bkp_reset_enable</code>             |
| <b>Function prototype</b>    | <code>void rcu_bkp_reset_enable(void);</code> |
| <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 3-589. 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 3-590. Function rcu\_system\_clock\_source\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rcu_system_clock_source_config                        |
| <b>Function prototype</b>    | void rcu_system_clock_source_config(uint32_t ck_sys); |
| <b>Function descriptions</b> | configure the system clock source                     |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>ck_sys</b>                | system clock source select                            |
| <i>RCU_CKSYSRC_IRC_8M</i>    | select CK_IRC8M as the CK_SYS source                  |
| <i>RCU_CKSYSRC_HXTAL</i>     | select CK_HXTAL as the CK_SYS source                  |
| <i>RCU_CKSYSRC_PLL</i>       | select CK_PLL as the CK_SYS source                    |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```

/* configure the CK_HXTAL as the CK_SYS source */

rcu_system_clock_source_config(RCU_CKSYSRC_HXTAL);

```

### **rcu\_system\_clock\_source\_get**

The description of `rcu_system_clock_source_get` is shown as below:

**Table 3-591. Function `rcu_system_clock_source_get`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rcu_system_clock_source_get                 |
| <b>Function prototype</b>    | uint32_t rcu_system_clock_source_get(void); |
| <b>Function descriptions</b> | get the system clock source                 |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>uint32_t</b>              | which clock is selected as CK_SYS source    |
| <i>RCU_SCSS_IRC8M</i>        | CK_IRC8M is selected as the CK_SYS source   |
| <i>RCU_SCSS_HXTAL</i>        | CK_HXTAL is selected as the CK_SYS source   |
| <i>RCU_SCSS_PLL</i>          | CK_PLL is selected as the CK_SYS source     |

Example:

```

uint32_t temp_cksys_status;

/* get the CK_SYS source */

temp_cksys_status = rcu_system_clock_source_get();

```

### **rcu\_ahb\_clock\_config**

The description of `rcu_ahb_clock_config` is shown as below:

**Table 3-592. Function `rcu_ahb_clock_config`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_ahb_clock_config                                     |
| <b>Function prototype</b>    | void rcu_ahb_clock_config(uint32_t ck_ahb);              |
| <b>Function descriptions</b> | configure the AHB clock prescaler selection              |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>ck_ahb</b>                | AHB clock prescaler selection                            |
| <i>RCU_AHB_CKSYS_DI_Vx</i>   | select CK_SYS / x, (x=1, 2, 4, 8, 16, 64, 128, 256, 512) |
| <b>Output parameter{out}</b> |  |

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

Example:

```
/* configure CK_SYS/128 */

rcu_ahb_clock_config(RCU_AHB_CKSYS_DIV128);
```

### rcu\_apb1\_clock\_config

The description of `rcu_apb1_clock_config` is shown as below:

**Table 3-593. Function `rcu_apb1_clock_config`**

|  |   |
|--|---|
| <b>Function name</b>                   | rcu_apb1_clock_config                         |
| <b>Function prototype</b>              | void rcu_apb1_clock_config(uint32_t ck_apb1); |
| <b>Function descriptions</b>           | configure the APB1 clock prescaler selection  |
| <b>Precondition</b>                    | -   |
| <b>The called functions</b>            | -   |
| <b>Input parameter{in}</b>             |   |
| <b>ck_apb1</b>                         | APB1 clock prescaler selection                |
| <i>RCU_APB1_CKAHB_D</i><br><i>IV1</i>  | select CK_AHB as CK_APB1                      |
| <i>RCU_APB1_CKAHB_D</i><br><i>IV2</i>  | select CK_AHB/2 as CK_APB1                    |
| <i>RCU_APB1_CKAHB_D</i><br><i>IV4</i>  | select CK_AHB/4 as CK_APB1                    |
| <i>RCU_APB1_CKAHB_D</i><br><i>IV8</i>  | select CK_AHB/8 as CK_APB1                    |
| <i>RCU_APB1_CKAHB_D</i><br><i>IV16</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 3-594. 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 3-595. Function rcu\_ckout0\_config**

|                                       |   |
|---------------------------------------|---|
| <b>Function name</b>                  | rcu_ckout0_config   |
| <b>Function prototype</b>             | void rcu_ckout0_config(uint32_t ckout0_src, uint32_t ckout0_div); |
| <b>Function descriptions</b>          | configure the CK_OUT0 clock source                                |
| <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_C8M</i>      | select high speed 8M internal oscillator clock |
| <i>RCU_CKOUT0SRC_HXTAL</i>       | select HXTAL                                   |
| <i>RCU_CKOUT0SRC_CKPLL_DIV2</i>  | select (CK_PLL / 2) clock                      |
| <i>RCU_CKOUT0SRC_CKPLL1</i>      | select CK_PLL1 clock                           |
| <i>RCU_CKOUT0SRC_CKPLL2_DIV2</i> | select (CK_PLL2 / 2) clock                     |
| <i>RCU_CKOUT0SRC_EXT1</i>        | select EXT1 clock                              |
| <i>RCU_CKOUT0SRC_CKPLL2</i>      | select CK_PLL2 clock                           |
| <b>Input parameter{in}</b>       |  |
| <b>ckout0_div</b>                | CK_OUT0 divider                                |
| <i>RCU_CKOUT0_DIVx(x=1..64)</i>  | CK_OUT0 is divided by x                        |
| <b>Output parameter{out}</b>     |  |
| -                                | -  |
| <b>Return value</b>              |  |
| -                                | -  |

Example:

```
/* configure the HXTAL as CK_OUT0 clock source */
rcu_ckout0_config(RCU_CKOUT0SRC_HXTAL, RCU_CKOUT0_DIV1);
```

### rcu\_ckout1\_config

The description of `rcu_ckout1_config` is shown as below:

**Table 3-596. Function `rcu_ckout1_config`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | <code>rcu_ckout1_config</code>   |
| <b>Function prototype</b>    | <code>void rcu_ckout1_config(uint32_t ckout1_src, uint32_t ckout1_div);</code> |
| <b>Function descriptions</b> | configure the CK_OUT1 clock source and divider                                 |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>ckout1_src</b>            | CK_OUT1 clock source selection   |
| <i>RCU_CKOUT1SRC_NONE</i>    | no clock selected  |
| <i>RCU_CKOUT1SRC_CKSYS</i>   | select system clock CK_SYS   |

|                                  |  |
|----------------------------------|--|
| <i>RCU_CKOUT1SRC_IRC8M</i>       | select high speed 8M internal oscillator clock |
| <i>RCU_CKOUT1SRC_XTAL</i>        | select HXTAL                                   |
| <i>RCU_CKOUT1SRC_CKPLL_DIV2</i>  | select (CK_PLL / 2) clock                      |
| <i>RCU_CKOUT0SRC_CKPLL1</i>      | select CK_PLL1 clock                           |
| <i>RCU_CKOUT1SRC_CKPLL2</i>      | select (CK_PLL2 / 2) clock                     |
| <i>RCU_CKOUT1SRC_EXT1</i>        | select EXT1 clock                              |
| <i>RCU_CKOUT1SRC_CKPLL2</i>      | select CK_PLL2 clock                           |
| <b>Input parameter{in}</b>       |  |
| <b>ckout1_div</b>                | CK_OUT1 divider                                |
| <i>RCU_CKOUT1_DIVx(x =1..64)</i> | CK_OUT1 is divided by x                        |
| <b>Output parameter{out}</b>     |  |
| -                                | -  |
| <b>Return value</b>              |  |
| -                                | -  |

Example:

```
/* configure the HXTAL as CK_OUT1 clock source */
rcu_ckout1_config(RCU_CKOUT1SRC_XTAL, RCU_CKOUT1_DIV1);
```

### rcu\_pll\_config

The description of rcu\_pll\_config is shown as below:

**Table 3-597. 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_DIV2</i> | IRC8M/2 clock is selected as source clock of PLL         |
| <i>RCU_PLLSRC_HXTAL</i>      | HXTAL 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>          | x = 2..14,16..32,6.5            |
| <b>Output parameter{out}</b> |                                 |
| -                            | -                               |
| <b>Return value</b>          |                                 |
| -                            | -                               |

Example:

```
/* configure the PLL */
rcu_pll_config(RCU_PLLSRC_HXTAL, RCU_PLL_MUL10);
```

### rcu\_predv0\_config

The description of rcu\_predv0\_config is shown as below:

**Table 3-598. 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 as PREDV0 input source clock                            |
| <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, RCU_PREDV0_DIV4);
```

### rcu\_predv1\_config

The description of rcu\_predv1\_config is shown as below:

**Table 3-599. Function rcu\_pdev1\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_pdev1_config                                 |
| <b>Function prototype</b>    | void rcu_pdev1_config(uint32_t pdev1_div);       |
| <b>Function descriptions</b> | configure the PREDV1 division factor             |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>pdev1_div</b>             | PREDV1 division factor                           |
| <b>RCU_PDEV1_DIVx</b>        | 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_pdev1_config(RCU_PDEV1_DIV8);
```

### **rcu\_pll1\_config**

The description of rcu\_pll1\_config is shown as below:

**Table 3-600. 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         |
| <b>RCU_PLL1_MULx</b>         | 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 3-601. 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 3-602. 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_CKAPB_2_DIV2</i>  | CK_ADC = CK_APB2 / 2                         |
| <i>RCU_CKADC_CKAPB_2_DIV4</i>  | CK_ADC = CK_APB2 / 4                         |
| <i>RCU_CKADC_CKAPB_2_DIV6</i>  | CK_ADC = CK_APB2 / 6                         |
| <i>RCU_CKADC_CKAPB_2_DIV8</i>  | CK_ADC = CK_APB2 / 8                         |
| <i>RCU_CKADC_CKAPB_2_DIV12</i> | CK_ADC = CK_APB2 / 12                        |
| <i>RCU_CKADC_CKAPB_2_DIV16</i> | CK_ADC = CK_APB2 / 16                        |
| <b>Output parameter{out}</b>   |  |

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

Example:

```
/* configure the ADC prescaler factor */

rcu_adc_clock_config(RCU_CKADC_CKAPB2_DIV8);
```

### **rcu\_usbfs\_trng\_clock\_config**

The description of `rcu_usbfs_trng_clock_config` is shown as below:

**Table 3-603. Function `rcu_usbfs_trng_clock_config`**

|                                     |  |
|-------------------------------------|--|
| <b>Function name</b>                | rcu_usbfs_trng_clock_config                                |
| <b>Function prototype</b>           | void rcu_usbfs_trng_clock_config(uint32_t usbfs_trng_psc); |
| <b>Function descriptions</b>        | configure the USBFS/TRNG prescaler factor                  |
| <b>Precondition</b>                 | -  |
| <b>The called functions</b>         | -  |
| <b>Input parameter{in}</b>          |  |
| <code>usbfs_trng_psc</code>         | USBFS/TRNG prescaler factor                                |
| <code>RCU_CKUSB_CKPLL_DIV1_5</code> | USBFS/TRNG prescaler select CK_PLL/1.5                     |
| <code>RCU_CKUSB_CKPLL_DIV1</code>   | USBFS/TRNG prescaler select CK_PLL/1                       |
| <code>RCU_CKUSB_CKPLL_DIV2_5</code> | USBFS/TRNG prescaler select CK_PLL/2.5                     |
| <code>RCU_CKUSB_CKPLL_DIV2</code>   | USBFS/TRNG prescaler select CK_PLL/2                       |
| <b>Output parameter{out}</b>        |  |
| -                                   | -  |
| <b>Return value</b>                 |  |
| -                                   | -  |

Example:

```
/* configure the USB prescaler factor */

rcu_usbfs_trng_clock_config(RCU_CKUSB_CKPLL_DIV2_5);
```

### **rcu\_rtc\_clock\_config**

The description of `rcu_rtc_clock_config` is shown as below:

**Table 3-604. Function `rcu_rtc_clock_config`**

|                           |   |
|---------------------------|---|
| <b>Function name</b>      | rcu_rtc_clock_config                                  |
| <b>Function prototype</b> | void rcu_rtc_clock_config(uint32_t rtc_clock_source); |

|                                 |  |
|---------------------------------|--|
| <b>Function descriptions</b>    | configure the RTC clock source selection |
| <b>Precondition</b>             | -  |
| <b>The called functions</b>     | -  |
| <b>Input parameter{in}</b>      |  |
| <b>rtc_clock_source</b>         | RTC clock source selection               |
| <i>RCU_RTCSRC_NONE</i>          | no clock selected                        |
| <i>RCU_RTCSRC_LXTAL</i>         | select CK_LXTAL as RTC source clock      |
| <i>RCU_RTCSRC_IRC40K</i>        | select CK_IRC40K as RTC source clock     |
| <i>RCU_RTCSRC_HXTAL_DIV_128</i> | select CK_HXTAL/128 as RTC source clock  |
| <b>Output parameter{out}</b>    |  |
| -                               | -  |
| <b>Return value</b>             |  |
| -                               | -  |

Example:

```
/* configure the RTC clock source selection */

rcu RTC_clock_config(RCU_RTCSRC_IRC40K);
```

### rcu\_i2s1\_clock\_config

The description of `rcu_i2s1_clock_config` is shown as below:

**Table 3-605. 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>        | I2S1 clock source selection                            |
| <i>RCU_I2S1SRC_CKSYS</i>       | select system clock as I2S1 source clock               |
| <i>RCU_I2S1SRC_CKPLL2_MUL2</i> | 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_CKSYS);
```

---

```
rcu_i2s1_clock_config(RCU_I2S1SRC_CKPLL2_MUL2);
```

### **rcu\_i2s2\_clock\_config**

The description of `rcu_i2s2_clock_config` is shown as below:

**Table 3-606. 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>        | I2S2 clock source selection                            |
| <i>RCU_I2S2SRC_CKSY_S</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\_pllt\_config**

The description of `rcu_pllt_config` is shown as below:

**Table 3-607. Function `rcu_pllt_config`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_pllt_config                          |
| <b>Function prototype</b>    | void rcu_pllt_config(uint32_t pllt_src); |
| <b>Function descriptions</b> | configure the PLLT clock selection       |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>pllt_src</b>              | PLLT clock source selection              |
| <i>RCU_PL LTSRC_IRC8_M</i>   | IRC8M selected as source clock of PLLT   |
| <i>RCU_PL LTSRC_HXTA_L</i>   | HXTAL selected as source clock of PLLT   |
| <b>Output parameter{out}</b> |  |

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

Example:

```
/* configure the PLLT clock selection */
rcu_pllt_config(RCU_PLLTSRC_IRC8M);
```

### rcu\_pllt\_vco\_config

The description of rcu\_pllt\_vco\_config is shown as below:

**Table 3-608. Function rcu\_pllt\_vco\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rcu_pllt_vco_config   |
| <b>Function prototype</b>    | ErrStatus rcu_pllt_vco_config(uint32_t pllt_psc, uint32_t pllt_mul, uint32_t pptr_psc); |
| <b>Function descriptions</b> | configure the PLLT clock multiplication and division factors                            |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| pllt_psc                     | the PLLT VCO input clock division factor  |
| parameter                    | between 2 and 63  |
| <b>Input parameter{in}</b>   |   |
| pllt_mul                     | the PLLT VCO output clock multiplication factor   |
| parameter                    | between 49 and 432  |
| <b>Input parameter{in}</b>   |   |
| pptr_psc                     | the PLLTR division factor   |
| parameter                    | between 2 and 7   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| ErrStatus                    | SUCCESS or ERROR  |

Example:

```
/* configure the PLLT clock multiplication and division factors */

if(SUCCESS == rcu_pllt_vco_config(0x4, 0x64, 0x4)){
}
```

### rcu\_tli\_clock\_config

The description of rcu\_tli\_clock\_config is shown as below:

**Table 3-609. Function rcu\_tli\_clock\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rcu_tli_clock_config                                |
| <b>Function prototype</b>    | void rcu_tli_clock_config(uint32_t tli_psc);        |
| <b>Function descriptions</b> | configure the TLI prescaler factor from PLLTR clock |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| tli_psc                      | TLI prescaler factor                                |
| RCU_CKTLI_CKPLLTR_DIV2       | TLI prescaler select CK_PLLTR/2                     |
| RCU_CKTLI_CKPLLTR_DIV4       | TLI prescaler select CK_PLLTR/4                     |
| RCU_CKTLI_CKPLLTR_DIV8       | TLI prescaler select CK_PLLTR/8                     |
| RCU_CKTLI_CKPLLTR_DIV16      | TLI prescaler select CK_PLLTR/16                    |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* configure the TLI prescaler factor from PLLTR clock */
rcu_tli_clock_config(RCU_CKTLI_CKPLLTR_DIV2);
```

### **rcu\_lxtal\_drive\_capability\_config**

The description of rcu\_lxtal\_drive\_capability\_config is shown as below:

**Table 3-610. 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>   |  |
| lxtal_dricap                 | drive capability of LXTAL                                      |
| RCU_LXTAL_LOWDRI             | lower driving capability                                       |
| RCU_LXTAL_MED_LOWDRI         | medium low driving capability                                  |
| RCU_LXTAL_MED_HIHDRI         | medium high driving capability                                 |
| RCU_LXTAL_HIGHDRI            | higher driving capability                                      |

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

Example:

```
/* configure the LXTAL drive capability */
rcu_lxtal_drive_capability_config(RCU_LXTAL_LOWDRI);
```

### rcu\_oscி\_stab\_wait

The description of rcu\_oscி\_stab\_wait is shown as below:

**Table 3-611. Function rcu\_oscி\_stab\_wait**

| Function name         | rcu_oscி_stab_wait  |
|-----------------------|---|
| Function prototype    | ErrStatus rcu_oscி_stab_wait(rcu_oscி_type_enum oscி);                          |
| Function descriptions | wait for oscillator stabilization flags is SET or oscillator startup is timeout |
| Precondition          | -   |
| The called functions  | rcu_flag_get  |
| Input parameter{in}   |   |
| <b>oscி</b>           | oscillator types, refer to <a href="#">Table 3-579. rcu_oscி_type enum</a> .    |
| <i>RCU_IRC8M</i>      | internal 8M RC oscillators(IRC8M)   |
| <i>RCU_HXTAL</i>      | high speed crystal oscillator(HXTAL)  |
| <i>RCU_PLL_CK</i>     | phase locked loop(PLL)  |
| <i>RCU_PLL1_CK</i>    | phase locked loop 1   |
| <i>RCU_PLL2_CK</i>    | phase locked loop 2   |
| <i>RCU_LXTAL</i>      | low speed crystal oscillator(LXTAL)   |
| <i>RCU_IRC40K</i>     | internal 40K RC oscillator(IRC40K)  |
| <i>RCU_PLLT_CK</i>    | TLI phase locked loop   |
| Output parameter{out} |   |
| -                     | -   |
| Return value          |   |
| <b>ErrStatus</b>      | SUCCESS or ERROR  |

Example:

```
/* wait for oscillator stabilization flag */
if(SUCCESS == rcu_oscி_stab_wait(RCU_HXTAL)){
}
```

### rcu\_oscி\_on

The description of rcu\_oscி\_on is shown as below:

**Table 3-612. Function rcu\_osci\_on**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_osci_on  |
| <b>Function prototype</b>    | void rcu_osci_on(rcu_osci_type_enum osci);                                   |
| <b>Function descriptions</b> | turn on the oscillator   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>osci</b>                  | oscillator types, refer to <a href="#">Table 3-579. rcu_osci_type_enum</a> . |
| <i>RCU_IRC8M</i>             | internal 8M RC oscillators(IRC8M)  |
| <i>RCU_HXTAL</i>             | high speed crystal oscillator(HXTAL)   |
| <i>RCU_PLL_CK</i>            | phase locked loop(PLL)   |
| <i>RCU_PLL1_CK</i>           | phase locked loop 1  |
| <i>RCU_PLL2_CK</i>           | phase locked loop 2  |
| <i>RCU_LXTAL</i>             | low speed crystal oscillator(LXTAL)  |
| <i>RCU_IRC40K</i>            | internal 40K RC oscillator(IRC40K)   |
| <i>RCU_PLLT_CK</i>           | TLI phase locked loop  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* turn on the high speed crystal oscillator */

rcu_osci_on(RCU_HXTAL);
```

### **rcu\_osci\_off**

The description of rcu\_osci\_off is shown as below:

**Table 3-613. Function rcu\_osci\_off**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | rcu_osci_off   |
| <b>Function prototype</b>    | void rcu_osci_off(rcu_osci_type_enum osci);                                  |
| <b>Function descriptions</b> | turn off the oscillator  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>osci</b>                  | oscillator types, refer to <a href="#">Table 3-579. rcu_osci_type_enum</a> . |
| <i>RCU_IRC8M</i>             | internal 8M RC oscillators(IRC8M)  |
| <i>RCU_HXTAL</i>             | high speed crystal oscillator(HXTAL)   |
| <i>RCU_PLL_CK</i>            | phase locked loop(PLL)   |
| <i>RCU_PLL1_CK</i>           | phase locked loop 1  |
| <i>RCU_PLL2_CK</i>           | phase locked loop 2  |
| <i>RCU_LXTAL</i>             | low speed crystal oscillator(LXTAL)  |

|                              |                                    |
|------------------------------|------------------------------------|
| <i>RCU_IRC40K</i>            | internal 40K RC oscillator(IRC40K) |
| <i>RCU_PLLT_CK</i>           | TLI phase locked loop              |
| <b>Output parameter{out}</b> |                                    |
| -                            | -                                  |
| <b>Return value</b>          |                                    |
| -                            | -                                  |

Example:

```
/* turn off the high speed crystal oscillator */

rcu_osci_off(RCU_HXTAL);
```

### **rcu\_osci\_bypass\_mode\_enable**

The description of *rcu\_osci\_bypass\_mode\_enable* is shown as below:

**Table 3-614. 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( <i>rcu_osci_type_enum osci</i> );                 |
| <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 3-579. <i>rcu_osci_type_enum</i></a> . |
| <i>RCU_HXTAL</i>             | high speed crystal oscillator(HXTAL)  |
| <i>RCU_LXTAL</i>             | low speed crystal oscillator(LXTAL)   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable the high speed crystal oscillator bypass mode */

rcu_osci_bypass_mode_enable(RCU_HXTAL);
```

### **rcu\_osci\_bypass\_mode\_disable**

The description of *rcu\_osci\_bypass\_mode\_disable* is shown as below:

**Table 3-615. 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( <i>rcu_osci_type_enum osci</i> ); |
| <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>  | -  |

| Input parameter{in}   |   |
|-----------------------|---|
| <b>osci</b>           | oscillator types, refer to <a href="#">Table 3-579. rcu_osc_type enum</a> . |
| <i>RCU_HXTAL</i>      | high speed crystal oscillator(HXTAL)  |
| <i>RCU_LXTAL</i>      | low speed crystal oscillator(LXTAL)   |
| Output parameter{out} |   |
| -                     | -   |
| Return value          |   |
| -                     | -   |

Example:

```
/* disable the high speed crystal oscillator bypass mode */
rcu_osc_bypass_mode_disable(RCU_HXTAL);
```

### **rcu\_hxtal\_clock\_monitor\_enable**

The description of `rcu_hxtal_clock_monitor_enable` is shown as below:

**Table 3-616. Function `rcu_hxtal_clock_monitor_enable`**

| <b>Function name</b>         | rcu_hxtal_clock_monitor_enable             |
|------------------------------|--|
| <b>Function prototype</b>    | void rcu_hxtal_clock_monitor_enable(void); |
| <b>Function descriptions</b> | enable the HXTAL clock monitor             |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| 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 3-617. Function `rcu_hxtal_clock_monitor_disable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | rcu_hxtal_clock_monitor_disable             |
| <b>Function prototype</b>    | void rcu_hxtal_clock_monitor_disable(void); |
| <b>Function descriptions</b> | disable the HXTAL clock monitor             |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |

| 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 3-618. Function `rcu_irc8m_adjust_value_set`**

| Function name             | <code>rcu_irc8m_adjust_value_set</code>                             |  |
|---------------------------|---|--|
| Function prototype        | <code>void rcu_irc8m_adjust_value_set(uint8_t irc8m_adjval);</code> |  |
| Function descriptions     | set the IRC8M adjust value  |  |
| Precondition              | -   |  |
| The called functions      | -   |  |
| Input parameter{in}       |   |  |
| <code>irc8m_adjval</code> | IRC8M adjust value, must be between 0 and 0x1F                      |  |
| Output parameter{out}     |   |  |
| -                         | -   |  |
| Return value              |   |  |
| -                         | -   |  |

Example:

```
/* set the IRC8M adjust value */

rcu_irc8m_adjust_value_set(0x10);
```

### **rcu\_deepsleep\_voltage\_set**

The description of `rcu_deepsleep_voltage_set` is shown as below:

**Table 3-619. Function `rcu_deepsleep_voltage_set`**

| Function name         | <code>rcu_deepsleep_voltage_set</code>                       |  |
|-----------------------|--|--|
| Function prototype    | <code>void rcu_deepsleep_voltage_set(uint32_t dsvol);</code> |  |
| Function descriptions | set the deep-sleep mode voltage value                        |  |
| Precondition          | -  |  |
| The called functions  | -  |  |
| Input parameter{in}   |  |  |
| <code>dsvol</code>    | deep sleep mode voltage                                      |  |

|                              |   |
|------------------------------|---|
| <i>RCU_DEEPSLEEP_V_1_2</i>   | the core voltage is 1.2V in deep-sleep mode |
| <i>RCU_DEEPSLEEP_V_1_1</i>   | the core voltage is 1.1V in deep-sleep mode |
| <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 |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* set the deep-sleep mode voltage */
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1_1);
```

### **rcu\_clock\_freq\_get**

The description of `rcu_clock_freq_get` is shown as below:

**Table 3-620. Function `rcu_clock_freq_get`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | <code>rcu_clock_freq_get</code>   |
| <b>Function prototype</b>    | <code>uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);</code>  |
| <b>Function descriptions</b> | get the system clock, bus clock frequency   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <code>clock</code>           | the clock frequency which to get, refer to <a href="#">Table 3-580.</a><br><a href="#"><u>rcu_clock_freq_enum</u></a> . |
| <code>CK_SYS</code>          | system clock frequency  |
| <code>CK_AHB</code>          | AHB clock frequency   |
| <code>CK_APB1</code>         | APB1 clock frequency  |
| <code>CK_APB2</code>         | APB2 clock frequency  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <code>ck_freq</code>         | 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);
```

### rcu\_flag\_get

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

**Table 3-621. 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 periphral reset flags  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>flag</b>                  | the clock stabilization and periphral reset flags, refer to <a href="#">Table 3-575. rcu_flag_enum</a> . |
| <i>RCU_FLAG_IRC8MSTB</i>     | IRC8M stabilization flag   |
| <i>RCU_FLAG_HXTALSTB</i>     | HXTAL stabilization flag   |
| <i>RCU_FLAG_PLLSTB</i>       | PLL stabilization flag   |
| <i>RCU_FLAG_PLL1STB</i>      | PLL1 stabilization flag  |
| <i>RCU_FLAG_PLL2STB</i>      | PLL2 stabilization flag  |
| <i>RCU_FLAG_PLLTSTB</i>      | PLLT stabilization flag  |
| <i>RCU_FLAG_LXTALSTB</i>     | LXTAL stabilization flag   |
| <i>RCU_FLAG_IRC40KSTB</i>    | IRC40K stabilization flag  |
| <i>RCU_FLAG_EPRST</i>        | external PIN reset flag  |
| <i>RCU_FLAG_PORRST</i>       | power reset flag   |
| <i>RCU_FLAG_SWRST</i>        | software reset flag  |
| <i>RCU_FLAG_FWDGTRST</i>     | free watchdog timer reset flag   |
| <i>RCU_FLAG_WWDGTRST</i>     | window watchdog timer reset flag   |
| <i>RCU_FLAG_LPRST</i>        | low-power reset flag   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>FlagStatus</b>            | SET or RESET   |

Example:

```
/* get the clock stabilization flag */

if(RESET != rcu_flag_get(RCU_FLAG_LXTALSTB)){
```

}

### rcu\_all\_reset\_flag\_clear

The description of rcu\_all\_reset\_flag\_clear is shown as below:

**Table 3-622. 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\_enable

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

**Table 3-623. 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 3-578. rcu_int_enum</a> . |
| <i>RCU_INT_IRC40KSTB</i>     | IRC40K stabilization interrupt enable   |
| <i>RCU_INT_LXTALSTB</i>      | LXTAL stabilization interrupt enable  |
| <i>RCU_INT_IRC8MSTB</i>      | IRC8M stabilization interrupt enable  |
| <i>RCU_INT_HXTALSTB</i>      | HXTAL stabilization interrupt enable  |
| <i>RCU_INT_PLLSTB</i>        | PLL stabilization interrupt enable  |
| <i>RCU_INT_PLL1STB</i>       | PLL1 stabilization interrupt enable   |
| <i>RCU_INT_PLL2STB</i>       | PLL2 stabilization interrupt enable   |
| <i>RCU_INT_PLLTSTB</i>       | PLLT stabilization interrupt enable   |
| <b>Output parameter{out}</b> |   |

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

Example:

```
/* enable the HXTAL stabilization interrupt */
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

### **rcu\_interrupt\_disable**

The description of `rcu_interrupt_disable` is shown as below:

**Table 3-624. 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 3-578. rcu int enum</a> . |
| <i>RCU_INT_IRC40KSTB</i>     | IRC40K stabilization interrupt enable   |
| <i>RCU_INT_LXTALSTB</i>      | LXTAL stabilization interrupt enable  |
| <i>RCU_INT_IRC8MSTB</i>      | IRC8M stabilization interrupt enable  |
| <i>RCU_INT_HXTALSTB</i>      | HXTAL stabilization interrupt enable  |
| <i>RCU_INT_PLLSTB</i>        | PLL stabilization interrupt enable  |
| <i>RCU_INT_PLL1STB</i>       | PLL1 stabilization interrupt enable   |
| <i>RCU_INT_PLL2STB</i>       | PLL2 stabilization interrupt enable   |
| <i>RCU_INT_PLLTSTB</i>       | PLLT stabilization interrupt enable   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable the HXTAL stabilization interrupt */
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

### **rcu\_interrupt\_flag\_get**

The description of `rcu_interrupt_flag_get` is shown as below:

**Table 3-625. 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 3-576. rcu_int_flag_enum</a> . |
| <i>RCU_INT_FLAG_IRC4OKSTB</i> | IRC40K stabilization interrupt flag  |
| <i>RCU_INT_FLAG_LXTALSTB</i>  | LXTAL stabilization interrupt flag   |
| <i>RCU_INT_FLAG_IRC8MSTB</i>  | IRC8M stabilization interrupt flag   |
| <i>RCU_INT_FLAG_HXTALSTB</i>  | HXTAL stabilization interrupt flag   |
| <i>RCU_INT_FLAG_PLLSTB</i>    | PLL stabilization interrupt flag   |
| <i>RCU_INT_FLAG_PLL1STB</i>   | PLL1 stabilization interrupt flag  |
| <i>RCU_INT_FLAG_PLL2STB</i>   | PLL2 stabilization interrupt flag  |
| <i>RCU_INT_FLAG_CKM</i>       | HXTAL clock stuck interrupt flag   |
| <i>RCU_INT_FLAG_PLLTSTB</i>   | PLL T stabilization interrupt flag   |
| <b>Output parameter{out}</b>  |  |
| -                             | -  |
| <b>Return value</b>           |  |
| <b>FlagStatus</b>             | SET or RESET   |

Example:

```
/* get the clock stabilization interrupt flag */

if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){
}
```

### **rcu\_interrupt\_flag\_clear**

The description of `rcu_interrupt_flag_clear` is shown as below:

**Table 3-626. Function `rcu_interrupt_flag_clear`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | <code>rcu_interrupt_flag_clear</code>  |
| <b>Function prototype</b>    | <code>void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag_clear)</code> |
| <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 3-577.</a><br><a href="#"><i>rcu_int_flag_clear_enum</i></a> . |
| <i>RCU_INT_FLAG_IRC4_OKSTB_CLR</i> | IRC40K stabilization interrupt flag clear  |
| <i>RCU_INT_FLAG_LXTA_LSTB_CLR</i>  | LXTAL stabilization interrupt flag clear   |
| <i>RCU_INT_FLAG_IRC8_MSTB_CLR</i>  | IRC8M stabilization interrupt flag clear   |
| <i>RCU_INT_FLAG_HXT_ALSTB_CLR</i>  | HXTAL stabilization interrupt flag clear   |
| <i>RCU_INT_FLAG_PLLS_TB_CLR</i>    | PLL stabilization interrupt flag clear   |
| <i>RCU_INT_FLAG_PLL1_STB_CLR</i>   | PLL1 stabilization interrupt flag clear  |
| <i>RCU_INT_FLAG_PLL2_STB_CLR</i>   | PLL2 stabilization interrupt flag clear  |
| <i>RCU_INT_FLAG_CKM_CLR</i>        | clock stuck interrupt flag clear   |
| <i>RCU_INT_FLAG_PLLT_STB_CLR</i>   | PLLT stabilization interrupt flag clear  |
| <b>Output parameter{out}</b>       |  |
| -                                  |  |
| <b>Return value</b>                |  |
| -                                  |  |

Example:

```
/* clear the interrupt HXTAL stabilization interrupt flag */
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

## 3.22. 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 [3.22.1](#), the RTC firmware functions are introduced in chapter [3.22.2](#).

### 3.22.1. Descriptions of Peripheral registers

RTC registers are listed in the table shown as below:

**Table 3-627. 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        |

### 3.22.2. Descriptions of Peripheral functions

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

**Table 3-628. RTC firmware function**

| Function name                | Function description                            |
|------------------------------|---|
| rtc_configuration_mode_enter | enter RTC configuration mode                    |
| rtc_configuration_mode_exit  | exit RTC configuration mode                     |
| rtc_lwoff_wait               | wait RTC last write operation finished flag set |
| rtc_register_sync_wait       | wait RTC registers synchronized flag set        |
| rtc_counter_get              | get RTC counter value                           |
| rtc_counter_set              | set RTC counter value                           |
| rtc_prescaler_set            | set RTC prescaler value                         |
| rtc_alarm_config             | set RTC alarm value                             |
| rtc_divider_get              | get RTC divider value                           |
| rtc_flag_get                 | get RTC flag status                             |
| rtc_flag_clear               | clear RTC 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 3-629. 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 3-630. 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\_lwoff\_wait**

The description of `rtc_lwoff_wait` is shown as below:

**Table 3-631. 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();
```

### **rtc\_register\_sync\_wait**

The description of `rtc_register_sync_wait` is shown as below:

**Table 3-632. 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\_counter\_get**

The description of `rtc_counter_get` is shown as below:

**Table 3-633. 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>          |                                 |

|          |                          |
|----------|--------------------------|
| uint32_t | the value of RTC counter |
|----------|--------------------------|

Example:

```

/* get the counter value */

uint32_t rtc_counter_value;

rtc_counter_value = rtc_counter_get();
  
```

### **rtc\_counter\_set**

The description of `rtc_counter_set` is shown as below:

**Table 3-634. 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>          | before using this function, you must call <code>rtc_lwoff_wait()</code> function (wait until LWOFF flag is set). |
| <b>The called functions</b>  | <code>rtc_configuration_mode_enter</code> / <code>rtc_configuration_mode_exit</code>                             |
| <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 3-635. Function `rtc_prescaler_set`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | 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 <code>rtc_lwoff_wait()</code> function (wait until LWOFF flag is set). |
| <b>The called functions</b>  | <code>rtc_configuration_mode_enter</code> / <code>rtc_configuration_mode_exit</code>                             |

| Input parameter{in}   |                                     |
|-----------------------|-------------------------------------|
| psc                   | RTC prescaler value (0-0x000F FFFF) |
| Output parameter{out} |                                     |
| -                     | -                                   |
| Return value          |                                     |
| -                     | -                                   |

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\_alarm\_config

The description of rtc\_alarm\_config is shown as below:

**Table 3-636. Function rtc\_alarm\_config**

| Function name  | rtc_alarm_config  |
|--|---|
| Function prototype   | void rtc_alarm_config(uint32_t alarm);  |
| Function descriptions                                      | set RTC alarm value   |
| Precondition   | before using this function, you must call rtc_lwoff_wait() function (wait until LWOFF flag is set). |
| The called functions                                       |   |
| rtc_configuration_mode_enter / rtc_configuration_mode_exit |   |
| Input parameter{in}  |   |
| alarm  | RTC alarm value (0-0xFFFF FFFF)   |
| Output parameter{out}                                      |   |
| -  | -   |
| Return value   |   |
| -  | -   |

Example:

```
/* wait until last write operation on RTC registers has finished */

rtc_lwoff_wait();

/* set alarm value to 0xFFFF */

rtc_alarm_config(0xFFFF);
```

### rtc\_divider\_get

The description of rtc\_divider\_get is shown as below:

**Table 3-637. 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\_get/rtc\_interrupt\_enable is shown as below:

**Table 3-638. 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><br><i>W</i> | overflow interrupt flag                 |
| <i>RTC_FLAG_RSYN</i>                 | registers synchronized flag             |
| <i>RTC_FLAG_LWOF</i>                 | last write operation finished flag      |
| <b>Output parameter{out}</b>         |   |
| -                                    | -                                       |
| <b>Return value</b>                  |   |
| <b>FlagStatus</b>                    | SET or RESET                            |

Example:

```
/* get the RTC alarm flag 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 3-639. Function `rtc_flag_clear`**

|                                      |  |
|--------------------------------------|--|
| <b>Function name</b>                 | rtc_flag_clear   |
| <b>Function prototype</b>            | void rtc_flag_clear(uint32_t flag);  |
| <b>Function descriptions</b>         | clear RTC flag status  |
| <b>Precondition</b>                  | before using this function, you must call <code>rtc_lwoff_wait()</code> function (wait until LWOFF flag is set). |
| <b>The called functions</b>          | -  |
| <b>Input parameter{in}</b>           |  |
| <b>flag</b>                          | specify which RTC flag status to clear   |
| <i>RTC_FLAG_SECOND</i>               | second interrupt flag  |
| <i>RTC_FLAG_ALARM</i>                | alarm interrupt flag   |
| <i>RTC_FLAG_OVERFLOW</i><br><i>W</i> | overflow interrupt flag  |
| <i>RTC_FLAG_RSYN</i>                 | registers synchronized flag  |
| <b>Output parameter{out}</b>         |  |
| -                                    | -  |
| <b>Return value</b>                  |  |
| -                                    | -  |

Example:

```

/* wait until last write operation on RTC registers has finished */

rtc_lwoff_wait();

/* clear the RTC alarm flag */

rtc_flag_clear(RTC_FLAG_ALARM);
  
```

### **rtc\_interrupt\_enable**

The description of `rtc_interrupt_enable` is shown as below:

**Table 3-640. 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 <code>rtc_lwoff_wait()</code> function (wait until LWOFF flag is set). |
| <b>The called functions</b>  | -  |

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

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 3-641. 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 <code>rtc_lwoff_wait()</code> 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);
```

## 3.23. SDIO

The secure digital input/output interface (SDIO) defines the SD, SD I/O, MMC and CE-ATA card host interface, which provides command/data transfer between the AHB system bus and SD memory cards, SD I/O cards, Multimedia Card (MMC) and CE-ATA devices. The SDIO registers are listed in chapter [3.23.1](#), the SDIO firmware functions are introduced in chapter [3.23.2](#).

### 3.23.1. Descriptions of Peripheral registers

SDIO registers are listed in the table shown as below:

**Table 3-642. SDIO Registers**

| Registers            | Descriptions                         |
|----------------------|--------------------------------------|
| SDIO_PWRCTL          | SDIO power control register          |
| SDIO_CLKCTL          | SDIO clock control register          |
| SDIO_CMDAGMT         | SDIO command argument register       |
| SDIO_CMDCTL          | SDIO command control register        |
| SDIO_RSPCMDIDX       | SDIO command index response register |
| SDIO_RESPx<br>x=0..3 | SDIO response register               |
| SDIO_DATATO          | SDIO data timeout register           |
| SDIO_DATALEN         | SDIO data length register            |
| SDIO_DATACTL         | SDIO data control register           |
| SDIO_DATACNT         | SDIO data counter register           |
| SDIO_STAT            | SDIO status register                 |
| SDIO_INTC            | SDIO interrupt clear register        |
| SDIO_INTEN           | SDIO interrupt enable register       |
| SDIO_FIFOCNT         | SDIO FIFO counter register           |
| SDIO_FIFO            | SDIO FIFO data register              |

### 3.23.2. Descriptions of Peripheral functions

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

**Table 3-643. SDIO firmware function**

| Function name               | Function description           |
|-----------------------------|--------------------------------|
| sdio_deinit                 | deinitialize the SDIO          |
| sdio_clock_config           | configure the SDIO clock       |
| sdio_hardware_clock_enable  | enable hardware clock control  |
| sdio_hardware_clock_disable | disable hardware clock control |

| <b>Function name</b>                  | <b>Function description</b>                                       |
|---------------------------------------|---|
| sdio_bus_mode_set                     | set different SDIO card bus mode                                  |
| sdio_power_state_set                  | set the SDIO power state  |
| sdio_power_state_get                  | get the SDIO power state  |
| sdio_clock_enable                     | enable SDIO_CLK clock output                                      |
| sdio_clock_disable                    | disable SDIO_CLK clock output                                     |
| sdio_command_response_config          | configure the command and response                                |
| sdio_wait_type_set                    | set the command state machine wait type                           |
| sdio_csm_enable                       | enable the CSM(command state machine)                             |
| sdio_csm_disable                      | disable the CSM(command state machine)                            |
| sdio_command_index_get                | get the last response command index                               |
| sdio_response_get                     | get the response for the last received command                    |
| sdio_data_config                      | configure the data timeout, data length and data block size       |
| sdio_data_transfer_config             | configure the data transfer mode and direction                    |
| sdio_dsm_enable                       | enable the DSM(data state machine) for data transfer              |
| sdio_dsm_disable                      | disable the DSM(data state machine)                               |
| sdio_data_write                       | write data(one word) to the transmit FIFO                         |
| sdio_data_read                        | read data(one word) from the receive FIFO                         |
| sdio_data_counter_get                 | get the number of remaining data bytes to be transferred to card  |
| sdio_fifo_counter_get                 | get the number of words remaining to be written or read from FIFO |
| sdio_dma_enable                       | enable the DMA request for SDIO                                   |
| sdio_dma_disable                      | disable the DMA request for SDIO                                  |
| sdio_readwait_enable                  | enable the read wait mode(SD I/O only)                            |
| sdio_readwait_disable                 | disable the read wait mode(SD I/O only)                           |
| sdio_stop_readwait_enable             | enable the function that stop the read wait process(SD I/O only)  |
| sdio_stop_readwait_disable            | disable the function that stop the read wait process(SD I/O only) |
| sdio_readwait_type_set                | set the read wait type(SD I/O only)                               |
| sdio_operation_enable                 | enable the SD I/O mode specific operation(SD I/O only)            |
| sdio_operation_disable                | disable the SD I/O mode specific operation(SD I/O only)           |
| sdio_suspend_enable                   | enable the SD I/O suspend operation(SD I/O only)                  |
| sdio_suspend_disable                  | disable the SD I/O suspend operation(SD I/O only)                 |
| sdio_ceata_command_enable             | enable the CE-ATA command(CE-ATA only)                            |
| sdio_ceata_command_disable            | disable the CE-ATA command(CE-ATA only)                           |
| sdio_ceata_interrupt_enable           | enable the CE-ATA interrupt(CE-ATA only)                          |
| sdio_ceata_interrupt_disable          | disable the CE-ATA interrupt(CE-ATA only)                         |
| sdio_ceata_command_completion_enable  | enable the CE-ATA command completion signal(CE-ATA only)          |
| sdio_ceata_command_completion_disable | disable the CE-ATA command completion signal(CE-ATA only)         |

| Function name             | Function description                      |
|---------------------------|---|
| able                      | only)                                     |
| sdio_flag_get             | get the flags state of SDIO               |
| sdio_flag_clear           | clear the pending flags of SDIO           |
| sdio_interrupt_enable     | enable the SDIO interrupt                 |
| sdio_interrupt_disable    | disable the SDIO interrupt                |
| sdio_interrupt_flag_get   | get the interrupt flags state of SDIO     |
| sdio_interrupt_flag_clear | clear the interrupt pending flags of SDIO |

### sdio\_deinit

The description of sdio\_deinit is shown as below:

**Table 3-644. Function sdio\_deinit**

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

Example:

```
/* deinitialize the SDIO */
sdio_deinit();
```

### sdio\_clock\_config

The description of sdio\_clock\_config is shown as below:

**Table 3-645. Function sdio\_clock\_config**

|                       |  |
|-----------------------|--|
| Function name         | sdio_clock_config  |
| Function prototype    | void sdio_clock_config(uint32_t clock_edge, uint32_t clock_bypass, uint32_t clock_powersave, uint16_t clock_division); |
| Function descriptions | configure the SDIO clock   |
| Precondition          | -  |
| The called functions  | -  |
| Input parameter{in}   |  |
| clock_edge            | SDIO_CLK clock edge  |
| SDIO_SDIOCLKEDGE      | select the rising edge of the SDIOCLK to generate SDIO_CLK   |

|                                  |   |
|----------------------------------|---|
| <u>_RISING</u>                   |   |
| <u>SDIO_SDIOCLKEDGE_FALLING</u>  | select the falling edge of the SDIOCLK to generate SDIO_CLK |
| <b>Input parameter{in}</b>       |   |
| <u>clock_bypass</u>              | clock bypass  |
| <u>SDIO_CLOCKBYPASS_ENABLE</u>   | clock bypass  |
| <u>SDIO_CLOCKBYPASS_DISABLE</u>  | no bypass   |
| <b>Input parameter{in}</b>       |   |
| <u>clock_powersave</u>           | SDIO_CLK clock dynamic switch on/off for power saving       |
| <u>SDIO_CLOCKPWRSAVE_ENABLE</u>  | SDIO_CLK closed when bus is idle                            |
| <u>SDIO_CLOCKPWRSAVE_DISABLE</u> | SDIO_CLK clock is always on                                 |
| <b>Input parameter{in}</b>       |   |
| <u>clock_division</u>            | clock division, less than 512                               |
| <b>Output parameter{out}</b>     |   |
| -                                | -   |
| <b>Return value</b>              |   |
| -                                | -   |

Example:

```
/* configure the SDIO clock */
sdio_clock_config(SDIO_SDIOCLKEDGE_RISING,      SDIO_CLOCKBYPASS_DISABLE,
SDIO_CLOCKPWRSAVE_DISABLE, SD_CLK_DIV_TRANS);
```

### **sdio\_hardware\_clock\_enable**

The description of `sdio_hardware_clock_enable` is shown as below:

**Table 3-646. Function `sdio_hardware_clock_enable`**

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

Example:

```
/* enable hardware clock control */

sdio_hardware_clock_enable();
```

### **sdio\_hardware\_clock\_disable**

The description of `sdio_hardware_clock_disable` is shown as below:

**Table 3-647. Function `sdio_hardware_clock_disable`**

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

Example:

```
/* disable hardware clock control */

sdio_hardware_clock_disable();
```

### **sdio\_bus\_mode\_set**

The description of `sdio_bus_mode_set` is shown as below:

**Table 3-648. Function `sdio_bus_mode_set`**

|                                     |  |
|-------------------------------------|--|
| <b>Function name</b>                | sdio_bus_mode_set                          |
| <b>Function prototype</b>           | void sdio_bus_mode_set(uint32_t bus_mode); |
| <b>Function descriptions</b>        | set different SDIO card bus mode           |
| <b>Precondition</b>                 | -  |
| <b>The called functions</b>         | -  |
| <b>Input parameter{in}</b>          |  |
| <b>bus_mode</b>                     | SDIO card bus mode                         |
| <b>SDIO_BUSMODE_1Bi</b><br><i>T</i> | 1-bit SDIO card bus mode                   |
| <b>SDIO_BUSMODE_4Bi</b><br><i>T</i> | 4-bit SDIO card bus mode                   |
| <b>SDIO_BUSMODE_8Bi</b><br><i>T</i> | 8-bit SDIO card bus mode                   |

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

Example:

```
/* set SDIO bus mode */

sdio_bus_mode_set(SDIO_BUSMODE_1BIT);
```

### **sdio\_power\_state\_set**

The description of `sdio_power_state_set` is shown as below:

**Table 3-649. Function `sdio_power_state_set`**

| <b>Function name</b>         | <code>sdio_power_state_set</code>                             |
|------------------------------|---|
| <b>Function prototype</b>    | <code>void sdio_power_state_set(uint32_t power_state);</code> |
| <b>Function descriptions</b> | set the SDIO power state                                      |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| Input parameter{in}          |   |
| <b>power_state</b>           | SDIO power state  |
| <b>SDIO_POWER_ON</b>         | SDIO power on   |
| <b>SDIO_POWER_OFF</b>        | SDIO power off  |
| Output parameter{out}        |   |
| -                            | -   |
| Return value                 |   |
| -                            | -   |

Example:

```
/* set SDIO power state */

sdio_power_state_set(SDIO_POWER_ON);
```

### **sdio\_power\_state\_get**

The description of `sdio_power_state_get` is shown as below:

**Table 3-650. Function `sdio_power_state_get`**

| <b>Function name</b>         | <code>sdio_power_state_get</code>                 |
|------------------------------|---|
| <b>Function prototype</b>    | <code>uint32_t sdio_power_state_get(void);</code> |
| <b>Function descriptions</b> | get the SDIO power state                          |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| Input parameter{in}          |   |
| -                            | -   |

| Output parameter{out} |                                |
|-----------------------|--------------------------------|
| -                     | -                              |
| Return value          |                                |
| uint32_t              | SDIO_POWER_ON / SDIO_POWER_OFF |

Example:

```
/* get the SDIO power state */

uint32_t sdio_power_value;

sdio_power_value = sdio_power_state_get();
```

### **sdio\_clock\_enable**

The description of `sdio_clock_enable` is shown as below:

**Table 3-651. Function `sdio_clock_enable`**

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

Example:

```
/* enable SDIO_CLK clock output */

sdio_clock_enable();
```

### **sdio\_clock\_disable**

The description of `sdio_clock_disable` is shown as below:

**Table 3-652. Function `sdio_clock_disable`**

| <b>Function name</b>         | sdio_clock_disable             |
|------------------------------|--------------------------------|
| <b>Function prototype</b>    | void sdio_clock_disable(void); |
| <b>Function descriptions</b> | disable SDIO_CLK clock output  |
| <b>Precondition</b>          | -                              |
| <b>The called functions</b>  | -                              |
| Input parameter{in}          |                                |
| -                            | -                              |

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

Example:

```
/* disable SDIO_CLK clock output */

sdio_clock_disable();
```

### **sdio\_command\_response\_config**

The description of `sdio_command_response_config` is shown as below:

**Table 3-653. Function `sdio_command_response_config`**

|                                       |  |
|---------------------------------------|--|
| Function name                         | <code>sdio_command_response_config</code>  |
| Function prototype                    | <code>void sdio_command_response_config(uint32_t cmd_index, uint32_t cmd_argument, uint32_t response_type);</code> |
| Function descriptions                 | configure the command and response   |
| Precondition                          | -  |
| The called functions                  | -  |
| Input parameter{in}                   |  |
| <code>cmd_index</code>                | command index, refer to the related specifications   |
| Input parameter{in}                   |  |
| <code>cmd_argument</code>             | command argument, refer to the related specifications  |
| Input parameter{in}                   |  |
| <code>response_type</code>            | response type  |
| <code>SDIO_RESPONSETYP_E_NO</code>    | no response  |
| <code>SDIO_RESPONSETYP_E_SHORT</code> | short response   |
| <code>SDIO_RESPONSETYP_E_LONG</code>  | long response  |
| Output parameter{out}                 |  |
| -                                     | -  |
| Return value                          |  |
| -                                     | -  |

Example:

```
/* CMD2(SD_CMD_ALL_SEND_CID) command response config */

sdio_command_response_config(SD_CMD_ALL_SEND_CID,
                             (uint32_t)0x0,
                             SDIO_RESPONSETYPE_LONG);
```

### sdio\_wait\_type\_set

The description of sdio\_wait\_type\_set is shown as below:

**Table 3-654. Function sdio\_wait\_type\_set**

|                                 |  |
|---------------------------------|--|
| <b>Function name</b>            | sdio_wait_type_set                           |
| <b>Function prototype</b>       | void sdio_wait_type_set(uint32_t wait_type); |
| <b>Function descriptions</b>    | set the command state machine wait type      |
| <b>Precondition</b>             | -  |
| <b>The called functions</b>     | -  |
| <b>Input parameter{in}</b>      |  |
| <b>wait_type</b>                | wait type                                    |
| <i>SDIO_WAITTYPE_NO</i>         | not wait interrupt                           |
| <i>SDIO_WAITTYPE_INT_ERRUPT</i> | wait interrupt                               |
| <i>SDIO_WAITTYPE_DAT_AEND</i>   | wait the end of data transfer                |
| <b>Output parameter{out}</b>    |  |
| -                               | -  |
| <b>Return value</b>             |  |
| -                               | -  |

Example:

```
/* set the command state machine wait type */
sdio_wait_type_set(SDIO_WAITTYPE_NO);
```

### sdio\_csm\_enable

The description of sdio\_csm\_enable is shown as below:

**Table 3-655. Function sdio\_csm\_enable**

|                              |                                       |
|------------------------------|---------------------------------------|
| <b>Function name</b>         | sdio_csm_enable                       |
| <b>Function prototype</b>    | void sdio_csm_enable(void);           |
| <b>Function descriptions</b> | enable the CSM(command state machine) |
| <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 CSM(command state machine) */
```

```
sdio_csm_enable();
```

### **sdio\_csm\_disable**

The description of sdio\_csm\_disable is shown as below:

**Table 3-656. Function sdio\_csm\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | sdio_csm_disable                       |
| <b>Function prototype</b>    | void sdio_csm_disable(void);           |
| <b>Function descriptions</b> | disable the CSM(command state machine) |
| <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 CSM(command state machine) */
```

```
sdio_csm_disable();
```

### **sdio\_command\_index\_get**

The description of sdio\_command\_index\_get is shown as below:

**Table 3-657. Function sdio\_command\_index\_get**

|                              |                                       |
|------------------------------|---------------------------------------|
| <b>Function name</b>         | sdio_command_index_get                |
| <b>Function prototype</b>    | uint8_t sdio_command_index_get(void); |
| <b>Function descriptions</b> | get the last response command index   |
| <b>Precondition</b>          | -                                     |
| <b>The called functions</b>  | -                                     |
| <b>Input parameter{in}</b>   |                                       |
| -                            | -                                     |
| <b>Output parameter{out}</b> |                                       |
| -                            | -                                     |
| <b>Return value</b>          |                                       |
| <b>uint8_t</b>               | last response command index           |

Example:

```
/* get SDIO command index */
```

```

  uint8_t sdio_command_value;
  sdio_command_value = sdio_command_index_get();

```

### **sdio\_response\_get**

The description of `sdio_response_get` is shown as below:

**Table 3-658. Function `sdio_response_get`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | sdio_response_get                              |
| <b>Function prototype</b>    | uint32_t sdio_response_get(uint32_t response); |
| <b>Function descriptions</b> | get the response for the last received command |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>response</b>              | SDIO response                                  |
| <i>SDIO_RESPONSE0</i>        | card response[31:0]/card response[127:96]      |
| <i>SDIO_RESPONSE1</i>        | card response[95:64]                           |
| <i>SDIO_RESPONSE2</i>        | card response[63:32]                           |
| <i>SDIO_RESPONSE3</i>        | card response[31:1], plus bit 0                |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>uint32_t</b>              | response for the last received command         |

Example:

```

/* store the CID0 numbers */
uint32_t sdio_cid[4] = {0, 0, 0, 0};
sdio_cid[0] = sdio_response_get(SDIO_RESPONSE0);

```

### **sdio\_data\_config**

The description of `sdio_data_config` is shown as below:

**Table 3-659. Function `sdio_data_config`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | sdio_data_config   |
| <b>Function prototype</b>    | void sdio_data_config(uint32_t data_timeout, uint32_t data_length, uint32_t data_blocksize); |
| <b>Function descriptions</b> | configure the data timeout, data length and data block size                                  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>data_timeout</b>          | data timeout period in card bus clock periods  |
| <b>Input parameter{in}</b>   |  |

|                                      |  |
|--------------------------------------|--|
| <b>data_length</b>                   | number of data bytes to be transferred |
| <b>Input parameter{in}</b>           |  |
| <b>data_blocksize</b>                | size of data block for block transfer  |
| <i>SDIO_DATABLOCKSIZE_1BYTE</i>      | block size = 1 byte                    |
| <i>SDIO_DATABLOCKSIZE_2BYTES</i>     | block size = 2 bytes                   |
| <i>SDIO_DATABLOCKSIZE_4BYTES</i>     | block size = 4 bytes                   |
| <i>SDIO_DATABLOCKSIZE_8BYTES</i>     | block size = 8 bytes                   |
| <i>SDIO_DATABLOCKSIZE_16BYTES</i>    | block size = 16 bytes                  |
| <i>SDIO_DATABLOCKSIZE_32BYTES</i>    | block size = 32 bytes                  |
| <i>SDIO_DATABLOCKSIZE_64BYTES</i>    | block size = 64 bytes                  |
| <i>SDIO_DATABLOCKSIZE_128BYTES</i>   | block size = 128 bytes                 |
| <i>SDIO_DATABLOCKSIZE_256BYTES</i>   | block size = 256 bytes                 |
| <i>SDIO_DATABLOCKSIZE_512BYTES</i>   | block size = 512 bytes                 |
| <i>SDIO_DATABLOCKSIZE_1024BYTES</i>  | block size = 1024 bytes                |
| <i>SDIO_DATABLOCKSIZE_2048BYTES</i>  | block size = 2048 bytes                |
| <i>SDIO_DATABLOCKSIZE_4096BYTES</i>  | block size = 4096 bytes                |
| <i>SDIO_DATABLOCKSIZE_8192BYTES</i>  | block size = 8192 bytes                |
| <i>SDIO_DATABLOCKSIZE_16384BYTES</i> | block size = 16384 bytes               |
| <b>Output parameter{out}</b>         |  |
| -                                    | -                                      |
| <b>Return value</b>                  |  |
| -                                    | -                                      |

Example:

```
/* configure SDIO data */
sdio_data_config(0, 0, SDIO_DATABLOCKSIZE_1BYTE);
```

### **sdio\_data\_transfer\_config**

The description of `sdio_data_transfer_config` is shown as below:

**Table 3-660. Function `sdio_data_transfer_config`**

|  |   |
|--|---|
| <b>Function name</b>                                     | <code>sdio_data_transfer_config</code>  |
| <b>Function prototype</b>                                | <code>void sdio_data_transfer_config(uint32_t transfer_mode, uint32_t transfer_direction);</code> |
| <b>Function descriptions</b>                             | configure the data transfer mode and direction  |
| <b>Precondition</b>                                      | -   |
| <b>The called functions</b>                              | -   |
| <b>Input parameter{in}</b>                               |   |
| <code>transfer_mode</code>                               | mode of data transfer   |
| <code>SDIO_TRANSMODE_BLOCK</code><br><code>LOCK</code>   | block transfer  |
| <code>SDIO_TRANSMODE_STREAM</code><br><code>TREAM</code> | stream transfer or SDIO multibyte transfer  |
| <b>Input parameter{in}</b>                               |   |
| <code>transfer_direction</code>                          | data transfer direction, read or write  |
| <code>SDIO_TRANSDIRECTI</code><br><code>ON_TOCARD</code> | write data to card  |
| <code>SDIO_TRANSDIRECTI</code><br><code>ON_TOSDIO</code> | read data from card   |
| <b>Output parameter{out}</b>                             |   |
| -  | -   |
| <b>Return value</b>                                      |   |
| -  | -   |

Example:

```
/* configure SDIO data transmission */

sdio_data_transfer_config(SDIO_TRANSDIRECTI_TOSDIO,
SDIO_TRANSMODE_BLOCK);
```

### **sdio\_dsm\_enable**

The description of `sdio_dsm_enable` is shown as below:

**Table 3-661. Function `sdio_dsm_enable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | <code>sdio_dsm_enable</code>                         |
| <b>Function prototype</b>    | <code>void sdio_dsm_enable(void);</code>             |
| <b>Function descriptions</b> | enable the DSM(data state machine) for data transfer |
| <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 DSM(data state machine) */

sdio_dsm_enable();
```

### **sdio\_dsm\_disable**

The description of `sdio_dsm_disable` is shown as below:

**Table 3-662. Function `sdio_dsm_disable`**

|                              |                                     |
|------------------------------|-------------------------------------|
| <b>Function name</b>         | sdio_dsm_disable                    |
| <b>Function prototype</b>    | void sdio_dsm_disable(void);        |
| <b>Function descriptions</b> | disable the DSM(data state machine) |
| <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 DSM(data state machine) */

sdio_dsm_disable();
```

### **sdio\_data\_write**

The description of `sdio_data_write` is shown as below:

**Table 3-663. Function `sdio_data_write`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | sdio_data_write                           |
| <b>Function prototype</b>    | void sdio_data_write(uint32_t data);      |
| <b>Function descriptions</b> | write data(one word) to the transmit FIFO |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>data</b>                  | 32-bit data write to card                 |
| <b>Output parameter{out}</b> |   |

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

Example:

```
/* write data(one word) to the transmit FIFO */

sdio_data_write(0x0000 0001);
```

### **sdio\_data\_read**

The description of `sdio_data_read` is shown as below:

**Table 3-664. Function `sdio_data_read`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | sdio_data_read                            |
| <b>Function prototype</b>    | uint32_t sdio_data_read(void);            |
| <b>Function descriptions</b> | read data(one word) from the receive FIFO |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>uint32_t</b>              | received data                             |

Example:

```
/* read data(one word) from the receive FIFO */

sdio_data_read();
```

### **sdio\_data\_counter\_get**

The description of `sdio_data_counter_get` is shown as below:

**Table 3-665. Function `sdio_data_counter_get`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | sdio_data_counter_get  |
| <b>Function prototype</b>    | uint32_t sdio_data_counter_get(void);                            |
| <b>Function descriptions</b> | get the number of remaining data bytes to be transferred to card |
| <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> | number of remaining data bytes to be transferred |
|-----------------|--|

Example:

```
/* get the number of remaining data bytes to be transferred to card */

uint32_t sdio_data_value;

sdio_data_value = sdio_data_counter_get();
```

### **sdio\_fifo\_counter\_get**

The description of `sdio_fifo_counter_get` is shown as below:

**Table 3-666. Function `sdio_data_counter_get`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | sdio_fifo_counter_get   |
| <b>Function prototype</b>    | uint32_t sdio_fifo_counter_get(void);                             |
| <b>Function descriptions</b> | get the number of words remaining to be written or read from FIFO |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>uint32_t</b>              | remaining number of words   |

Example:

```
/* get the number of words remaining to be written or read from FIFO */

uint32_t sdio_fifo_value;

sdio_fifo_value = sdio_fifo_counter_get();
```

### **sdio\_dma\_enable**

The description of `sdio_dma_enable` is shown as below:

**Table 3-667. Function `sdio_dma_enable`**

|                              |                                 |
|------------------------------|---------------------------------|
| <b>Function name</b>         | sdio_dma_enable                 |
| <b>Function prototype</b>    | void sdio_dma_enable(void);     |
| <b>Function descriptions</b> | enable the DMA request for SDIO |
| <b>Precondition</b>          | -                               |
| <b>The called functions</b>  | -                               |
| <b>Input parameter{in}</b>   |                                 |
| -                            | -                               |
| <b>Output parameter{out}</b> |                                 |
| -                            | -                               |

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

Example:

```
/* enable the SDIO DMA */

sdio_dma_enable();
```

### **sdio\_dma\_disable**

The description of `sdio_dma_disable` is shown as below:

**Table 3-668. Function `sdio_dma_disable`**

|                              |                                  |
|------------------------------|----------------------------------|
| <b>Function name</b>         | sdio_dma_disable                 |
| <b>Function prototype</b>    | void sdio_dma_disable(void);     |
| <b>Function descriptions</b> | disable the DMA request for SDIO |
| <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 SDIO DMA */

sdio_dma_disable();
```

### **sdio\_readwait\_enable**

The description of `sdio_readwait_enable` is shown as below:

**Table 3-669. Function `sdio_readwait_enable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | sdio_readwait_enable                   |
| <b>Function prototype</b>    | void sdio_readwait_enable(void);       |
| <b>Function descriptions</b> | enable the read wait mode(SD I/O only) |
| <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 read wait mode(SD I/O only) */

sdio_readwait_enable();
```

### **sdio\_readwait\_disable**

The description of `sdio_readwait_disable` is shown as below:

**Table 3-670. Function `sdio_readwait_disable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | sdio_readwait_disable                   |
| <b>Function prototype</b>    | void sdio_readwait_disable(void);       |
| <b>Function descriptions</b> | disable the read wait mode(SD I/O only) |
| <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 read wait mode(SD I/O only) */

sdio_readwait_disable();
```

### **sdio\_stop\_readwait\_enable**

The description of `sdio_stop_readwait_enable` is shown as below:

**Table 3-671. Function `sdio_stop_readwait_enable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | sdio_stop_readwait_enable  |
| <b>Function prototype</b>    | void sdio_stop_readwait_enable(void);                            |
| <b>Function descriptions</b> | enable the function that stop the read wait process(SD I/O only) |
| <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 function that stop the read wait process(SD I/O only) */
```

```
sdio_stop_readwait_enable();
```

### **sdio\_stop\_readwait\_disable**

The description of `sdio_stop_readwait_disable` is shown as below:

**Table 3-672. Function `sdio_stop_readwait_disable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | sdio_stop_readwait_disable  |
| <b>Function prototype</b>    | void sdio_stop_readwait_disable(void);                            |
| <b>Function descriptions</b> | disable the function that stop the read wait process(SD I/O only) |
| <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 function that stop the read wait process(SD I/O only) */
```

```
sdio_stop_readwait_disable();
```

### **sdio\_readwait\_type\_set**

The description of `sdio_readwait_type_set` is shown as below:

**Table 3-673. Function `sdio_readwait_type_set`**

|                                |  |
|--------------------------------|--|
| <b>Function name</b>           | sdio_readwait_type_set                               |
| <b>Function prototype</b>      | void sdio_readwait_type_set(uint32_t readwait_type); |
| <b>Function descriptions</b>   | set the read wait type(SD I/O only)                  |
| <b>Precondition</b>            | -  |
| <b>The called functions</b>    | -  |
| <b>Input parameter{in}</b>     |  |
| <b>readwait_type</b>           | SD I/O read wait type                                |
| <b>SDIO_READWAITTYP_E_CLK</b>  | read wait control by stopping SDIO_CLK               |
| <b>SDIO_READWAITTYP_E_DAT2</b> | read wait control using SDIO_DAT[2]                  |
| <b>Output parameter{out}</b>   |  |
| -                              | -  |
| <b>Return value</b>            |  |
| -                              | -  |

Example:

```
/* set the read wait type(SD I/O only) */
sdio_readwait_type_set(SDIO_READWAITTYPE_CLK);
```

### **sdio\_operation\_enable**

The description of `sdio_operation_enable` is shown as below:

**Table 3-674. Function `sdio_operation_enable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | sdio_operation_enable                                  |
| <b>Function prototype</b>    | void sdio_operation_enable(void);                      |
| <b>Function descriptions</b> | enable the SD I/O mode specific operation(SD I/O only) |
| <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 SD I/O mode specific operation(SD I/O only) */
sdio_operation_enable();
```

### **sdio\_operation\_disable**

The description of `sdio_operation_disable` is shown as below:

**Table 3-675. Function `sdio_operation_disable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | sdio_operation_disable                                  |
| <b>Function prototype</b>    | void sdio_operation_disable(void);                      |
| <b>Function descriptions</b> | disable the SD I/O mode specific operation(SD I/O only) |
| <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 SD I/O mode specific operation(SD I/O only) */
```

```
void sdio_operation_disable();
```

### **sdio\_suspend\_enable**

The description of `sdio_suspend_enable` is shown as below:

**Table 3-676. Function `sdio_suspend_enable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | sdio_suspend_enable                              |
| <b>Function prototype</b>    | void sdio_suspend_enable(void);                  |
| <b>Function descriptions</b> | enable the SD I/O suspend operation(SD I/O only) |
| <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 SD I/O suspend operation(SD I/O only) */
```

```
sdio_suspend_enable();
```

### **sdio\_suspend\_disable**

The description of `sdio_suspend_disable` is shown as below:

**Table 3-677. Function `sdio_suspend_disable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | sdio_suspend_disable                              |
| <b>Function prototype</b>    | void sdio_suspend_disable(void);                  |
| <b>Function descriptions</b> | disable the SD I/O suspend operation(SD I/O only) |
| <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 SD I/O suspend operation(SD I/O only) */
```

```
sdio_suspend_disable();
```

### **sdio\_ceata\_command\_enable**

The description of `sdio_ceata_command_enable` is shown as below:

**Table 3-678. Function `sdio_ceata_command_enable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | sdio_ceata_command_enable              |
| <b>Function prototype</b>    | void sdio_ceata_command_enable(void);  |
| <b>Function descriptions</b> | enable the CE-ATA command(CE-ATA only) |
| <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 CE-ATA command(CE-ATA only) */

sdio_ceata_command_enable();
```

### **sdio\_ceata\_command\_disable**

The description of `sdio_ceata_command_disable` is shown as below:

**Table 3-679. Function `sdio_ceata_command_disable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | sdio_ceata_command_disable              |
| <b>Function prototype</b>    | void sdio_ceata_command_disable(void);  |
| <b>Function descriptions</b> | disable the CE-ATA command(CE-ATA only) |
| <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 CE-ATA command(CE-ATA only) */

sdio_ceata_command_disable();
```

### **sdio\_ceata\_interrupt\_enable**

The description of `sdio_ceata_interrupt_enable` is shown as below:

**Table 3-680. Function `sdio_ceata_interrupt_enable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | sdio_ceata_interrupt_enable              |
| <b>Function prototype</b>    | void sdio_ceata_interrupt_enable(void);  |
| <b>Function descriptions</b> | enable the CE-ATA interrupt(CE-ATA only) |
| <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 CE-ATA interrupt(CE-ATA only) */
sdio_ceata_interrupt_enable();
```

### **sdio\_ceata\_interrupt\_disable**

The description of `sdio_ceata_interrupt_disable` is shown as below:

**Table 3-681. Function `sdio_ceata_interrupt_disable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | sdio_ceata_interrupt_disable              |
| <b>Function prototype</b>    | void sdio_ceata_interrupt_disable(void);  |
| <b>Function descriptions</b> | disable the CE-ATA interrupt(CE-ATA only) |
| <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 CE-ATA interrupt(CE-ATA only) */
sdio_ceata_interrupt_disable();
```

### **sdio\_ceata\_command\_completion\_enable**

The description of `sdio_ceata_command_completion_enable` is shown as below:

**Table 3-682. Function `sdio_ceata_command_completion_enable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | <code>sdio_ceata_command_completion_enable</code>             |
| <b>Function prototype</b>    | <code>void sdio_ceata_command_completion_enable(void);</code> |
| <b>Function descriptions</b> | enable the CE-ATA command completion signal(CE-ATA only)      |
| <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 CE-ATA command completion signal(CE-ATA only) */
sdio_ceata_command_completion_enable();
```

### **sdio\_ceata\_command\_completion\_disable**

The description of `sdio_ceata_command_completion_disable` is shown as below:

**Table 3-683. Function `sdio_ceata_command_completion_disable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | <code>sdio_ceata_command_completion_disable</code>             |
| <b>Function prototype</b>    | <code>void sdio_ceata_command_completion_disable(void);</code> |
| <b>Function descriptions</b> | disable the CE-ATA command completion signal(CE-ATA only)      |
| <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 CE-ATA command completion signal(CE-ATA only) */
sdio_ceata_command_completion_disable();
```

### **sdio\_flag\_get**

The description of `sdio_flag_get` is shown as below:

**Table 3-684. Function `sdio_flag_get`**

|  |   |
|--|---|
| <b>Function name</b>                       | <code>sdio_flag_get</code>  |
| <b>Function prototype</b>                  | <code>FlagStatus sdio_flag_get(uint32_t flag);</code>                           |
| <b>Function descriptions</b>               | get the flags state of SDIO   |
| <b>Precondition</b>                        | -   |
| <b>The called functions</b>                | -   |
| <b>Input parameter{in}</b>                 |   |
| <b>flag</b>                                | flags state of SDIO   |
| <code>SDIO_FLAG_CCRCER</code><br><i>R</i>  | command response received (CRC check failed) flag                               |
| <code>SDIO_FLAG_DTCRCE</code><br><i>RR</i> | data block sent/received (CRC check failed) flag                                |
| <code>SDIO_FLAG_CMDTMO</code><br><i>UT</i> | command response timeout flag   |
| <code>SDIO_FLAG_DTTMOU</code><br><i>T</i>  | data timeout flag   |
| <code>SDIO_FLAG_TXURE</code>               | transmit FIFO underrun error occurs flag  |
| <code>SDIO_FLAG_RXORE</code>               | received FIFO overrun error occurs flag   |
| <code>SDIO_FLAG_CMDREC</code><br><i>V</i>  | command response received (CRC check passed) flag                               |
| <code>SDIO_FLAG_CMDSEN</code><br><i>D</i>  | command sent (no response required) flag  |
| <code>SDIO_FLAG_DTEND</code>               | data end (data counter, SDIO_DATACNT, is zero) flag                             |
| <code>SDIO_FLAG_STBITE</code>              | start bit error in the bus flag   |
| <code>SDIO_FLAG_DTBKE</code><br><i>ND</i>  | data block sent/received (CRC check passed) flag                                |
| <code>SDIO_FLAG_CMDRUN</code>              | command transmission in progress flag   |
| <code>SDIO_FLAG_TXRUN</code>               | data transmission in progress flag  |
| <code>SDIO_FLAG_RXRUN</code>               | data reception in progress flag   |
| <code>SDIO_FLAG_TFH</code>                 | transmit FIFO is half empty flag: at least 8 words can be written into the FIFO |
| <code>SDIO_FLAG_RFH</code>                 | receive FIFO is half full flag: at least 8 words can be read in the FIFO        |
| <code>SDIO_FLAG_TFF</code>                 | transmit FIFO is full flag  |
| <code>SDIO_FLAG_RFF</code>                 | receive FIFO is full flag   |
| <code>SDIO_FLAG_TFE</code>                 | transmit FIFO is empty flag   |
| <code>SDIO_FLAG_RFE</code>                 | receive FIFO is empty flag  |
| <code>SDIO_FLAG_TXDTVAL</code>             | data is valid in transmit FIFO flag   |
| <code>SDIO_FLAG_RXDTVA</code><br><i>L</i>  | data is valid in receive FIFO flag  |

|                                |   |
|--------------------------------|---|
| <code>SDIO_FLAG_SDIOINT</code> | SD I/O interrupt received flag                                  |
| <code>SDIO_FLAG_ATAEND</code>  | CE-ATA command completion signal received (only for CMD61) flag |
| <b>Output parameter{out}</b>   |   |
| -                              | -   |
| <b>Return value</b>            |   |
| <code>FlagStatus</code>        | SET or RESET  |

Example:

```
/* get the flags state of SDIO */

FlagStatus flag_value;

flag_value = sdio_flag_get(SDIO_FLAG_RFH);
```

### **sdio\_flag\_clear**

The description of `sdio_flag_clear` is shown as below:

**Table 3-685. Function `sdio_flag_clear`**

|  |   |
|--|---|
| <b>Function name</b>                       | <code>sdio_flag_clear</code>                        |
| <b>Function prototype</b>                  | <code>void sdio_flag_clear(uint32_t flag);</code>   |
| <b>Function descriptions</b>               | clear the pending flags of SDIO                     |
| <b>Precondition</b>                        | -   |
| <b>The called functions</b>                | -   |
| <b>Input parameter{in}</b>                 |   |
| <code>flag</code>                          | flags state of SDIO                                 |
| <code>SDIO_FLAG_CCRCER</code><br><i>R</i>  | command response received (CRC check failed) flag   |
| <code>SDIO_FLAG_DTCRCE</code><br><i>RR</i> | data block sent/received (CRC check failed) flag    |
| <code>SDIO_FLAG_CMDTMO</code><br><i>UT</i> | command response timeout flag                       |
| <code>SDIO_FLAG_DTTMOU</code><br><i>T</i>  | data timeout flag                                   |
| <code>SDIO_FLAG_TXURE</code>               | transmit FIFO underrun error occurs flag            |
| <code>SDIO_FLAG_RXORE</code>               | received FIFO overrun error occurs flag             |
| <code>SDIO_FLAG_CMDREC</code><br><i>V</i>  | command response received (CRC check passed) flag   |
| <code>SDIO_FLAG_CMDSEN</code><br><i>D</i>  | command sent (no response required) flag            |
| <code>SDIO_FLAG_DTEND</code>               | data end (data counter, SDIO_DATACNT, is zero) flag |
| <code>SDIO_FLAG_STBITE</code>              | start bit error in the bus flag                     |
| <code>SDIO_FLAG_DTBLKE</code><br><i>ND</i> | data block sent/received (CRC check passed) flag    |
| <code>SDIO_FLAG_SDIOINT</code>             | SD I/O interrupt received flag                      |

|                               |   |
|-------------------------------|---|
| <code>SDIO_FLAG_ATAEND</code> | CE-ATA command completion signal received (only for CMD61) flag |
| <b>Output parameter{out}</b>  |   |
| -                             | -   |
| <b>Return value</b>           |   |
| -                             | -   |

Example:

```
/* clear the pending flags of SDIO */
sdio_flag_clear(SDIO_FLAG_DTCRCERR);
```

### **sdio\_interrupt\_enable**

The description of `sdio_interrupt_enable` is shown as below:

**Table 3-686. Function `sdio_interrupt_enable`**

|   |   |
|---|---|
| <b>Function name</b>                      | <code>sdio_interrupt_enable</code>                          |
| <b>Function prototype</b>                 | <code>void sdio_interrupt_enable(uint32_t int_flag);</code> |
| <b>Function descriptions</b>              | enable the SDIO interrupt                                   |
| <b>Precondition</b>                       | -   |
| <b>The called functions</b>               | -   |
| <b>Input parameter{in}</b>                |   |
| <code>int_flag</code>                     | interrupt flags state of SDIO                               |
| <code>SDIO_INT_CCRCERR</code>             | SDIO CCRCERR interrupt                                      |
| <code>SDIO_INT_DTCRCER</code><br><i>R</i> | SDIO DTCRCERR interrupt                                     |
| <code>SDIO_INT_CMDTMOU</code><br><i>T</i> | SDIO CMDTMOU interrupt                                      |
| <code>SDIO_INT_DTTMOUT</code>             | SDIO DTTMOUT interrupt                                      |
| <code>SDIO_INT_TXURE</code>               | SDIO TXURE interrupt  |
| <code>SDIO_INT_RXORE</code>               | SDIO INT_RXORE  |
| <code>SDIO_INT_CMDRECV</code>             | SDIO CMDRECV interrupt                                      |
| <code>SDIO_INT_CMDSEND</code>             | SDIO CMDSEND interrupt                                      |
| <code>SDIO_INT_DTEND</code>               | SDIO DTEND interrupt  |
| <code>SDIO_INT_STBITE</code>              | SDIO STBITE interrupt                                       |
| <code>SDIO_INT_DTBLKEND</code>            | SDIO DTBLKEND interrupt                                     |
| <code>SDIO_INT_CMDRUN</code>              | SDIO CMDRUN interrupt                                       |
| <code>SDIO_INT_TXRUN</code>               | SDIO TXRUN interrupt  |
| <code>SDIO_INT_RXRUN</code>               | SDIO RXRUN interrupt  |
| <code>SDIO_INT_TFH</code>                 | SDIO TFH interrupt  |
| <code>SDIO_INT_RFH</code>                 | SDIO RFH interrupt  |
| <code>SDIO_INT_TFF</code>                 | SDIO TFF interrupt  |
| <code>SDIO_INT_RFF</code>                 | SDIO RFF interrupt  |
| <code>SDIO_INT_TFE</code>                 | SDIO TFE interrupt  |

|                              |                        |
|------------------------------|------------------------|
| <i>SDIO_INT_RFE</i>          | SDIO RFE interrupt     |
| <i>SDIO_INT_TXDTVAL</i>      | SDIO TXDTVAL interrupt |
| <i>SDIO_INT_RXDTVAL</i>      | SDIO RXDTVAL interrupt |
| <i>SDIO_INT_SDIOINT</i>      | SDIO SDIOINT interrupt |
| <i>SDIO_INT_ATAEND</i>       | SDIO ATAEND interrupt  |
| <b>Output parameter{out}</b> |                        |
| -                            | -                      |
| <b>Return value</b>          |                        |
| -                            | -                      |

Example:

```
/* enable the SDIO corresponding interrupts */

sdio_interrupt_enable(SDIO_INT_CCRCERR | SDIO_INT_DTTMOUT | SDIO_INT_RXORE
| SDIO_INT_DTEND | SDIO_INT_STBITE);
```

### **sdio\_interrupt\_disable**

The description of `sdio_interrupt_disable` is shown as below:

**Table 3-687. Function `sdio_interrupt_disable`**

|                                     |  |
|-------------------------------------|--|
| <b>Function name</b>                | <code>sdio_interrupt_disable</code>                          |
| <b>Function prototype</b>           | <code>void sdio_interrupt_disable(uint32_t int_flag);</code> |
| <b>Function descriptions</b>        | disable the SDIO interrupt                                   |
| <b>Precondition</b>                 | -  |
| <b>The called functions</b>         | -  |
| <b>Input parameter{in}</b>          |  |
| <b>int_flag</b>                     | interrupt flags state of SDIO                                |
| <i>SDIO_INT_CCRCERR</i>             | SDIO CCRCERR interrupt                                       |
| <i>SDIO_INT_DTCRCER</i><br><i>R</i> | SDIO DTCRCERR interrupt                                      |
| <i>SDIO_INT_CMDTMOU</i><br><i>T</i> | SDIO CMDTMOU interrupt                                       |
| <i>SDIO_INT_DTTMOUT</i>             | SDIO DTTMOUT interrupt                                       |
| <i>SDIO_INT_TXURE</i>               | SDIO TXURE interrupt   |
| <i>SDIO_INT_RXORE</i>               | SDIO_RXORE   |
| <i>SDIO_INT_CMDRECV</i>             | SDIO CMDRECV interrupt                                       |
| <i>SDIO_INT_CMDSEND</i>             | SDIO CMDSEND interrupt                                       |
| <i>SDIO_INT_DTEND</i>               | SDIO DTEND interrupt   |
| <i>SDIO_INT_STBITE</i>              | SDIO STBITE interrupt  |
| <i>SDIO_INT_DTBLKEND</i>            | SDIO DTBLKEND interrupt                                      |
| <i>SDIO_INT_CMDRUN</i>              | SDIO CMDRUN interrupt  |
| <i>SDIO_INT_TXRUN</i>               | SDIO TXRUN interrupt   |
| <i>SDIO_INT_RXRUN</i>               | SDIO RXRUN interrupt   |

|                              |                        |
|------------------------------|------------------------|
| <i>SDIO_INT_TFH</i>          | SDIO TFH interrupt     |
| <i>SDIO_INT_RFH</i>          | SDIO RFH interrupt     |
| <i>SDIO_INT_TFF</i>          | SDIO TFF interrupt     |
| <i>SDIO_INT_RFF</i>          | SDIO RFF interrupt     |
| <i>SDIO_INT_TFE</i>          | SDIO TFE interrupt     |
| <i>SDIO_INT_RFE</i>          | SDIO RFE interrupt     |
| <i>SDIO_INT_TXDTVAL</i>      | SDIO TXDTVAL interrupt |
| <i>SDIO_INT_RXDTVAL</i>      | SDIO RXDTVAL interrupt |
| <i>SDIO_INT_SDIOINT</i>      | SDIO SDIOINT interrupt |
| <i>SDIO_INT_ATAEND</i>       | SDIO ATAEND interrupt  |
| <b>Output parameter{out}</b> |                        |
| -                            | -                      |
| <b>Return value</b>          |                        |
| -                            | -                      |

Example:

```
/* disable the SDIO interrupt */
sdio_interrupt_disable(SDIO_INT_DTCRCERR);
```

### **sdio\_interrupt\_flag\_get**

The description of `sdio_interrupt_flag_get` is shown as below:

**Table 3-688. Function `sdio_interrupt_flag_get`**

|                                    |   |
|------------------------------------|---|
| <b>Function name</b>               | <code>sdio_interrupt_flag_get</code>                                |
| <b>Function prototype</b>          | <code>FlagStatus sdio_interrupt_flag_get(uint32_t int_flag);</code> |
| <b>Function descriptions</b>       | get the interrupt flags state of SDIO                               |
| <b>Precondition</b>                | -   |
| <b>The called functions</b>        | -   |
| <b>Input parameter{in}</b>         |   |
| <b>int_flag</b>                    | interrupt flags state of SDIO                                       |
| <i>SDIO_INT_FLAG_CCR<br/>CERR</i>  | SDIO CCRCERR interrupt  |
| <i>SDIO_INT_FLAG_DTC<br/>RCERR</i> | SDIO DTCRCERR interrupt   |
| <i>SDIO_INT_FLAG_CMD<br/>TMOUT</i> | SDIO CMDTMOOUT interrupt  |
| <i>SDIO_INT_FLAG_DTT<br/>MOUT</i>  | SDIO DTTMOUT interrupt  |
| <i>SDIO_INT_FLAG_TXU<br/>RE</i>    | SDIO TXURE interrupt  |
| <i>SDIO_INT_FLAG_RXO<br/>RE</i>    | SDIO_INT_RXORE  |

|                                     |                         |
|-------------------------------------|-------------------------|
| <i>SDIO_INT_FLAG_CMD_RECV</i>       | SDIO CMDRECV interrupt  |
| <i>SDIO_INT_FLAG_CMD_SEND</i>       | SDIO CMDSEND interrupt  |
| <i>SDIO_INT_FLAG_DTE_ND</i>         | SDIO DTEND interrupt    |
| <i>SDIO_INT_FLAG_STBITE_TE</i>      | SDIO STBITE interrupt   |
| <i>SDIO_INT_FLAG_DTBLKEND_LKEND</i> | SDIO DTBLKEND interrupt |
| <i>SDIO_INT_FLAG_CMD_RUN</i>        | SDIO CMDRUN interrupt   |
| <i>SDIO_INT_FLAG_TXRUN_UN</i>       | SDIO TXRUN interrupt    |
| <i>SDIO_INT_FLAG_RXRUN_UN</i>       | SDIO RXRUN interrupt    |
| <i>SDIO_INT_FLAG_TFH</i>            | SDIO TFH interrupt      |
| <i>SDIO_INT_FLAG_RFH</i>            | SDIO RFH interrupt      |
| <i>SDIO_INT_FLAG_TFF</i>            | SDIO TFF interrupt      |
| <i>SDIO_INT_FLAG_RFF</i>            | SDIO RFF interrupt      |
| <i>SDIO_INT_FLAG_TFE</i>            | SDIO TFE interrupt      |
| <i>SDIO_INT_FLAG_RFE</i>            | SDIO RFE interrupt      |
| <i>SDIO_INT_FLAG_TXDTVAL_TVAL</i>   | SDIO TXDTVAL interrupt  |
| <i>SDIO_INT_FLAG_RXDTVAL_TVAL</i>   | SDIO RXDTVAL interrupt  |
| <i>SDIO_INT_FLAG_SDIOINT_OINT</i>   | SDIO SDIOINT interrupt  |
| <i>SDIO_INT_FLAG_ATAEND_END</i>     | SDIO ATAEND interrupt   |
| <b>Output parameter{out}</b>        |                         |
| -                                   | -                       |
| <b>Return value</b>                 |                         |
| <b>FlagStatus</b>                   | SET or RESET            |

Example:

```

/* get the interrupt flags state of SDIO */

FlagStatus flag_value;

flag_value = sdio_interrupt_flag_get(SDIO_INT_FLAG_DTEND);

```

### sdio\_interrupt\_flag\_clear

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

**Table 3-689. Function sdio\_interrupt\_flag\_clear**

|                                    |  |
|------------------------------------|--|
| <b>Function name</b>               | sdio_interrupt_flag_clear                          |
| <b>Function prototype</b>          | void sdio_interrupt_flag_clear(uint32_t int_flag); |
| <b>Function descriptions</b>       | clear the interrupt pending flags of SDIO          |
| <b>Precondition</b>                | -  |
| <b>The called functions</b>        | -  |
| <b>Input parameter{in}</b>         |  |
| <b>int_flag</b>                    | interrupt flags state of SDIO                      |
| <i>SDIO_INT_FLAG_CCR<br/>CERR</i>  | SDIO CCRCERR interrupt                             |
| <i>SDIO_INT_FLAG_DTC<br/>RCERR</i> | SDIO DTCRCERR interrupt                            |
| <i>SDIO_INT_FLAG_CMD<br/>TMOUT</i> | SDIO CMDTMOUT interrupt                            |
| <i>SDIO_INT_FLAG_DTT<br/>MOUT</i>  | SDIO DTTMOUT interrupt                             |
| <i>SDIO_INT_FLAG_TXU<br/>RE</i>    | SDIO TXURE interrupt                               |
| <i>SDIO_INT_FLAG_RXO<br/>RE</i>    | SDIO_INT_RXORE                                     |
| <i>SDIO_INT_FLAG_CMD<br/>RECV</i>  | SDIO CMDRECV interrupt                             |
| <i>SDIO_INT_FLAG_CMD<br/>SEND</i>  | SDIO CMDSEND interrupt                             |
| <i>SDIO_INT_FLAG_DTE<br/>ND</i>    | SDIO DTEND interrupt                               |
| <i>SDIO_INT_FLAG_STB<br/>TE</i>    | SDIO STBITE interrupt                              |
| <i>SDIO_INT_FLAG_DTB<br/>LKEND</i> | SDIO DTBLKEND interrupt                            |
| <i>SDIO_INT_FLAG_SDI<br/>OINT</i>  | SDIO SDIOINT interrupt                             |
| <i>SDIO_INT_FLAG_ATA<br/>END</i>   | SDIO ATAEND interrupt                              |
| <b>Output parameter{out}</b>       |  |
| -                                  | -  |
| <b>Return value</b>                |  |
| -                                  | -  |

Example:

---

```

/* clear the interrupt pending flags of SDIO */

sdio_interrupt_flag_clear(SDIO_INT_FLAG_DTEND);

```

## 3.24. 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 [3.24.1](#), the SPI/I2S firmware functions are introduced in chapter [3.24.2](#).

### 3.24.1. Descriptions of Peripheral registers

SPI/I2S registers are listed in the table shown as below:

**Table 3-690. 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   |

### 3.24.2. Descriptions of Peripheral functions

SPI/I2S firmware functions are listed in the table shown as below:

**Table 3-691. SPI/I2S firmware function**

| Function name         | Function description   |
|-----------------------|--|
| spi_i2s_deinit        | reset SPI and I2S  |
| spi_struct_para_init  | initialize the parameters of SPI structure with the default values |
| spi_init              | initialize SPI parameters  |
| spi_enable            | enable SPI   |
| spi_disable           | disable SPI  |
| i2s_init              | initialize I2S parameters  |
| i2s_psc_config        | configure I2S prescaler  |
| i2s_enable            | enable I2S   |
| i2s_disable           | disable I2S  |
| spi_nss_output_enable | enable SPI NSS output  |

| <b>Function name</b>              | <b>Function description</b>                      |
|-----------------------------------|--|
| spi_nss_output_disable            | disable SPI NSS output                           |
| 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 send or receive                   |
| spi_dma_disable                   | disable SPI DMA send or receive                  |
| spi_i2s_data_frame_format_config  | configure SPI data frame format                  |
| spi_bidirectional_transfer_config | configure SPI bidirectional transfer direction   |
| spi_i2s_data_transmit             | SPI transmit data                                |
| spi_i2s_data_receive              | SPI receive data                                 |
| 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_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_flag_get                  | get SPI and I2S flag status                      |
| spi_i2s_interrupt_enable          | enable SPI and I2S interrupt                     |
| spi_i2s_interrupt_disable         | disable SPI and I2S interrupt                    |
| spi_i2s_interrupt_flag_get        | get SPI and I2S interrupt status                 |
| spi_crc_error_clear               | clear SPI CRC error flag status                  |

### Structure spi\_parameter\_struct

**Table 3-692. spi\_parameter\_struct**

| <b>Member name</b> | <b>Function description</b>  |
|--------------------|--|
| device_mode        | SPI master or slave<br>(SPI_MASTER, SPI_SLAVE)   |
| trans_mode         | SPI transfer type<br>(SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY,<br>SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT) |
| frame_size         | SPI frame size<br>(SPI_FRAMESIZE_16BIT, SPI_FRAMESIZE_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_e | SPI clock phase and polarity<br>(SPI_CK_PL_LOW_PH_1EDGE,<br>SPI_CK_PL_HIGH_PH_1EDGE SPI_CK_PL_LOW_PH_2EDGE,<br>SPI_CK_PL_HIGH_PH_2EDGE) |
| prescale               | SPI prescaler 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 3-693. 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                                  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | rcu_periph_reset_enable / rcu_periph_reset_disable |
| <b>Input parameter{in}</b>   |  |
| spi_periph                   | SPI/I2S peripheral                                 |
| SPIx(x=0,1,2)                | SPI/I2S peripheral selection                       |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* reset SPI0 */
spi_i2s_deinit(SPI0);
```

### spi\_struct\_para\_init

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

**Table 3-694. 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 structure with the default values   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| spi_struct                   | SPI parameter struct, the structure members can refer to members of the structure <a href="#">错误!未找到引用源。</a> |

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

Example:

```
/* initialize the parameters of SPI */
spi_parameter_struct spi_init_struct;
spi_struct_para_init(&spi_init_struct);
```

### spi\_init

The description of spi\_init is shown as below:

**Table 3-695. 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 parameter   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| spi_periph                   | SPI peripheral   |
| SPIx(x=0,1,2)                | SPI peripheral selection   |
| <b>Input parameter{in}</b>   |  |
| spi_struct                   | SPI parameter initialization structure, the structure members can refer to members of the structure <a href="#">错误!未找到引用源。</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* initialize SPI0 */
spi_parameter_struct spi_init_struct;
spi_init_struct.trans_mode = SPI_TRANSMODE_BDTRANSMIT;
spi_init_struct.device_mode = SPI_MASTER;
spi_init_struct.frame_size = SPI_FRAMESIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_CK_PL_HIGH_PH_2EDGE;
spi_init_struct.nss = SPI_NSS_SOFT;
spi_init_struct.prescale = SPI_PSC_8;
spi_init_struct.endian = SPI_ENDIAN_MSB;
```

```
spi_init(SPI0, &spi_init_struct);
```

### spi\_enable

The description of spi\_enable is shown as below:

**Table 3-696. 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>   |                                       |
| spi_periph                   | SPI peripheral                        |
| SPIx(x=0,1,2)                | SPI peripheral selection              |
| <b>Output parameter{out}</b> |                                       |
| -                            | -                                     |
| <b>Return value</b>          |                                       |
| -                            | -                                     |

Example:

```
/* enable SPI0 */
spi_enable(SPI0);
```

### spi\_disable

The description of spi\_disable is shown as below:

**Table 3-697. 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>   |  |
| spi_periph                   | SPI peripheral                         |
| SPIx(x=0,1,2)                | SPI peripheral selection               |
| <b>Output parameter{out}</b> |  |
| -                            | -                                      |
| <b>Return value</b>          |  |
| -                            | -                                      |

Example:

```
/* disable SPI0 */
```

```
spi_disable(SPI0);
```

### i2s\_init

The description of i2s\_init is shown as below:

**Table 3-698. 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 parameter  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>spi_periph</b>            | I2S peripheral  |
| <i>SPIx(x=1,2)</i>           | I2S peripheral selection  |
| <b>Input parameter{in}</b>   |   |
| <b>mode</b>                  | I2S operation mode  |
| <i>I2S_MODE_SLAVETX</i>      | I2S slave transmit mode   |
| <i>I2S_MODE_SLAVERX</i>      | I2S slave receive mode  |
| <i>I2S_MODE_MASTERTX</i>     | I2S master transmit mode  |
| <i>I2S_MODE_MASTERRX</i>     | I2S master receive mode   |
| <b>Input parameter{in}</b>   |   |
| <b>standard</b>              | I2S standard  |
| <i>I2S_STD_PHILLIPS</i>      | I2S phillips standard   |
| <i>I2S_STD_MSB</i>           | I2S MSB standard  |
| <i>I2S_STD_LSB</i>           | I2S LSB standard  |
| <i>I2S_STD_PCMSHORT</i>      | I2S PCM short standard  |
| <i>I2S_STD_PCMLONG</i>       | I2S PCM long standard   |
| <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_PHILLIPS, I2S_CKPL_LOW);
```

## i2s\_psc\_config

The description of i2s\_psc\_config is shown as below:

**Table 3-699. 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>   |  |
| spi_periph                   | I2S peripheral   |
| SPIx(x=1,2)                  | I2S peripheral selection   |
| <b>Input parameter{in}</b>   |  |
| audiosample                  | I2S audio sample rate  |
| I2S_AUDIOSAMPLE_8K           | audio sample rate is 8KHz  |
| I2S_AUDIOSAMPLE_11K          | audio sample rate is 11KHz   |
| I2S_AUDIOSAMPLE_16K          | audio sample rate is 16KHz   |
| I2S_AUDIOSAMPLE_22K          | audio sample rate is 22KHz   |
| I2S_AUDIOSAMPLE_32K          | audio sample rate is 32KHz   |
| I2S_AUDIOSAMPLE_44K          | audio sample rate is 44KHz   |
| I2S_AUDIOSAMPLE_48K          | audio sample rate is 48KHz   |
| I2S_AUDIOSAMPLE_96K          | audio sample rate is 96KHz   |
| I2S_AUDIOSAMPLE_192K         | audio sample rate is 192KHz  |
| <b>Input parameter{in}</b>   |  |
| frameformat                  | I2S data length and channel length   |
| I2S_FRAMEFORMAT_DT16B_CH16B  | I2S data length is 16 bit and channel length is 16 bit   |
| I2S_FRAMEFORMAT_DT16B_CH32B  | I2S data length is 16 bit and channel length is 32 bit   |
| I2S_FRAMEFORMAT_DT24B_CH32B  | I2S data length is 24 bit and channel length is 32 bit   |
| I2S_FRAMEFORMAT_DT32B_CH32B  | I2S data length is 32 bit and channel length is 32 bit   |

| Input parameter{in}     |                                 |
|-------------------------|---------------------------------|
| mckout                  | I2S master clock output         |
| I2S_MCKOUT_ENABL<br>E   | I2S master clock output enable  |
| I2S_MCKOUT_DISABLE<br>E | I2S master clock output disable |
| Output parameter{out}   |                                 |
| -                       | -                               |
| Return value            |                                 |
| -                       | -                               |

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 3-700. 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>  | -                                     |
| Input parameter{in}          |                                       |
| spi_periph                   | I2S peripheral                        |
| SPIx(x=1,2)                  | I2S peripheral selection              |
| Output parameter{out}        |                                       |
| -                            | -                                     |
| Return value                 |                                       |
| -                            | -                                     |

Example:

```
/* enable I2S1*/

i2s_enable(SPI1);
```

### i2s\_disable

The description of i2s\_disable is shown as below:

**Table 3-701. Function i2s\_disable**

|                      |             |
|----------------------|-------------|
| <b>Function name</b> | i2s_disable |
|----------------------|-------------|

|                              |  |
|------------------------------|--|
| <b>Function prototype</b>    | void i2s_disable(uint32_t spi_periph); |
| <b>Function descriptions</b> | disable I2S                            |
| <b>Precondition</b>          | -                                      |
| <b>The called functions</b>  | -                                      |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | I2S peripheral                         |
| <b>SPIx(x=1,2)</b>           | I2S peripheral selection               |
| <b>Output parameter{out}</b> |  |
| -                            | -                                      |
| <b>Return value</b>          |  |
| -                            | -                                      |

Example:

```
/* disable I2S1*/
i2s_disable(SPI1);
```

### **spi\_nss\_output\_enable**

The description of `spi_nss_output_enable` is shown as below:

**Table 3-702. Function `spi_nss_output_enable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_nss_output_enable                            |
| <b>Function prototype</b>    | void spi_nss_output_enable(uint32_t spi_periph); |
| <b>Function descriptions</b> | enable SPI NSS output                            |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | SPI peripheral                                   |
| <b>SPIx(x=0,1,2)</b>         | SPI peripheral selection                         |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable SPI0 NSS output */
spi_nss_output_enable(SPI0);
```

### **spi\_nss\_output\_disable**

The description of `spi_nss_output_disable` is shown as below:

**Table 3-703. 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                            |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>spi_periph</b>            | SPI peripheral                                    |
| <b>SPIx(x=0,1,2)</b>         | SPI peripheral selection                          |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable SPI0 NSS output */

spi_nss_output_disable(SPI0);
```

### **spi\_nss\_internal\_high**

The description of **spi\_nss\_internal\_high** is shown as below:

**Table 3-704. 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                                   |
| <b>SPIx(x=0,1,2)</b>         | SPI peripheral selection                         |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* SPI0 NSS pin is pulled high level in software mode */

spi_nss_internal_high(SPI0);
```

### **spi\_nss\_internal\_low**

The description of **spi\_nss\_internal\_low** is shown as below:

**Table 3-705. 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                                  |
| <b>SPIx(x=0,1,2)</b>         | SPI peripheral selection                        |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* SPI0 NSS pin is pulled low level in software mode */

spi_nss_internal_low(SPI0);
```

### **spi\_dma\_enable**

The description of **spi\_dma\_enable** is shown as below:

**Table 3-706. 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 send or receive                         |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | SPI peripheral   |
| <b>SPIx(x=0,1,2)</b>         | SPI peripheral selection                               |
| <b>Input parameter{in}</b>   |  |
| <b>dma</b>                   | SPI DMA mode   |
| <b>SPI_DMA_TRANSMIT</b>      | SPI transmit data use DMA                              |
| <b>SPI_DMA_RECEIVE</b>       | 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 3-707. 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 send or receive                         |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| spi_periph                   | SPI peripheral  |
| SPIx(x=0,1,2)                | SPI peripheral selection                                |
| <b>Input parameter{in}</b>   |   |
| dma                          | SPI DMA mode  |
| SPI_DMA_TRANSMIT             | SPI transmit data use DMA                               |
| SPI_DMA_RECEIVE              | SPI receive data use DMA                                |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable SPI0 transmit data DMA function */

spi_dma_disable(SPI0, SPI_DMA_TRANSMIT);
```

### spi\_i2s\_data\_frame\_format\_config

The description of spi\_i2s\_data\_frame\_format\_config is shown as below:

**Table 3-708. Function spi\_i2s\_data\_frame\_format\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_i2s_data_frame_format_config   |
| <b>Function prototype</b>    | void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format); |
| <b>Function descriptions</b> | configure SPI data frame format  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| spi_periph                   | SPI peripheral   |
| SPIx(x=0,1,2)                | SPI peripheral selection   |
| <b>Input parameter{in}</b>   |  |
| frame_format                 | SPI frame size   |
| SPI_FRAMESIZE_16BIT          | SPI frame size is 16 bits  |

|                              |                          |
|------------------------------|--------------------------|
| SPI_FRAME_SIZE_8BIT          | 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\_bidirectional\_transfer\_config**

The description of spi\_bidirectional\_transfer\_config is shown as below:

**Table 3-709. Function spi\_bidirectional\_transfer\_config**

|                              |   |
|------------------------------|---|
| Function name                | spi_bidirectional_transfer_config   |
| Function prototype           | void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction); |
| Function descriptions        | configure SPI bidirectional transfer direction  |
| Precondition                 | -   |
| The called functions         | -   |
| <b>Input parameter{in}</b>   |   |
| spi_periph                   | SPI peripheral  |
| SPIx(x=0,1,2)                | SPI peripheral selection  |
| <b>Input parameter{in}</b>   |   |
| transfer_direction           | SPI transfer direction  |
| SPI_BIDIRECTIONAL_TRANSMIT   | SPI work in transmit-only mode  |
| SPI_BIDIRECTIONAL_RECEIVE    | 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\_i2s\_data\_transmit**

The description of spi\_i2s\_data\_transmit is shown as below:

Table 3-710. Function spi\_i2s\_data\_transmit

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | spi_i2s_data_transmit   |
| <b>Function prototype</b>    | void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data); |
| <b>Function descriptions</b> | SPI transmit data   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| spi_periph                   | SPI peripheral  |
| SPIx(x=0,1,2)                | SPI peripheral selection  |
| <b>Input parameter{in}</b>   |   |
| data                         | 16-bit data   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* SPI0 transmit data */

uint16_t spi0_send_array[10];

uint8_t send_n = 1;

spi_i2s_data_transmit(SPI0, spi0_send_array[send_n]);
```

### spi\_i2s\_data\_receive

The description of spi\_i2s\_data\_receive is shown as below:

Table 3-711. 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>   |   |
| spi_periph                   | SPI peripheral                                      |
| SPIx(x=0,1,2)                | SPI peripheral selection                            |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| uint16_t                     | 16-bit data   |

Example:

```
/* SPI0 receive data */
```

```

uint16_t spi0_receive_array[10];

uint8_t receive_n = 1;

spi0_receive_array[receive_n] = spi_i2s_data_receive(SPI0);
  
```

### **spi\_crc\_polynomial\_set**

The description of **spi\_crc\_polynomial\_set** is shown as below:

**Table 3-712. Function spi\_crc\_polynomial\_set**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_crc_polynomial_set   |
| <b>Function prototype</b>    | void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly); |
| <b>Function descriptions</b> | set SPI CRC polynomial   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| spi_periph                   | SPI peripheral   |
| SPIx(x=0,1,2)                | SPI peripheral selection   |
| <b>Input parameter{in}</b>   |  |
| crc_poly                     | CRC polynomial value   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```

/* set SPI0 CRC polynomial */

uint16_t CRC_VALUE = 0x8;

spi_crc_polynomial_set(SPI0,CRC_VALUE);
  
```

### **spi\_crc\_polynomial\_get**

The description of **spi\_crc\_polynomial\_get** is shown as below:

**Table 3-713. Function spi\_crc\_polynomial\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | spi_crc_polynomial_get                                |
| <b>Function prototype</b>    | uint16_t spi_crc_polynomial_get(uint32_t spi_periph); |
| <b>Function descriptions</b> | get SPI CRC polynomial                                |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| spi_periph                   | SPI peripheral  |
| SPIx(x=0,1,2)                | SPI peripheral selection                              |
| <b>Output parameter{out}</b> |   |

|                     |                                  |
|---------------------|----------------------------------|
| -                   | -                                |
| <b>Return value</b> |                                  |
| <b>uint16_t</b>     | 16-bit CRC polynomial (0-0xFFFF) |

Example:

```
/* get SPI0 CRC polynomial */

uint16_t CRC_VALUE;
CRC_VALUE = spi_crc_polynomial_get(SPI0);
```

### spi\_crc\_on

The description of spi\_crc\_on is shown as below:

**Table 3-714. Function spi\_crc\_on**

|                              |                                       |
|------------------------------|---------------------------------------|
| <b>Function name</b>         | spi_crc_on                            |
| <b>Function prototype</b>    | void spi_crc_on(uint32_t spi_periph); |
| <b>Function descriptions</b> | turn on CRC function                  |
| <b>Precondition</b>          | -                                     |
| <b>The called functions</b>  | -                                     |
| <b>Input parameter{in}</b>   |                                       |
| <b>spi_periph</b>            | SPI peripheral                        |
| SPIx(x=0,1,2)                | SPI peripheral selection              |
| <b>Output parameter{out}</b> |                                       |
| -                            | -                                     |
| <b>Return value</b>          |                                       |
| -                            | -                                     |

Example:

```
/* turn on SPI0 CRC function */

spi_crc_on(SPI0);
```

### spi\_crc\_off

The description of spi\_crc\_off is shown as below:

**Table 3-715. Function spi\_crc\_off**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_crc_off                            |
| <b>Function prototype</b>    | void spi_crc_off(uint32_t spi_periph); |
| <b>Function descriptions</b> | turn off CRC function                  |
| <b>Precondition</b>          | -                                      |
| <b>The called functions</b>  | -                                      |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | SPI peripheral                         |

|                              |                          |
|------------------------------|--------------------------|
| SPIx(x=0,1,2)                | SPI peripheral selection |
| <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 3-716. Function spi\_crc\_next**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | spi_crc_next                            |
| <b>Function prototype</b>    | void spi_crc_next(uint32_t spi_periph); |
| <b>Function descriptions</b> | SPI next data is CRC value              |
| <b>Precondition</b>          | -                                       |
| <b>The called functions</b>  | -                                       |
| <b>Input parameter{in}</b>   |   |
| spi_periph                   | SPI peripheral                          |
| SPIx(x=0,1,2)                | SPI peripheral selection                |
| <b>Output parameter{out}</b> |   |
| -                            | -                                       |
| <b>Return value</b>          |   |
| -                            | -                                       |

Example:

```
/* SPI0 next data is CRC value */

spi_crc_next(SPI0);
```

### **spi\_crc\_get**

The description of spi\_crc\_get is shown as below:

**Table 3-717. Function spi\_crc\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | spi_crc_get   |
| <b>Function prototype</b>    | uint16_t spi_crc_get(uint32_t spi_periph, uint8_t crc); |
| <b>Function descriptions</b> | get SPI CRC send value or receive value                 |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| spi_periph                   | SPI peripheral  |

|                              |                             |
|------------------------------|-----------------------------|
| <i>SPIx(x=0,1,2)</i>         | SPI peripheral selection    |
| <b>Input parameter{in}</b>   |                             |
| <b>crc</b>                   | SPI crc value               |
| <i>SPI_CRC_TX</i>            | get transmit crc value      |
| <i>SPI_CRC_RX</i>            | get receive crc value       |
| <b>Output parameter{out}</b> |                             |
| -                            | -                           |
| <b>Return value</b>          |                             |
| <b>uint16_t</b>              | 16-bit CRC value (0-0xFFFF) |

Example:

```
/* get SPI0 CRC send value */

uint16_t value;

value = spi_crc_get(SPI0, SPI_CRC_TX);
```

### spi\_quad\_enable

The description of `spi_quad_enable` is shown as below:

**Table 3-718. 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(x=0)</i>             | SPI peripheral selection                   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable quad wire SPI */

spi_quad_enable(SPI0);
```

### spi\_quad\_disable

The description of `spi_quad_disable` is shown as below:

**Table 3-719. Function `spi_quad_disable`**

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

|                              |   |
|------------------------------|---|
| <b>Function prototype</b>    | void spi_quad_disable(uint32_t spi_periph); |
| <b>Function descriptions</b> | disable quad wire SPI                       |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>spi_periph</b>            | SPI peripheral                              |
| <b>SPIx(x=0)</b>             | SPI peripheral selection                    |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable quad wire SPI */

spi_quad_disable(SPI0);
```

### spi\_quad\_write\_enable

The description of spi\_quad\_write\_enable is shown as below:

**Table 3-720. 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                                   |
| <b>SPIx(x=0)</b>             | SPI peripheral selection                         |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable quad wire SPI write */

spi_quad_write_enable(SPI0);
```

### spi\_quad\_read\_enable

The description of spi\_quad\_read\_enable is shown as below:

**Table 3-721. 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                                  |
| <b>SPIx(x=0)</b>             | SPI peripheral selection                        |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable quad wire SPI read */

spi_quad_read_enable(SPI0);
```

### **spi\_quad\_io23\_output\_enable**

The description of `spi_quad_io23_output_enable` is shown as below:

**Table 3-722. 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   |
| <b>SPIx(x=0)</b>             | SPI peripheral selection                               |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable 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 3-723. 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  |
| <b>SPIx</b>                  | x=0   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable SPI_IO2 and SPI_IO3 pin output */

spi_quad_io23_output_disable(SPI0);
```

### spi\_i2s\_flag\_get

The description of spi\_i2s\_flag\_get is shown as below:

**Table 3-724. Function spi\_i2s\_flag\_get**

|                                 |  |
|---------------------------------|--|
| <b>Function name</b>            | spi_i2s_flag_get   |
| <b>Function prototype</b>       | FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag); |
| <b>Function descriptions</b>    | get SPI and I2S flag status                                      |
| <b>Precondition</b>             | -  |
| <b>The called functions</b>     | -  |
| <b>Input parameter{in}</b>      |  |
| <b>spi_periph</b>               | SPI/I2S peripheral   |
| <b>SPIx(x=0,1,2)</b>            | SPI/I2S peripheral selection                                     |
| <b>Input parameter{in}</b>      |  |
| <b>flag</b>                     | SPI/I2S flag status  |
| <b>SPI_FLAG_TBE</b>             | transmit buffer empty flag                                       |
| <b>SPI_FLAG_RBNE</b>            | receive buffer not empty flag                                    |
| <b>SPI_FLAG_TRANS</b>           | transmit on-going flag   |
| <b>SPI_I2S_INT_FLAG_RXORERR</b> | receive overrun error flag                                       |
| <b>SPI_FLAG_CONFERR</b>         | mode config error flag   |
| <b>SPI_FLAG_CRCERR</b>          | CRC error flag   |
| <b>I2S_FLAG_TBE</b>             | transmit buffer empty flag                                       |
| <b>I2S_FLAG_RBNE</b>            | receive buffer not empty flag                                    |
| <b>I2S_FLAG_TRANS</b>           | transmit on-going flag   |
| <b>I2S_FLAG_RXORERR</b>         | overrun error flag   |
| <b>I2S_FLAG_TXURERR</b>         | underrun error flag  |

|                              |                   |
|------------------------------|-------------------|
| <i>I2S_FLAG_CH</i>           | channel side flag |
| <b>Output parameter{out}</b> |                   |
| -                            | -                 |
| <b>Return value</b>          |                   |
| <b>FlagStatus</b>            | SET or RESET      |

Example:

```
/* get SPI0 transmit buffer empty flag status */
```

```
FlagStatus Flag = RESET;
```

```
Flag = spi_i2s_flag_get(SPI0, SPI_FLAG_TBE);
```

### **spi\_i2s\_interrupt\_enable**

The description of **spi\_i2s\_interrupt\_enable** is shown as below:

**Table 3-725. 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/I2S peripheral  |
| <b>SPI/x(x=0,1,2)</b>        | SPI/I2S peripheral selection  |
| <b>Input parameter{in}</b>   |   |
| <b>interrupt</b>             | SPI/I2S interrupt   |
| <b>SPI_I2S_INT_TBE</b>       | transmit buffer empty interrupt   |
| <b>SPI_I2S_INT_RBNE</b>      | receive buffer not empty interrupt  |
| <b>SPI_I2S_INT_ERR</b>       | CRC error, configuration error, reception overrun error, transmission underrun error and format error interrupt |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable SPI0 transmit buffer empty interrupt */
```

```
spi_i2s_interrupt_enable(SPI0, SPI_I2S_INT_TBE);
```

### **spi\_i2s\_interrupt\_disable**

The description of **spi\_i2s\_interrupt\_disable** is shown as below:

**Table 3-726. 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>   |   |
| spi_periph                   | SPI/I2S peripheral  |
| SPIx(x=0,1,2)                | SPI/I2S peripheral selection  |
| <b>Input parameter{in}</b>   |   |
| interrupt                    | SPI/I2S interrupt   |
| SPI_I2S_INT_TBE              | transmit buffer empty interrupt   |
| SPI_I2S_INT_RBNE             | receive buffer not empty interrupt  |
| SPI_I2S_INT_ERR              | CRC error, configuration error, reception overrun error, transmission underrun error and format error interrupt |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable SPI0 transmit buffer empty interrupt */
spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_TBE);
```

### spi\_i2s\_interrupt\_flag\_get

The description of spi\_i2s\_interrupt\_flag\_get is shown as below:

**Table 3-727. Function spi\_i2s\_interrupt\_flag\_get**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_i2s_interrupt_flag_get   |
| <b>Function prototype</b>    | FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt); |
| <b>Function descriptions</b> | get SPI and I2S interrupt status   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| spi_periph                   | SPI/I2S peripheral   |
| SPIx(x=0,1,2)                | SPI/I2S peripheral selection   |
| <b>Input parameter{in}</b>   |  |
| interrupt                    | SPI/I2S interrupt flag status  |
| SPI_I2S_INT_FLAG_TBE         | transmit buffer empty interrupt  |
| SPI_I2S_INT_FLAG_RBN         | receive buffer not empty interrupt   |

|                              |                          |
|------------------------------|--------------------------|
| <i>SPI_I2S_INT_FLAG_R</i>    | overrun interrupt        |
| <i>SPI_INT_FLAG_CONF</i>     | config error interrupt   |
| <i>SPI_INT_FLAG_CRCERR</i>   | CRC error interrupt      |
| <i>I2S_INT_FLAG_TXUR</i>     | underrun error interrupt |
| <b>Output parameter{out}</b> |                          |
| -                            | -                        |
| <b>Return value</b>          |                          |
| <b>FlagStatus</b>            | SET or RESET             |

Example:

```
/* get SPI0 transmit buffer empty interrupt status */

FlagStatus Flag_interrupt = RESET;

Flag_interrupt = spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE);
```

### spi\_crc\_error\_clear

The description of spi\_crc\_error\_clear is shown as below:

**Table 3-728. Function spi\_crc\_error\_clear**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | spi_crc_error_clear                            |
| <b>Function prototype</b>    | void spi_crc_error_clear(uint32_t spi_periph); |
| <b>Function descriptions</b> | clear SPI CRC error flag status                |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>spi_periph</b>            | SPI peripheral                                 |
| <i>SPIx(x=0,1,2)</i>         | SPI peripheral selection                       |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* clear SPI0 CRC error flag status */

spi_crc_error_clear(SPI0);
```

## 3.25. TIMER

The timers have a 16-bit counter that can be used as an unsigned counter and supports both input capture and output compare. Timers (TIMERx) are divided into five sorts: advanced timer (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), and basic timer (TIMERx, x=5, 6). The specific functions of different types of timer are different. The TIMER registers are listed in chapter [3.25.1](#), the TIMER firmware functions are introduced in chapter [3.25.2](#).

### 3.25.1. Descriptions of Peripheral registers

TIMERx registers are listed in the table shown as below:

**Table 3-729. TIMERx Registers**

| Registers     | Descriptions                              |
|---------------|---|
| TIMER_CTL0    | control register 0                        |
| TIMER_CTL1    | control register 1                        |
| TIMER_SMCFG   | slave mode configuration register         |
| TIMER_DMINTEN | 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              |

### 3.25.2. Descriptions of Peripheral functions

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

**Table 3-730. TIMERx firmware function**

| Function name | Function description |
|---------------|----------------------|
| timer_deinit  | deinit a TIMER       |

| Function name                              | Function description  |
|--|---|
| timer_struct_para_init                     | initialize TIMER init parameter struct                            |
| 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                    | configure TIMER counter alignment mode                            |
| timer_counter_up_direction                 | configure TIMER counter up direction                              |
| timer_counter_down_direction               | configure 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 TIMER break parameter struct                           |
| 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                | enable or disable TIMER primary output function                   |
| timer_channel_control_shadow_config        | enable or disable channel capture/compare control shadow register |
| timer_channel_control_shadow_update_config | configure TIMER channel control shadow register update control    |
| timer_channel_output_struct_para_init      | initialize TIMER channel output parameter struct                  |
| 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                        |

| Function name                                      | Function description   |
|--|--|
| 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 TIMER channel input parameter struct              |
| timer_input_capture_config                         | configure TIMER input capture parameter                      |
| timer_channel_input_capture_prescaler_config       | configure TIMER channel input capture prescaler value        |
| timer_channel_capture_value_register_read          | read TIMER channel capture compare register value            |
| timer_input_pwm_capture_config                     | configure TIMER input pwm capture function                   |
| timer_hall_mode_config                             | configure TIMER hall sensor mode                             |
| timer_input_trigger_source_select                  | select TIMER input trigger source                            |
| timer_master_output_trigger_source_select          | select TIMER master mode output trigger source               |
| timer_slave_mode_select                            | select TIMER slave mode                                      |
| timer_master_slave_mode_config                     | configure TIMER master slave mode                            |
| timer_external_trigger_config                      | configure TIMER external trigger input                       |
| timer_quadrature_decoder_mode_config               | configure TIMER quadrature decoder mode                      |
| timer_internal_clock_config                        | configure TIMER internal clock mode                          |
| timer_internal_trigger_as_external_clock_config    | configure TIMER the internal trigger as external clock input |
| timer_external_trigger_as_external_clock_config    | configure TIMER the external trigger as external clock input |
| timer_external_clock_mode0_config                  | configure TIMER the external clock mode 0                    |
| timer_external_clock_mode1_config                  | configure TIMER the external clock mode 1                    |
| timer_external_clock_mode1_disable                 | disable TIMER the external clock mode 1                      |
| timer_flag_get                                     | get TIMER flags  |
| timer_flag_clear                                   | clear TIMER flags  |
| timer_interrupt_enable                             | enable the TIMER interrupt                                   |
| timer_interrupt_disable                            | disable the TIMER interrupt                                  |
| timer_interrupt_flag_get                           | get timer interrupt flags                                    |
| timer_interrupt_flag_clear                         | clear TIMER interrupt flags                                  |

### Structure timer\_parameter\_struct

**Table 3-731. Structure timer\_parameter\_struct**

| Member name       | Function description  |
|-------------------|---|
| prescaler         | prescaler value(0~65535)  |
| alignedmode       | aligned mode(TIMER_COUNTER_EDGE,<br>TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP,<br>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,<br>TIMER_CKDIV_DIV4)   |
| repetitioncounter | the counter repetition value(0~255)   |

### Structure timer\_break\_parameter\_struct

**Table 3-732. Structure timer\_break\_parameter\_struct**

| Member name     | Function description  |
|-----------------|---|
| runoffstate     | run mode off-state (TIMER_ROS_STATE_ENABLE,<br>TIMER_ROS_STATE_DISABLE)   |
| ideloffstate    | idle mode off-state(TIMER_IOS_STATE_ENABLE,<br>TIMER_IOS_STATE_DISABLE)   |
| deadtime        | dead time(0~255)  |
| breakpolarity   | break polarity(TIMER_BREAK_POLARITY_LOW,<br>TIMER_BREAK_POLARITY_HIGH)  |
| outputautostate | output automatic enable (TIMER_OUTAUTO_ENABLE,<br>TIMER_OUTAUTO_DISABLE)  |
| protectmode     | complementary register protect control(TIMER_CCHP_PROT_OFF,<br>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 3-733. 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,<br>TIMER_CCXN_DISABLE)              |
| ocpolarity   | channel output polarity(TIMER_OC_POLARITY_HIGH,<br>TIMER_OC_POLARITY_LOW)                 |
| ocnpolarity  | channel complementary output polarity(TIMER_OCN_POLARITY_HIGH,<br>TIMER_OCN_POLARITY_LOW) |
| ocidlestate  | idle state of channel output(TIMER_OC_IDLE_STATE_LOW,<br>TIMER_OC_IDLE_STATE_HIGH)        |

| Member name  | Function description  |
|--------------|---|
| ocnidlestate | idle state of channel complementary output(TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH) |

### Structure timer\_ic\_parameter\_struct

Table 3-734. 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 3-735. Function timer\_deinit

|                              |  |
|------------------------------|--|
| Function name                | timer_deinit                                       |
| Function prototype           | void timer_deinit(uint32_t timer_periph);          |
| Function descriptions        | deinit a TIMER                                     |
| Precondition                 | -  |
| The called functions         | rcu_periph_reset_enable / rcu_periph_reset_disable |
| <b>Input parameter{in}</b>   |  |
| timer_periph                 | TIMER peripheral                                   |
| TIMERx(x=0..13)              | 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 3-736. Function timer\_struct\_para\_init

|                    |  |
|--------------------|--|
| Function name      | timer_struct_para_init   |
| Function prototype | void timer_struct_para_init(timer_parameter_struct* initpara); |

|                              |  |
|------------------------------|--|
| <b>Function descriptions</b> | initialize TIMER init parameter struct with a default value  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| <b>initpara</b>              | TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-731. Structure timer parameter struct.</a> |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* initialize the TIMER structure */

timer_parameter_struct initpara;

timer_struct_para_init(&initpara);
```

### timer\_init

The description of timer\_init is shown as below:

**Table 3-737. 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   |
| <b>Input parameter{in}</b>   |  |
| <b>initpara</b>              | TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-731. Structure timer parameter struct.</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* initialize TIMER0 */

timer_parameter_struct timer_initpara;

timer_initpara.prescaler = 107;
```

```

timer_initpara.alignedmode = TIMER_COUNTER_EDGE;
timer_initpara.counterdirection = TIMER_COUNTER_UP;
timer_initpara.period = 999;
timer_initpara.clockdivision = TIMER_CKDIV_DIV1;
timer_initpara.repetitioncounter = 1;
timer_init(TIMER0, &timer_initpara);
  
```

### **timer\_enable**

The description of timer\_enable is shown as below:

**Table 3-738. Function timer\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_enable                              |
| <b>Function prototype</b>    | void timer_enable(uint32_t timer_periph); |
| <b>Function descriptions</b> | enable a TIMER                            |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| timer_periph                 | TIMER peripheral                          |
| TIMERx(x=0..13)              | TIMER peripheral selection                |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```

/* enable TIMER0 */

timer_enable(TIMER0);
  
```

### **timer\_disable**

The description of timer\_disable is shown as below:

**Table 3-739. 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>   |  |
| timer_periph                 | TIMER peripheral                           |
| TIMERx(x=0..13)              | TIMER peripheral selection                 |

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

Example:

```
/* disable TIMER0 */

timer_disable(TIMER0);
```

### **timer\_auto\_reload\_shadow\_enable**

The description of timer\_auto\_reload\_shadow\_enable is shown as below:

**Table 3-740. 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>  | -  |
| 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 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 3-741. 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>  | -   |
| Input parameter{in}          |   |
| <b>timer_periph</b>          | TIMER peripheral  |
| <i>TIMERx(x=0..13)</i>       | TIMER peripheral selection                                    |

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

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 3-742. Function timer\_update\_event\_enable**

| <b>Function name</b>         | timer_update_event_enable                              |
|------------------------------|--|
| <b>Function prototype</b>    | void timer_update_event_enable(uint32_t timer_periph); |
| <b>Function descriptions</b> | enable the update event                                |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| Input parameter{in}          |  |
| <b>timer_periph</b>          | TIMER peripheral                                       |
| <i>TIMERx(x=0..13)</i>       | TIMER peripheral selection                             |
| Output parameter{out}        |  |
| -                            | -  |
| Return value                 |  |
| -                            | -  |

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 3-743. Function timer\_update\_event\_disable**

| <b>Function name</b>         | timer_update_event_disable                              |
|------------------------------|---|
| <b>Function prototype</b>    | void timer_update_event_disable(uint32_t timer_periph); |
| <b>Function descriptions</b> | disable the update event                                |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| Input parameter{in}          |   |
| <b>timer_periph</b>          | TIMER peripheral  |
| <i>TIMERx(x=0..13)</i>       | TIMER peripheral selection                              |

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

Example:

```
/* disable TIMER0 the update event */

timer_update_event_disable(TIMER0);
```

### timer\_counter\_alignment

The description of timer\_counter\_alignment is shown as below:

**Table 3-744. Function timer\_counter\_alignment**

| Function name             |  | timer_counter_alignment   |
|---------------------------|--|---|
| Function prototype        |  | void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);  |
| Function descriptions     |  | configure TIMER counter alignment mode  |
| Precondition              |  | -   |
| The called functions      |  | -   |
| Input parameter{in}       |  |   |
| timer_periph              |  | TIMER peripheral  |
| TIMERx(x=0..4, 7..13)     |  | TIMER peripheral selection  |
| Input parameter{in}       |  |   |
| aligned                   |  | alignment mode  |
| TIMER_COUNTER_EDGE        |  | No center-aligned mode(edge-aligned mode). The direction of the counter is specified by the DIR bit.  |
| TIMER_COUNTER_CENTER_DOWN |  | Center-aligned and counting down assert mode. The counter counts under center aligned and channel is configured in output mode(CHxMS=00 in TIMERx_CHCTL0register). Only when the counter is counting down, compare interrupt flag of channels can be set.                     |
| TIMER_COUNTER_CENTER_UP   |  | Center-aligned and counting up assert mode. The counter counts under center aligned and channel is configured in output mode(CHxMS=00 in TIMERx_CHCTL0register). Only when the counter is counting up, compare interrupt flag of channels can be set.                         |
| TIMER_COUNTER_BOTH        |  | Center-aligned and counting up/down assert mode. The counter counts under center-aligned and channel is configured in output mode(CHxMS=00 in TIMERx_CHCTL0 register). Both when the counter is counting up and counting down, compare interrupt flag of channels can be set. |
| Output parameter{out}     |  |   |
| -                         |  | -   |
| Return value              |  |   |
| -                         |  | -   |

Example:

---

```

/* set TIMER0 counter center-aligned and counting up 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 3-745. 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>          | configure TIMER counter no center-aligned mode(edge-aligned mode) |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| timer_periph                 | 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 3-746. 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>          | configure TIMER counter no center-aligned mode(edge-aligned mode) |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| timer_periph                 | 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 3-747. 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,<br>uint8_t pscreload); |
| <b>Function descriptions</b>   | configure TIMER prescaler   |
| <b>Precondition</b>            | -   |
| <b>The called functions</b>    | -   |
| <b>Input parameter{in}</b>     |   |
| timer_periph                   | TIMER peripheral  |
| <i>TIMERx(x=0..13)</i>         | TIMER peripheral selection  |
| <b>Input parameter{in}</b>     |   |
| prescaler                      | prescaler value(0~65535)  |
| <b>Input parameter{in}</b>     |   |
| pscreload                      | 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 3-748. 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, uint8_t repetition); |
| <b>Function descriptions</b> | configure TIMER repetition register value                                      |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |

| Input parameter{in}   |                                     |
|-----------------------|-------------------------------------|
| timer_periph          | TIMER peripheral                    |
| <i>TIMERx(x=0..7)</i> | TIMER peripheral selection          |
| Input parameter{in}   |                                     |
| repetition            | the counter repetition value(0~255) |
| Output parameter{out} |                                     |
| -                     | -                                   |
| Return value          |                                     |
| -                     | -                                   |

Example:

```
/* configure TIMER0 repetition register value */

timer_repetition_value_config(TIMER0, 98);
```

### **timer\_autoreload\_value\_config**

The description of timer\_autoreload\_value\_config is shown as below:

**Table 3-749. Function timer\_autoreload\_value\_config**

| Function name          | timer_autoreload_value_config   |
|------------------------|---|
| Function prototype     | void timer_autoreload_value_config(uint32_t timer_periph, uint32_t autoreload); |
| Function descriptions  | configure TIMER autoreload 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}    |   |
| autoreload             | the counter auto-reload value, 0~65535  |
| Output parameter{out}  |   |
| -                      | -   |
| Return value           |   |
| -                      | -   |

Example:

```
/* configure TIMER autoreload register value */

timer_autoreload_value_config(TIMER0, 3000);
```

### **timer\_counter\_value\_config**

The description of timer\_counter\_value\_config is shown as below:

**Table 3-750. Function timer\_counter\_value\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_counter_value_config  |
| <b>Function prototype</b>    | void timer_counter_value_config(uint32_t timer_periph, uint32_t counter); |
| <b>Function descriptions</b> | configure TIMER counter register value                                    |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| timer_periph                 | TIMER peripheral  |
| <i>TIMERx(x=0..13)</i>       | TIMER peripheral selection  |
| <b>Input parameter{in}</b>   |   |
| counter                      | the counter value, 0~65535  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* configure TIMER0 counter register value */

timer_counter_value_config(TIMER0);
```

### **timer\_counter\_read**

The description of timer\_counter\_read is shown as below:

**Table 3-751. Function timer\_counter\_read**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_counter_read                                  |
| <b>Function prototype</b>    | uint32_t timer_counter_read(uint32_t timer_periph); |
| <b>Function descriptions</b> | read TIMER counter value                            |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| timer_periph                 | TIMER peripheral                                    |
| <i>TIMERx(x=0..13)</i>       | TIMER peripheral selection                          |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| uint32_t                     | counter value                                       |

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 3-752. 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, 0~65535                     |

Example:

```
/* read TIMER0 prescaler value */

uint16_t i = 0;

i = timer_prescaler_read(TIMER0);
```

### timer\_single\_pulse\_mode\_config

The description of timer\_single\_pulse\_mode\_config is shown as below:

**Table 3-753. 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..8, 11)</i>        | TIMER peripheral selection  |
| <b>Input parameter{in}</b>       |   |
| <b>spmode</b>                    | pulse mode  |
| <i>TIMER_SP_MODE_SIN_GLE</i>     | single pulse mode   |
| <i>TIMER_SP_MODE_REPEATITIVE</i> | repetitive pulse mode   |

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

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 3-754. 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, uint8_t update);   |
| <b>Function descriptions</b>    | configure TIMER update source   |
| <b>Precondition</b>             | -   |
| <b>The called functions</b>     | -   |
| Input parameter{in}             |   |
| <b>timer_periph</b>             | TIMER peripheral  |
| <i>TIMERx(x=0..13)</i>          | TIMER peripheral selection  |
| Input parameter{in}             |   |
| <b>update</b>                   | update source   |
| <i>TIMER_UPDATE_SRC_GLOBAL</i>  | any of the following events generate an update interrupt or DMA request:<br>- the UPG bit is set<br>- the counter generates an overflow or underflow event<br>- the slave mode controller generates an update event |
| <i>TIMER_UPDATE_SRC_REGULAR</i> | only counter overflow/underflow generates an update interrupt or DMA request  |
| Output parameter{out}           |   |
| -                               | -   |
| 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 3-755. 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>                | TIMER peripheral selection (x 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 3-756. 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>                | TIMER peripheral selection (x 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 3-757. 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,<br>uint8_t dma_request); |
| <b>Function descriptions</b>         | channel DMA request source selection   |
| <b>Precondition</b>                  | -  |
| <b>The called functions</b>          | -  |
| <b>Input parameter{in}</b>           |  |
| <b>timer_periph</b>                  | TIMER peripheral   |
| <i>TIMERx(x=0..4, 7)</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 y is sent when channel y event occurs                                 |
| <i>TIMER_DMAREQUEST_UPDATEEVENT</i>  | DMA request of channel y is sent when update event occurs                                    |
| <b>Output parameter{out}</b>         |  |
| -                                    | -  |
| <b>Return value</b>                  |  |
| -                                    | -  |

Example:

```
/* TIMER0 channel DMA request of channel y 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 3-758. Function timer\_dma\_transfer\_config**

|                                     |   |
|-------------------------------------|---|
| <b>Function name</b>                | timer_dma_transfer_config   |
| <b>Function prototype</b>           | void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth); |
| <b>Function descriptions</b>        | configure the TIMER DMA transfer  |
| <b>Precondition</b>                 | -   |
| <b>The called functions</b>         | -   |
| <b>Input parameter{in}</b>          |   |
| <b>timer_periph</b>                 | TIMER peripheral  |
| <i>TIMERx</i>                       | TIMER peripheral selection (x 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_TA_CHCTL0</i>   | DMA transfer address is TIMER_CHCTL0, TIMERx(x=0..4, 7)   |
| <i>TIMER_DMACFG_DMA_TA_CHCTL1</i>   | DMA transfer address is TIMER_CHCTL1, TIMERx(x=0..4, 7)   |
| <i>TIMER_DMACFG_DMA_TA_CHCTL2</i>   | DMA transfer address is TIMER_CHCTL2, TIMERx(x=0..4, 7)   |
| <i>TIMER_DMACFG_DMA_TA_CNT</i>      | DMA transfer address is TIMER_CNT, TIMERx(x=0..4, 7)  |
| <i>TIMER_DMACFG_DMA_TA_PSC</i>      | DMA transfer address is TIMER_PSC, TIMERx(x=0..4, 7)  |
| <i>TIMER_DMACFG_DMA_TA_CAR</i>      | MA transfer address is TIMER_CAR, TIMERx(x=0..4, 7)   |
| <i>TIMER_DMACFG_DMA_TA_CREP</i>     | DMA transfer address is TIMER_CREP, TIMERx(x=0, 7)  |

|                                   |   |
|-----------------------------------|---|
| <i>TIMER_DMACFG_DMA_TA_CH0CV</i>  | DMA transfer address is TIMER_CH0CV, TIMERx(x=0..4, 7)    |
| <i>TIMER_DMACFG_DMA_TA_CH1CV</i>  | DMA transfer address is TIMER_CH1CV, TIMERx(x=0..4, 7)    |
| <i>TIMER_DMACFG_DMA_TA_CH2CV</i>  | DMA transfer address is TIMER_CH2CV, TIMERx(x=0..4, 7)    |
| <i>TIMER_DMACFG_DMA_TA_CH3CV</i>  | DMA transfer address is TIMER_CH3CV, TIMERx(x=0..4, 7)    |
| <i>TIMER_DMACFG_DMA_TA_CCHP</i>   | DMA transfer address is TIMER_CCHP, TIMERx(x=0, 7)        |
| <i>TIMER_DMACFG_DMA_TA_DMACFG</i> | DMA transfer address is TIMER_DMACFG, TIMERx(x=0..4, 7)   |
| <i>TIMER_DMACFG_DMA_TA_DMATB</i>  | DMA transfer address is TIMER_DMATB, TIMERx(x=0..4, 7)    |
| <b>Input parameter{in}</b>        |   |
| <b>dma_lenth</b>                  | DMA transfer count, TIMER_DMACFG_DMATC_xTRANSFER(x=1..18) |
| <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 3-759. 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   |
| <b>TIMERx</b>                | TIMER peripheral selection (x please refer to the following parameters)    |
| <b>Input parameter{in}</b>   |  |
| <b>event</b>                 | the timer software event generation sources                                |
| <b>TIMER_EVENT_SRC_U</b>     | update event, TIMERx(x=0..13)  |

| PG                           |   |
|------------------------------|---|
| <i>TIMER_EVENT_SRC_C_H0G</i> | channel 0 capture or compare event generation, TIMERx(x=0..4, 7..13)    |
| <i>TIMER_EVENT_SRC_C_H1G</i> | channel 1 capture or compare event generation, TIMERx(x=0..4, 7, 8, 11) |
| <i>TIMER_EVENT_SRC_C_H2G</i> | channel 2 capture or compare event generation, TIMERx(x=0..4, 7)        |
| <i>TIMER_EVENT_SRC_C_H3G</i> | channel 3 capture or compare event generation, TIMERx(x=0..4, 7)        |
| <i>TIMER_EVENT_SRC_C_MTG</i> | channel commutation event generation, TIMERx(x=0, 7)                    |
| <i>TIMER_EVENT_SRC_T_RGG</i> | trigger event generation, TIMERx(x=0..4, 7, 8, 11)                      |
| <i>TIMER_EVENT_SRC_B_RKG</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 3-760. 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 TIMER break parameter struct with a default value   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| <b>breakpara</b>             | TIMER break parameter struct, the structure members can refer to <a href="#">Table 3-732. Structure timer break parameter struct</a> . |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* initialize the TIMER break parameter structure */
```

```
timer_break_parameter_struct breakpara;
```

```
timer_break_struct_para_init(&breakpara);
```

### **timer\_break\_config**

The description of timer\_break\_config is shown as below:

**Table 3-761. Function timer\_break\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | timer_break_config   |
| <b>Function prototype</b>    | void timer_break_config(uint32_t timer_periph,<br>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>   |  |
| timer_periph                 | TIMER peripheral   |
| TIMERx(x=0, 7)               | TIMER peripheral selection   |
| <b>Input parameter{in}</b>   |  |
| breakpara                    | TIMER break parameter struct, the structure members can refer to <a href="#">Table 3-732, 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_OUTAUTOSTATE_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 3-762. 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-filled in TIMERx_CCHP register is 00. |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| timer_periph                 | TIMER peripheral   |
| TIMERx(x=0, 7)               | 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 3-763. 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-filled in TIMERx_CCHP register is 00. |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| timer_periph                 | TIMER peripheral   |
| TIMERx(x=0, 7)               | 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 t is shown as below:

**Table 3-764. Function timer\_automatic\_output\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_automatic_output_enable   |
| <b>Function prototype</b>    | void timer_automatic_output_enable(uint32_t timer_periph);                                |
| <b>Function descriptions</b> | enable TIMER output automatic function  |
| <b>Precondition</b>          | This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00. |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>timer_periph</b>          | TIMER peripheral  |
| <i>TIMERx(x=0,7)</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 t is shown as below:

**Table 3-765. Function timer\_automatic\_output\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_automatic_output_disable  |
| <b>Function prototype</b>    | void timer_automatic_output_disable(uint32_t timer_periph);                               |
| <b>Function descriptions</b> | disable TIMER output automatic function   |
| <b>Precondition</b>          | This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00. |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>timer_periph</b>          | TIMER peripheral  |
| <i>TIMERx(x=0, 7)</i>        | TIMER peripheral selection  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable TIMER0 output automatic function */
```

```
timer_automatic_output_disable(TIMER0);
```

### **timer\_primary\_output\_config**

The description of timer\_primary\_output\_config is shown as below:

**Table 3-766. Function timer\_primary\_output\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | timer_primary_output_config  |
| <b>Function prototype</b>    | void timer_primary_output_config(uint32_t timer_periph, ControlStatus newvalue); |
| <b>Function descriptions</b> | enable or disable 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 3-767. 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> | enable or disable channel capture/compare control shadow register                        |
| <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>newvalue</b>       | control value    |
| <b>ENABLE</b>         | enable function  |
| <b>DISABLE</b>        | disable function |
| Output parameter{out} |                  |
| -                     | -                |
| Return value          |                  |
| -                     | -                |

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 3-768. Function timer\_channel\_control\_shadow\_update\_config**

| Function name                | timer_channel_control_shadow_update_config  |
|------------------------------|---|
| <b>Function prototype</b>    | void timer_channel_control_shadow_update_config(uint32_t timer_periph,<br>uint32_t ccuctl); |
| <b>Function descriptions</b> | configure TIMER channel control shadow register update control                              |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| Input parameter{in}          |   |
| <b>timer_periph</b>          | TIMER peripheral  |
| <b>TIMERx(x=0, 7)</b>        | TIMER peripheral selection  |
| Input parameter{in}          |   |
| <b>ccuctl</b>                | channel control shadow register update control  |
| <b>TIMER_UPDATECTL_CCU</b>   | the shadow registers update by when CMTG bit is set   |
| <b>TIMER_UPDATECTL_CUTRI</b> | 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 3-769. Function timer\_channel\_output\_struct\_para\_init**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_channel_output_struct_para_init   |
| <b>Function prototype</b>    | void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpara);  |
| <b>Function descriptions</b> | initialize TIMER channel output parameter struct with a default value   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| <b>ocpara</b>                | TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-733. Structure timer_oc_parameter_struct.</a> |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* initialize the TIMER channel output parameter structure */

timer_oc_parameter_struct ocpara;

timer_channel_output_struct_para_init(&ocpara);
```

### timer\_channel\_output\_config

The description of timer\_channel\_output\_config is shown as below:

**Table 3-770. 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  |
| <b>TIMERx</b>                | TIMER peripheral selection (x please refer to the following parameters)                                       |
| <b>Input parameter{in}</b>   |   |
| <b>channel</b>               | channel to be configured  |
| <b>TIMER_CH_0</b>            | TIMER channel0(TIMERx(x=0..4, 7..13))   |
| <b>TIMER_CH_1</b>            | TIMER channel1(TIMERx(x=0..4, 7, 8, 11))  |
| <b>TIMER_CH_2</b>            | TIMER channel2(TIMERx(x=0..4, 7))   |

|                              |  |
|------------------------------|--|
| <i>TIMER_CH_3</i>            | TIMER channel3(TIMERx(x=0..4, 7))  |
| <b>Input parameter{in}</b>   |  |
| <i>ocpara</i>                | TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-733. 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_ocintpara;

timer_ocintpara.outputstate = TIMER_CCX_ENABLE;
timer_ocintpara.outputnstate = TIMER_CCXN_ENABLE;
timer_ocintpara.ocpolarity = TIMER_OC_POLARITY_HIGH;
timer_ocintpara.ocnpolarity = TIMER_OCN_POLARITY_HIGH;
timer_ocintpara.ocidlestate = TIMER_OC_IDLE_STATE_HIGH;
timer_ocintpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;
timer_channel_output_config(TIMER0, TIMER_CH_0, &timer_ocintpara);

```

### **timer\_channel\_output\_mode\_config**

The description of `timer_channel_output_mode_config` is shown as below:

**Table 3-771. Function `timer_channel_output_mode_config`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | <code>timer_channel_output_mode_config</code>   |
| <b>Function prototype</b>    | <code>void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint16_t ocmode);</code> |
| <b>Function descriptions</b> | configure TIMER channel output compare mode   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <i>timer_periph</i>          | TIMER peripheral  |
| <i>TIMERx</i>                | TIMER peripheral selection (x please refer to the following parameters)                                       |
| <b>Input parameter{in}</b>   |   |
| <i>channel</i>               | channel to be configured  |
| <i>TIMER_CH_0</i>            | TIMER channel0(TIMERx(x=0..4, 7..13))   |
| <i>TIMER_CH_1</i>            | TIMER channel1(TIMERx(x=0..4, 7, 8, 11))  |
| <i>TIMER_CH_2</i>            | TIMER channel2(TIMERx(x=0..4, 7))   |
| <i>TIMER_CH_3</i>            | TIMER channel3(TIMERx(x=0..4, 7))   |

| Input parameter{in}            |                             |
|--------------------------------|-----------------------------|
| <b>ocmode</b>                  | channel output compare mode |
| <i>TIMER_OC_MODE_TIMING</i>    | timing mode                 |
| <i>TIMER_OC_MODE_ACTIVE</i>    | active mode                 |
| <i>TIMER_OC_MODE_INACTIVE</i>  | inactive mode               |
| <i>TIMER_OC_MODE_TOGGLE</i>    | toggle mode                 |
| <i>TIMER_OC_MODE_FORCELOW</i>  | force low mode              |
| <i>TIMER_OC_MODE_FORCEHIGH</i> | force high mode             |
| <i>TIMER_OC_MODE_PWM0</i>      | PWM mode 0                  |
| <i>TIMER_OC_MODE_PWM1</i>      | PWM mode 1                  |
| Output parameter{out}          |                             |
| -                              | -                           |
| Return value                   |                             |
| -                              | -                           |

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 3-772. 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,<br>uint16_t channel, uint16_t pulse); |
| <b>Function descriptions</b> | configure TIMER channel output pulse value  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| Input parameter{in}          |   |
| <b>timer_periph</b>          | TIMER peripheral  |
| <i>TIMERx</i>                | TIMER peripheral selection (x please refer to the following parameters)                                   |
| Input parameter{in}          |   |
| <b>channel</b>               | channel to be configured  |

|                              |  |
|------------------------------|--|
| <i>TIMER_CH_0</i>            | TIMER channel0(TIMERx(x=0..4, 7..13))    |
| <i>TIMER_CH_1</i>            | TIMER channel1(TIMERx(x=0..4, 7, 8, 11)) |
| <i>TIMER_CH_2</i>            | TIMER channel2(TIMERx(x=0..4, 7))        |
| <i>TIMER_CH_3</i>            | TIMER channel3(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 3-773. 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>                  | TIMER peripheral selection (x please refer to the following parameters)                              |
| <b>Input parameter{in}</b>     |  |
| <b>channel</b>                 | channel to be configured   |
| <i>TIMER_CH_0</i>              | TIMER channel0(TIMERx(x=0..4, 7..13))  |
| <i>TIMER_CH_1</i>              | TIMER channel1(TIMERx(x=0..4, 7, 8, 11))   |
| <i>TIMER_CH_2</i>              | TIMER channel2(TIMERx(x=0..4, 7))  |
| <i>TIMER_CH_3</i>              | TIMER channel3(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>   |  |
| -                              | -  |

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

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 3-774. Function timer\_channel\_output\_fast\_config**

|                                   |  |
|-----------------------------------|--|
| <b>Function name</b>              | timer_channel_output_fast_config   |
| <b>Function prototype</b>         | void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast); |
| <b>Function descriptions</b>      | configure TIMER channel output fast function   |
| <b>Precondition</b>               | -  |
| <b>The called functions</b>       | -  |
| <b>Input parameter{in}</b>        |  |
| <b>timer_periph</b>               | TIMER peripheral   |
| <b>TIMERx</b>                     | TIMER peripheral selection (x please refer to the following parameters)                          |
| <b>Input parameter{in}</b>        |  |
| <b>channel</b>                    | channel to be configured   |
| <b>TIMER_CH_0</b>                 | TIMER channel0(TIMERx(x=0..4, 7..13))  |
| <b>TIMER_CH_1</b>                 | TIMER channel1(TIMERx(x=0..4, 7, 8, 11))   |
| <b>TIMER_CH_2</b>                 | TIMER channel2(TIMERx(x=0..4, 7))  |
| <b>TIMER_CH_3</b>                 | TIMER channel3(TIMERx(x=0..4, 7))  |
| <b>Input parameter{in}</b>        |  |
| <b>ocfast</b>                     | channel output fast function   |
| <b>TIMER_OC_FAST_ENA<br/>BLE</b>  | channel output fast function enable  |
| <b>TIMER_OC_FAST_DIS<br/>ABLE</b> | channel output fast function disable   |
| <b>Output parameter{out}</b>      |  |
| -                                 | -  |
| <b>Return value</b>               |  |
| -                                 | -  |

Example:

```
/* configure TIMER0 channel 0 output fast function */
timer_channel_output_fast_config(TIMER0, TIMER_CH_0, TIMER_OC_FAST_ENABLE);
```

### timer\_channel\_output\_clear\_config

The description of timer\_channel\_output\_clear\_config is shown as below:

**Table 3-775. 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 peripheral   |
| <b>TIMERx(x=0..4, 7)</b>      | TIMER peripheral selection   |
| <b>Input parameter{in}</b>    |  |
| <b>channel</b>                | channel to be configured   |
| <b>TIMER_CH_0</b>             | TIMER channel0   |
| <b>TIMER_CH_1</b>             | TIMER channel1   |
| <b>TIMER_CH_2</b>             | TIMER channel2   |
| <b>TIMER_CH_3</b>             | TIMER channel3   |
| <b>Input parameter{in}</b>    |  |
| <b>occlear</b>                | channel output clear function  |
| <b>TIMER_OC_CLEAR_ENABLE</b>  | channel output clear function enable   |
| <b>TIMER_OC_CLEAR_DISABLE</b> | channel output clear function disable  |
| <b>Output parameter{out}</b>  |  |
| -                             | -  |
| <b>Return value</b>           |  |
| -                             | -  |

Example:

```
/* configure TIMER0 channel 0 output clear function */
timer_channel_output_clear_config(TIMER0, TIMER_CH_0,
TICKER_OC_CLEAR_ENABLE);
```

### timer\_channel\_output\_polarity\_config

The description of timer\_channel\_output\_polarity\_config is shown as below:

**Table 3-776. 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  |
| <b>TIMERx</b>                 | TIMER peripheral selection (x please refer to the following parameters) |
| <b>Input parameter{in}</b>    |   |
| <b>channel</b>                | channel to be configured  |
| <b>TIMER_CH_0</b>             | TIMER channel0(TIMERx(x=0..4, 7..13))                                   |
| <b>TIMER_CH_1</b>             | TIMER channel1(TIMERx(x=0..4, 7, 8, 11))                                |
| <b>TIMER_CH_2</b>             | TIMER channel2(TIMERx(x=0..4, 7))                                       |
| <b>TIMER_CH_3</b>             | TIMER channel3(TIMERx(x=0..4, 7))                                       |
| <b>Input parameter{in}</b>    |   |
| <b>ocpolarity</b>             | channel output polarity   |
| <b>TIMER_OC_POLARITY_HIGH</b> | channel output polarity is high   |
| <b>TIMER_OC_POLARITY_LOW</b>  | 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,
TICKER_OC_POLARITY_HIGH);
```

### **timer\_channel\_complementary\_output\_polarity\_config**

The description of timer\_channel\_complementary\_output\_polarity\_config is shown as below:

**Table 3-777. 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   |
| <b>TIMERx</b>                | TIMER peripheral selection (x please refer to the following parameters)  |
| <b>Input parameter{in}</b>   |  |

|                                |   |
|--------------------------------|---|
| <b>channel</b>                 | channel to be configured                      |
| <i>TIMER_CH_0</i>              | TIMER channel0(TIMERx(x=0..4, 7..13))         |
| <i>TIMER_CH_1</i>              | TIMER channel1(TIMERx(x=0..4, 7, 8, 11))      |
| <i>TIMER_CH_2</i>              | TIMER channel2(TIMERx(x=0..4, 7))             |
| <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,
TICKER_OCN_POLARITY_HIGH);
```

### **timer\_channel\_output\_state\_config**

The description of `timer_channel_output_state_config` is shown as below:

**Table 3-778. 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>                | TIMER peripheral selection (x please refer to the following parameters)                          |
| <b>Input parameter{in}</b>   |  |
| <b>channel</b>               | channel to be configured   |
| <i>TIMER_CH_0</i>            | TIMER channel0(TIMERx(x=0..4, 7..13))  |
| <i>TIMER_CH_1</i>            | TIMER channel1(TIMERx(x=0..4, 7, 8, 11))   |
| <i>TIMER_CH_2</i>            | TIMER channel2(TIMERx(x=0..4, 7))  |
| <i>TIMER_CH_3</i>            | TIMER channel3(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 3-779. 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  |
| <b>Input parameter{in}</b>   |   |
| <b>channel</b>               | channel to be configured  |
| <i>TIMER_CH_0</i>            | TIMER channel0  |
| <i>TIMER_CH_1</i>            | TIMER channel1  |
| <i>TIMER_CH_2</i>            | TIMER channel2  |
| <b>Input parameter{in}</b>   |   |
| <b>ocnstate</b>              | TIMER channel complementary output enable state   |
| <i>TIMER_CCXN_ENABLE</i>     | channel complementary enable  |
| <i>TIMER_CCXN_DISABLE</i>    | channel complementary disable   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* configure TIMER0 channel 0 complementary output enable state */

timer_channel_complementary_output_state_config(TIMER0, TIMER_CH_0,
TICKER_CCXN_ENABLE);
```

### **timer\_channel\_input\_struct\_para\_init**

The description of timer\_channel\_input\_struct\_para\_init is shown as below:

**Table 3-780. 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 TIMER channel input parameter struct with a default value   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| <b>icpara</b>                | TIMER channel intput parameter struct, the structure members can refer to <a href="#">Table 3-734. Structure timer_ic_parameter_struct</a> . |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* initialize the TIMER channel input parameter structure */

timer_ic_parameter_struct icpara;

timer_channel_input_struct_para_init(&icpara);
```

### **timer\_input\_capture\_config**

The description of timer\_input\_capture\_config is shown as below:

**Table 3-781. Function timer\_input\_capture\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | timer_input_capture_config   |
| <b>Function prototype</b>    | void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara); |
| <b>Function descriptions</b> | configure TIMER input capture parameter  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | timer_channel_input_capture_prescaler_config   |
| <b>Input parameter{in}</b>   |  |
| <b>timer_periph</b>          | TIMER peripheral   |
| <b>TIMERx</b>                | TIMER peripheral selection (x please refer to the following parameters)                                      |
| <b>Input parameter{in}</b>   |  |
| <b>channel</b>               | channel to be configured   |
| <b>TIMER_CH_0</b>            | TIMER channel0(TIMERx(x=0..4, 7..13))  |
| <b>TIMER_CH_1</b>            | TIMER channel1(TIMERx(x=0..4, 7, 8, 11))   |
| <b>TIMER_CH_2</b>            | TIMER channel2(TIMERx(x=0..4, 7))  |

|                              |  |
|------------------------------|--|
| <i>TIMER_CH_3</i>            | TIMER channel3(TIMERx(x=0..4, 7))  |
| <b>Input parameter{in}</b>   |  |
| <i>icpara</i>                | TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-734. Structure timer_ic_parameter_struct.</a> |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```

/* configure TIMER0 input capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;

timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);
  
```

### **timer\_channel\_input\_capture\_prescaler\_config**

The description of *timer\_channel\_input\_capture\_prescaler\_config* is shown as below:

**Table 3-782. Function timer\_channel\_input\_capture\_prescaler\_config**

| Function name              | timer_channel_input_capture_prescaler_config   |
|----------------------------|--|
| Function prototype         | void timer_channel_input_capture_prescaler_config(uint32_t timer_periph,<br>uint16_t channel, uint16_t prescaler); |
| Function descriptions      | configure TIMER channel input capture prescaler value  |
| Precondition               | -  |
| The called functions       | -  |
| <b>Input parameter{in}</b> |  |
| <i>timer_periph</i>        | TIMER peripheral   |
| <i>TIMERx</i>              | TIMER peripheral selection (x please refer to the following parameters)  |
| <b>Input parameter{in}</b> |  |
| <i>channel</i>             | channel to be configured   |
| <i>TIMER_CH_0</i>          | TIMER channel0(TIMERx(x=0..4, 7..13))  |
| <i>TIMER_CH_1</i>          | TIMER channel1(TIMERx(x=0..4, 7, 8, 11))   |
| <i>TIMER_CH_2</i>          | TIMER channel2(TIMERx(x=0..4, 7))  |
| <i>TIMER_CH_3</i>          | TIMER channel3(TIMERx(x=0..4, 7))  |
| <b>Input parameter{in}</b> |  |
| <i>prescaler</i>           | 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,
TICKER_IC_PSC_DIV2);
```

### **timer\_channel\_capture\_value\_register\_read**

The description of timer\_channel\_capture\_value\_register\_read is shown as below:

**Table 3-783. 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   |
| <b>TIMERx</b>                | TIMER peripheral selection (x please refer to the following parameters)                      |
| <b>Input parameter{in}</b>   |  |
| <b>channel</b>               | channel to be configured   |
| <b>TIMER_CH_0</b>            | TIMER channel0(TIMERx(x=0..4, 7..13))  |
| <b>TIMER_CH_1</b>            | TIMER channel1(TIMERx(x=0..4, 7, 8, 11))   |
| <b>TIMER_CH_2</b>            | TIMER channel2(TIMERx(x=0..4, 7))  |
| <b>TIMER_CH_3</b>            | TIMER channel3(TIMERx(x=0..4, 7))  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>uint32_t</b>              | channel capture compare register value, 0~65535  |

Example:

```
/* read TIMER0 channel 0 capture compare register value */

uint32_t CH0_value = 0;

CH0_value = timer_channel_capture_value_register_read(TIMER0, TIMER_CH_0);
```

### timer\_input\_pwm\_capture\_config

The description of timer\_input\_pwm\_capture\_config is shown as below:

**Table 3-784. 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 channel0   |
| <i>TIMER_CH_1</i>               | TIMER channel1   |
| <b>Input parameter{in}</b>      |  |
| <b>icpwm</b>                    | TIMER channel intput parameter struct, the structure members can refer to <a href="#">Table 3-734. Structure timer_ic_parameter_struct</a> . |
| <b>Output parameter{out}</b>    |  |
| -                               | -  |
| <b>Return value</b>             |  |
| -                               | -  |

Example:

```
/* configure TIMER0 input pwm capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;

timer_input_pwm_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);
```

### timer\_hall\_mode\_config

The description of timer\_hall\_mode\_config is shown as below:

**Table 3-785. Function timer\_hall\_mode\_config**

|                           |   |
|---------------------------|---|
| <b>Function name</b>      | timer_hall_mode_config  |
| <b>Function prototype</b> | void timer_hall_mode_config(uint32_t timer_periph, uint8_t hallmode); |

|                                       |                                  |
|---------------------------------------|----------------------------------|
| <b>Function descriptions</b>          | configure TIMER hall sensor mode |
| <b>Precondition</b>                   | -                                |
| <b>The called functions</b>           | -                                |
| <b>Input parameter{in}</b>            |                                  |
| <b>timer_periph</b>                   | TIMER peripheral                 |
| <b>TIMERx(x=0..4, 7)</b>              | TIMER peripheral selection       |
| <b>Input parameter{in}</b>            |                                  |
| <b>hallmode</b>                       | TIMER hall sensor mode state     |
| <b>TIMER_HALLINTERFACE_CE_ENABLE</b>  | TIMER hall sensor mode enable    |
| <b>TIMER_HALLINTERFACE_CE_DISABLE</b> | TIMER hall sensor mode disable   |
| <b>Output parameter{out}</b>          |                                  |
| -                                     | -                                |
| <b>Return value</b>                   |                                  |
| -                                     | -                                |

Example:

```
/* configure TIMER0 hall sensor mode */
timer_hall_mode_config(TIMER0, TIMER_HALLINTERFACE_ENABLE);
```

### timer\_input\_trigger\_source\_select

The description of timer\_input\_trigger\_source\_select is shown as below:

**Table 3-786. 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   |
| <b>TIMERx</b>                   | TIMER peripheral selection (x please refer to the following parameters)            |
| <b>Input parameter{in}</b>      |  |
| <b>intrigger</b>                | trigger selection  |
| <b>TIMER_SMCFG_TRGS_EL_ITI0</b> | internal trigger input 0(ITI0), TIMERx(x=0..4, 7, 8, 11)                           |
| <b>TIMER_SMCFG_TRGS_EL_ITI1</b> | internal trigger input 0(ITI1), TIMERx(x=0..4, 7, 8, 11)                           |
| <b>TIMER_SMCFG_TRGS_EL_ITI2</b> | internal trigger input 0(ITI2), TIMERx(x=0..4, 7, 8, 11)                           |

|  |   |
|--|---|
| <i>TIMER_SMCFG_TRGS</i><br><i>EL_ITI3</i>    | internal trigger input 0(ITI3), TIMERx(x=0..4, 7, 8, 11)          |
| <i>TIMER_SMCFG_TRGS</i><br><i>EL_CI0F_ED</i> | CI0 edge flag(CI0F_ED), TIMERx(x=0..4, 7, 8, 11)                  |
| <i>TIMER_SMCFG_TRGS</i><br><i>EL_CI0FE0</i>  | channel 0 input Filtered output(CI0FE0), TIMERx(x=0..4, 7, 8, 11) |
| <i>TIMER_SMCFG_TRGS</i><br><i>EL_CI1FE1</i>  | channel 1 input Filtered output(CI1FE1), TIMERx(x=0..4, 7, 8, 11) |
| <i>TIMER_SMCFG_TRGS</i><br><i>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 3-787. 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,<br>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   |
| <b>TIMERx</b>                              | TIMER peripheral selection (x please refer to the following parameters)  |
| <b>Input parameter{in}</b>                 |  |
| <b>outrigger</b>                           | master mode control  |
| <i>TIMER_TRI_OUT_SRC</i><br><i>_RESET</i>  | reset, when the UPG bit in the TIMERx_SWEVG register is set or a reset is generated by the slave mode controller, a TRGO pulse occurs. And in the latter case, the signal on TRGO is delayed compared to the actual reset.(TIMERx(x=0..7))   |
| <i>TIMER_TRI_OUT_SRC</i><br><i>_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.(TIMERx(x=0..7)) |
| <i>TIMER_TRI_OUT_SRC_UPDATE</i> | update, in this mode the master mode controller selects the update event as TRGO.(TIMERx(x=0..7))  |
| <i>TIMER_TRI_OUT_SRC_CH0</i>    | a capture or a compare match occurred in channel 0 as trigger output TRGO.(TIMERx(x=0..4,7))   |
| <i>TIMER_TRI_OUT_SRC_O0CPRE</i> | compare, in this mode the master mode controller selects the O0CPRE signal is used as TRGO(TIMERx(x=0..4,7))   |
| <i>TIMER_TRI_OUT_SRC_O1CPRE</i> | compare, in this mode the master mode controller selects the O1CPRE signal is used as TRGO(TIMERx(x=0..4,7))   |
| <i>TIMER_TRI_OUT_SRC_O2CPRE</i> | compare, in this mode the master mode controller selects the O2CPRE signal is used as TRGO(TIMERx(x=0..4,7))   |
| <i>TIMER_TRI_OUT_SRC_O3CPRE</i> | compare, in this mode the master mode controller selects the O3CPRE signal is used as TRGO(TIMERx(x=0..4,7))   |
| <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 3-788. 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_ENCODER_MODE_0</i>     | encoder mode 0   |

|                              |                       |
|------------------------------|-----------------------|
| <i>TIMER_ENCODER_MODE_</i>   | encoder mode 1        |
| <i>DE1</i>                   |                       |
| <i>TIMER_ENCODER_MODE_</i>   | encoder mode 2        |
| <i>DE2</i>                   |                       |
| <i>TIMER_SLAVE_MODE_</i>     | restart mode          |
| <i>RESTART</i>               |                       |
| <i>TIMER_SLAVE_MODE_</i>     | pause mode            |
| <i>PAUSE</i>                 |                       |
| <i>TIMER_SLAVE_MODE_</i>     | event mode            |
| <i>EVENT</i>                 |                       |
| <i>TIMER_SLAVE_MODE_</i>     | external clock mode 0 |
| <i>EXTERNAL0</i>             |                       |
| <b>Output parameter{out}</b> |                       |
| -                            | -                     |
| <b>Return value</b>          |                       |
| -                            | -                     |

Example:

```
/* select TIMER0 slave mode */
timer_slave_mode_select(TIMER0, TIMER_ENCODER_MODE0);
```

### timer\_master\_slave\_mode\_config

The description of timer\_master\_slave\_mode\_config is shown as below:

**Table 3-789. 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, uint32_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>           |   |
| -                                      | -   |

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

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 3-790. Function timer\_external\_trigger\_config**

|                        |   |
|------------------------|---|
| Function name          | timer_external_trigger_config   |
| Function prototype     | void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint8_t extfilter); |
| Function descriptions  | configure TIMER external trigger input  |
| Precondition           | -   |
| The called functions   | -   |
| Input parameter{in}    |   |
| timer_periph           | TIMER peripheral  |
| TIMERx(x=0..4, 7)      | TIMER peripheral selection  |
| Input parameter{in}    |   |
| extprescaler           | external trigger prescaler  |
| TIMER_EXT_TRI_PSC_OFF  | no divided  |
| TIMER_EXT_TRI_PSC_DIV2 | divided by 2  |
| TIMER_EXT_TRI_PSC_DIV4 | divided by 4  |
| TIMER_EXT_TRI_PSC_DIV8 | divided by 8  |
| Input parameter{in}    |   |
| expolarity             | external trigger polarity   |
| TIMER_ETP_FALLING      | active low or falling edge active   |
| TIMER_ETP_RISING       | active high or rising edge active   |
| Input parameter{in}    |   |
| extfilter              | external trigger filter control(0~15)   |
| Output parameter{out}  |   |
| -                      | -   |
| Return value           |   |
| -                      | -   |

Example:

```

/* configure TIMER0 external trigger input */

timer_external_trigger_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,
    TIMER_ETP_FALLING, 10);
  
```

### **timer\_quadrature\_decoder\_mode\_config**

The description of timer\_quadrature\_decoder\_mode\_config is shown as below:

**Table 3-791. 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  |
| <b>TIMERx(x=0..4, 7, 8, 11)</b>  | TIMER peripheral selection  |
| <b>Input parameter{in}</b>       |   |
| <b>decomode</b>                  | quadrature decoder mode   |
| <b>TIMER_ENCODER_MO_DE0</b>      | counter counts on CI0FE0 edge depending on CI1FE1 level   |
| <b>TIMER_ENCODER_MO_DE1</b>      | counter counts on CI1FE1 edge depending on CI0FE0 level   |
| <b>TIMER_ENCODER_MO_DE2</b>      | counter counts on both CI0FE0 and CI1FE1 edges depending on the level<br>of the other input   |
| <b>Input parameter{in}</b>       |   |
| <b>ic0polarity</b>               | IC0 polarity  |
| <b>TIMER_IC_POLARITY_RISING</b>  | capture rising edge   |
| <b>TIMER_IC_POLARITY_FALLING</b> | capture falling edge  |
| <b>Input parameter{in}</b>       |   |
| <b>ic1polarity</b>               | IC1 polarity  |
| <b>TIMER_IC_POLARITY_RISING</b>  | capture rising edge   |
| <b>TIMER_IC_POLARITY_FALLING</b> | capture falling edge  |
| <b>Output parameter{out}</b>     |   |
| -                                | -   |
| <b>Return value</b>              |   |
| -                                | -   |

Example:

```

/* configure TIMER0 quadrature decoder mode */

timer_quadrature_decoder_mode_config(TIMER0, TIMER_ENCODER_MODE0,
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
  
```

### **timer\_internal\_clock\_config**

The description of timer\_internal\_clock\_config is shown as below:

**Table 3-792. 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 3-793. 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_ITIO</i> | internal trigger input 0(ITIO)   |

|                                 |                                |
|---------------------------------|--------------------------------|
| <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 3-794. Function timer\_external\_trigger\_as\_external\_clock\_config**

|                                    |   |
|------------------------------------|---|
| <b>Function name</b>               | timer_external_trigger_as_external_clock_config   |
| <b>Function prototype</b>          | void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t exttrigger, uint16_t expolarity, uint8_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>exttrigger</b>                  | external trigger selection  |
| <i>TIMER_SMCFG_TRGS_EL_CI0F_ED</i> | CI0 edge flag(CI0F_ED)  |
| <i>TIMER_SMCFG_TRGS_EL_CI0FE0</i>  | channel 0 input Filtered output(CI0FE0)   |
| <i>TIMER_SMCFG_TRGS_EL_CI1FE1</i>  | channel 1 input Filtered output(CI1FE1)   |
| <b>Input parameter{in}</b>         |   |
| <b>expolarity</b>                  | external trigger polarity   |
| <i>TIMER_IC_POLARITY_RISING</i>    | active high or rising edge active   |
| <i>TIMER_IC_POLARITY_FALLING</i>   | active low or falling edge active   |

| Input parameter{in}   |                                       |
|-----------------------|---------------------------------------|
| <b>extfilter</b>      | external trigger filter control(0~15) |
| Output parameter{out} |                                       |
| -                     | -                                     |
| Return value          |                                       |
| -                     | -                                     |

Example:

```
/* configure TIMER0 the external trigger CI0FE0 as external clock input */
timer_external_trigger_as_external_clock_config(TIMER0,
  TIMER_SMCFG_TRGSEL_CI0FE0, TIMER_IC_POLARITY_RISING, 0);
```

### **timer\_external\_clock\_mode0\_config**

The description of timer\_external\_clock\_mode0\_config is shown as below:

**Table 3-795. 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, uint8_t extfilter); |
| <b>Function descriptions</b>    | configure TIMER the external clock mode0   |
| <b>Precondition</b>             | -  |
| <b>The called functions</b>     | timer_external_trigger_config  |
| Input parameter{in}             |  |
| <b>timer_periph</b>             | TIMER peripheral   |
| <b>TIMERx(x=0..4, 7, 8, 11)</b> | TIMER peripheral selection   |
| Input parameter{in}             |  |
| <b>extprescaler</b>             | ETI external trigger prescaler   |
| <b>TIMER_EXT_TRI_PSC_OFF</b>    | no divided   |
| <b>TIMER_EXT_TRI_PSC_DIV2</b>   | divided by 2   |
| <b>TIMER_EXT_TRI_PSC_DIV4</b>   | divided by 4   |
| <b>TIMER_EXT_TRI_PSC_DIV8</b>   | divided by 8   |
| Input parameter{in}             |  |
| <b>expolarity</b>               | ETI external trigger polarity  |
| <b>TIMER_ETP_FALLING</b>        | active low or falling edge active  |
| <b>TIMER_ETP_RISING</b>         | active high or rising edge active  |
| Input parameter{in}             |  |
| <b>extfilter</b>                | ETI external trigger filter control(0~15)  |
| Output parameter{out}           |  |

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

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 3-796. 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, uint8_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 3-797. Function timer\_external\_clock\_mode1\_disable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_external_clock_mode1_disable                              |
| <b>Function prototype</b>    | void timer_external_clock_mode1_disable(uint32_t timer_periph); |
| <b>Function descriptions</b> | disable TIMER the external clock mode1                          |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| timer_periph                 | TIMER peripheral  |
| TIMERx(x=0..4, 7)            | TIMER peripheral selection                                      |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable TIMER0 the external clock mode1 */

timer_external_clock_mode1_disable(TIMER0);
```

### **timer\_flag\_get**

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

**Table 3-798. 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>   |   |
| timer_periph                 | TIMER peripheral  |
| TIMERx                       | TIMER peripheral selection (x please refer to the following parameters) |
| <b>Input parameter{in}</b>   |   |
| flag                         | the timer interrupt flags   |
| TIMER_FLAG_UP                | update flag, TIMERx(x=0..13)  |

|                              |  |
|------------------------------|--|
| <i>TIMER_FLAG_CH0</i>        | channel 0 flag, TIMERx(x=0..4, 7..13)                |
| <i>TIMER_FLAG_CH1</i>        | channel 1 flag, TIMERx(x=0..4, 7, 8, 11)             |
| <i>TIMER_FLAG_CH2</i>        | channel 2 flag, TIMERx(x=0..4, 7)                    |
| <i>TIMER_FLAG_CH3</i>        | channel 3 flag, TIMERx(x=0..4, 7)                    |
| <i>TIMER_FLAG_CMT</i>        | channel commutation flag, TIMERx(x=0, 7)             |
| <i>TIMER_FLAG_TRG</i>        | trigger flag, TIMERx(x=0, 7, 8, 11)                  |
| <i>TIMER_FLAG_BRK</i>        | break flag, TIMERx(x=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>          |  |
| <b>FlagStatus</b>            | SET or RESET   |

Example:

```
/* get TIMER0 update flags */
FlagStatus Flag_status = RESET;
Flag_status = timer_flag_get(TIMER0, TIMER_FLAG_UP);
```

### **timer\_flag\_clear**

The description of `timer_flag_clear` is shown as below:

**Table 3-799. Function `timer_flag_clear`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | <code>timer_flag_clear</code>   |
| <b>Function prototype</b>    | <code>void timer_flag_clear(uint32_t timer_periph, uint32_t flag);</code> |
| <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>                | TIMER peripheral selection (x please refer to the following parameters)   |
| <b>Input parameter{in}</b>   |   |
| <b>flag</b>                  | the timer interrupt flags   |
| <i>TIMER_FLAG_UP</i>         | update flag, TIMERx(x=0..13)  |
| <i>TIMER_FLAG_CH0</i>        | channel 0 flag, TIMERx(x=0..4, 7..13)                                     |
| <i>TIMER_FLAG_CH1</i>        | channel 1 flag, TIMERx(x=0..4, 7, 8, 11)                                  |
| <i>TIMER_FLAG_CH2</i>        | channel 2 flag, TIMERx(x=0..4, 7)   |
| <i>TIMER_FLAG_CH3</i>        | channel 3 flag, TIMERx(x=0..4, 7)   |
| <i>TIMER_FLAG_CMT</i>        | channel commutation flag, TIMERx(x=0, 7)                                  |

|                              |  |
|------------------------------|--|
| <i>TIMER_FLAG_TRG</i>        | trigger flag, TIMERx(x=0, 7, 8, 11)                  |
| <i>TIMER_FLAG_BRK</i>        | break flag, TIMERx(x=0, 7)                           |
| <i>TIMER_FLAG_CH00</i>       | channel 0 overcapture flag, TIMERx(x=0..4, 7..11)    |
| <i>TIMER_FLAG_CH10</i>       | channel 1 overcapture flag, TIMERx(x=0..4, 7, 8, 11) |
| <i>TIMER_FLAG_CH20</i>       | channel 2 overcapture flag, TIMERx(x=0..4, 7)        |
| <i>TIMER_FLAG_CH30</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);
```

### **timer\_interrupt\_enable**

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

**Table 3-800. 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>                | TIMER peripheral selection (x please refer to the following parameters) |
| <b>Input parameter{in}</b>   |   |
| <b>interrupt</b>             | timer interrupt enable source   |
| <i>TIMER_INT_UP</i>          | update interrupt enable, TIMERx(x=0..13)                                |
| <i>TIMER_INT_CH0</i>         | channel 0 interrupt enable, TIMERx(x=0..4, 7..13)                       |
| <i>TIMER_INT_CH1</i>         | channel 1 interrupt enable, TIMERx(x=0..4, 7, 8, 11)                    |
| <i>TIMER_INT_CH2</i>         | channel 2 interrupt enable, TIMERx(x=0..4, 7)                           |
| <i>TIMER_INT_CH3</i>         | channel 3 interrupt enable , TIMERx(x=0..4, 7)                          |
| <i>TIMER_INT_CMT</i>         | commutation interrupt enable, TIMERx(x=0, 7)                            |
| <i>TIMER_INT_TRG</i>         | trigger interrupt enable, TIMERx(x=0..4, 7, 8, 11)                      |
| <i>TIMER_INT_BRK</i>         | break interrupt enable, TIMERx(x=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 3-801. Function timer\_interrupt\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | timer_interrupt_disable  |
| <b>Function prototype</b>    | void timer_interrupt_disable(uint32_t timer_periph, uint32_t interrupt); |
| <b>Function descriptions</b> | disable the TIMER interrupt  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| timer_periph                 | TIMER peripheral   |
| TIMERx                       | TIMER peripheral selection (x please refer to the following parameters)  |
| <b>Input parameter{in}</b>   |  |
| interrupt                    | timer interrupt enable source  |
| TIMER_INT_UP                 | update interrupt enable, TIMERx(x=0..13)                                 |
| TIMER_INT_CH0                | channel 0 interrupt enable, TIMERx(x=0..4, 7..13)                        |
| TIMER_INT_CH1                | channel 1 interrupt enable, TIMERx(x=0..4, 7, 8, 11)                     |
| TIMER_INT_CH2                | channel 2 interrupt enable, TIMERx(x=0..4, 7)                            |
| TIMER_INT_CH3                | channel 3 interrupt enable , TIMERx(x=0..4, 7)                           |
| TIMER_INT_CMT                | commutation interrupt enable, TIMERx(x=0, 7)                             |
| TIMER_INT_TRG                | trigger interrupt enable, TIMERx(x=0..4, 7, 8, 11)                       |
| TIMER_INT_BRK                | break interrupt enable, TIMERx(x=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 3-802. Function timer\_interrupt\_flag\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_interrupt_flag_get  |
| <b>Function prototype</b>    | FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t interrupt); |
| <b>Function descriptions</b> | get timer interrupt flag  |

|                                      |   |
|--------------------------------------|---|
| <b>Precondition</b>                  | -   |
| <b>The called functions</b>          | -   |
| <b>Input parameter{in}</b>           |   |
| <b>timer_periph</b>                  | TIMER peripheral  |
| <b>TIMERx</b>                        | TIMER peripheral selection (x please refer to the following parameters) |
| <b>Input parameter{in}</b>           |   |
| <b>interrupt</b>                     | the timer interrupt bits  |
| <b>TIMER_INT_FLAG_UP</b>             | update interrupt flag, TIMERx(x=0..13)                                  |
| <b>TIMER_INT_FLAG_CH0</b>            | channel 0 interrupt flag, TIMERx(x=0..4, 7..13)                         |
| <b>TIMER_INT_FLAG_CH1</b>            | channel 1 interrupt flag, TIMERx(x=0..4, 7, 8, 11)                      |
| <b>TIMER_INT_FLAG_CH2</b>            | channel 2 interrupt flag, TIMERx(x=0..4, 7)                             |
| <b>TIMER_INT_FLAG_CH3</b>            | channel 3 interrupt flag, TIMERx(x=0..4, 7)                             |
| <b>TIMER_INT_FLAG_CM<sub>T</sub></b> | channel commutation interrupt flag, TIMERx(x=0, 7)                      |
| <b>TIMER_INT_FLAG_TRG</b>            | trigger interrupt flag, TIMERx(x=0, 7, 8, 11)                           |
| <b>TIMER_INT_FLAG_BRK</b>            | break interrupt flag, TIMERx(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 3-803. Function `timer_interrupt_flag_clear`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | timer_interrupt_flag_clear  |
| <b>Function prototype</b>    | void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t interrupt); |
| <b>Function descriptions</b> | clear TIMER interrupt flag  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>timer_periph</b>          | TIMER peripheral  |
| <b>TIMERx</b>                | TIMER peripheral selection (x please refer to the following parameters)     |
| <b>Input parameter{in}</b>   |   |
| <b>interrupt</b>             | the timer interrupt bits  |
| <b>TIMER_INT_FLAG_UP</b>     | update interrupt flag, TIMERx(x=0..13)                                      |
| <b>TIMER_INT_FLAG_CH0</b>    | channel 0 interrupt flag, TIMERx(x=0..4, 7..13)                             |

|                              |  |
|------------------------------|--|
| <i>TIMER_INT_FLAG_CH1</i>    | channel 1 interrupt flag, TIMERx(x=0..4, 7, 8, 11) |
| <i>TIMER_INT_FLAG_CH2</i>    | channel 2 interrupt flag, TIMERx(x=0..4, 7)        |
| <i>TIMER_INT_FLAG_CH3</i>    | channel 3 interrupt flag, TIMERx(x=0..4, 7)        |
| <i>TIMER_INT_FLAG_CM_T</i>   | channel commutation interrupt flag, TIMERx(x=0, 7) |
| <i>TIMER_INT_FLAG_TRG</i>    | trigger interrupt flag, TIMERx(x=0, 7, 8, 11)      |
| <i>TIMER_INT_FLAG_BRK</i>    | break interrupt flag, TIMERx(x=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);
```

## 3.26. TLI

The TFT(LCD) display interface provides a parallel digital RGB (Red, Green, Blue) and signals for horizontal, vertical synchronization, Pixel Clock and Data Enable as output to interface directly to a variety of LCD(Liquid Crystal Display) and TFT(Thin Film Transistor) panels. The TLI registers are listed in chapter [3.26.1](#), the TLI firmware functions are introduced in chapter [3.26.2](#).

### 3.26.1. Descriptions of Peripheral registers

TLI registers are listed in the table shown as below:

**Table 3-804. TLI Registers**

| Registers | Descriptions                        |
|-----------|-------------------------------------|
| TLI_SPSZ  | TLI synchronous pulse size register |
| TLI_BPSZ  | TLI back-porch size register        |
| TLI_ASZ   | TLI active size register            |
| TLI_TSZ   | TLI total size register             |
| TLI_CTL   | TLI control register                |
| TLI_RL    | TLI reload layer register           |
| TLI_BGC   | TLI background color register       |
| TLI_INTEN | TLI interrupt enable register       |
| TLI_INTF  | TLI interrupt flag register         |
| TLI_INTC  | TLI interrupt flag clear register   |
| TLI_LM    | TLI line mark register              |
| TLI_CPPOS | TLI current pixel position register |

| Registers    | Descriptions  |
|--------------|---|
| TLI_STAT     | TLI status register                                 |
| TLI_LxCTL    | TLI layer x control register                        |
| TLI_LxHPOS   | TLI layer x horizontal position parameters register |
| TLI_LxVPOS   | TLI layer x vertical position parameters register   |
| TLI_LxCKEY   | TLI layer x color key register                      |
| TLI_LxPPF    | TLI layer x packeted pixel format register          |
| TLI_LxSA     | TLI layer x specified alpha register                |
| TLI_LxDC     | TLI layer x default color register                  |
| TLI_LxBLEND  | TLI layer x blending register                       |
| TLI_LxFBADDR | TLI layer x frame base address register             |
| TLI_LxFLLEN  | TLI layer x frame line length register              |
| TLI_LxFTLN   | TLI layer x frame total line number register        |
| TLI_LxLUT    | TLI layer x look up table register                  |

### 3.26.2. Descriptions of Peripheral functions

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

**Table 3-805. TLI firmware function**

| Function name                  | Function description                     |
|--------------------------------|--|
| tli_deinit                     | deinitialize TLI registers               |
| tli_init                       | initialize TLI display timing parameters |
| tli_dither_config              | configure TLI dither function            |
| tli_enable                     | enable TLI                               |
| tli_disable                    | disable TLI                              |
| tli_reload_config              | configure TLI reload mode                |
| tli_line_mark_set              | set line mark value                      |
| tli_current_pos_get            | get current displayed position           |
| tli_layer_enable               | TLI layer enable                         |
| tli_layer_disable              | TLI layer disable                        |
| tli_color_key_enable           | TLI layer color keying enable            |
| tli_color_key_disable          | TLI layer color keying disable           |
| tli_lut_enable                 | TLI layer LUT enable                     |
| tli_lut_disable                | TLI layer LUT disable                    |
| tli_layer_init                 | TLI layer initialize                     |
| tli_layer_window_offset_modify | reconfigure window position              |
| tli_lut_init                   | TLI layer LUT initialize                 |
| tli_ckey_init                  | TLI layer key initialize                 |
| tli_flag_get                   | get TLI flag or state                    |
| tli_interrupt_enable           | enable TLI interrupt                     |
| tli_interrupt_disable          | disable TLI interrupt                    |
| tli_interrupt_flag_get         | get TLI interrupt flag                   |

| Function name            | Function description     |
|--------------------------|--------------------------|
| tli_interrupt_flag_clear | clear TLI interrupt flag |

### Structure tli\_parameter\_struct

**Table 3-806. Structure tli\_parameter\_struct**

| Member name            | Function description   |
|------------------------|--|
| synpsz_vpsz            | size of the vertical synchronous pulse   |
| synpsz_hpsz            | size of the horizontal synchronous pulse                                       |
| backpsz_vbpsz          | size of the vertical back porch plus synchronous pulse                         |
| backpsz_hbpsz          | size of the horizontal back porch plus synchronous pulse                       |
| activesz_vasz          | size of the vertical active area width plus back porch and synchronous pulse   |
| activesz_hasz          | size of the horizontal active area width plus back porch and synchronous pulse |
| totalsz_vtsz           | vertical total size of the display   |
| totalsz_htsz           | horizontal total size of the display   |
| backcolor_red          | background value red   |
| backcolor_green        | background value green   |
| backcolor_blue         | background value blue  |
| signalpolarity_hs      | horizontal pulse polarity selection  |
| signalpolarity_vs      | vertical pulse polarity selection  |
| signalpolarity_de      | data enable polarity selection   |
| signalpolarity_pixelck | pixel clock polarity selection   |

### Structure tli\_layer\_parameter\_struct

**Table 3-807. Structure tli\_layer\_parameter\_struct**

| Member name                   | Function description                          |
|-------------------------------|---|
| layer_window_rightpos         | window right position                         |
| layer_window_leftpos          | window left position                          |
| layer_window_bottompos        | window bottom position                        |
| layer_window_toppos           | window top position                           |
| layer_ppf                     | packetized pixel format                       |
| layer_sa                      | specified alpha                               |
| layer_default_alpha           | the default color alpha                       |
| layer_default_red             | the default color red                         |
| layer_default_green           | the default color green                       |
| layer_default_blue            | the default color blue                        |
| layer_acf1                    | alpha calculation factor 1 of blending method |
| layer_acf2                    | alpha calculation factor 2 of blending method |
| layer_frame_bufaddr           | frame buffer base address                     |
| layer_frame_buf_stride_offset | frame buffer stride offset                    |

|                               |                         |
|-------------------------------|-------------------------|
| layer_frame_line_length       | frame line length       |
| layer_frame_total_line_number | frame total line number |

### Structure tli\_layer\_lut\_parameter\_struct

**Table 3-808. Structure tli\_layer\_lut\_parameter\_struct**

| Member name             | Function description         |
|-------------------------|------------------------------|
| layer_table_addr        | look up table write address  |
| layer_lut_channel_red   | red channel of a LUT entry   |
| layer_lut_channel_green | green channel of a LUT entry |
| layer_lut_channel_blue  | blue channel of a LUT entry  |

### Enumeration tli\_layer\_ppf\_enum

**Table 3-809. Enumeration tli\_layer\_ppf\_enum**

| Member name        | Function description                  |
|--------------------|---------------------------------------|
| LAYER_PPF_ARGB8888 | layerx packeted pixel format ARGB8888 |
| LAYER_PPF_RGB888   | layerx packeted pixel format RGB888   |
| LAYER_PPF_RGB565   | layerx packeted pixel format RGB565   |
| LAYER_PPF_ARGB1555 | layerx packeted pixel format ARGB1555 |
| LAYER_PPF_ARGB4444 | layerx packeted pixel format ARGB4444 |
| LAYER_PPF_L8       | layerx packeted pixel format L8       |
| LAYER_PPF_AL44     | layerx packeted pixel format AL44     |
| LAYER_PPF_AL88     | layerx packeted pixel format AL88     |

### tli\_deinit

The description of tli\_deinit is shown as below:

**Table 3-810. Function tli\_deinit**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | tli_deinit                                   |
| <b>Function prototype</b>    | void tli_deinit (void);                      |
| <b>Function descriptions</b> | deinitialize TLI registers                   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | rcu_bkp_reset_enable / rcu_bkp_reset_disable |
| <b>Input parameter{in}</b>   |  |
| -                            | -  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* deinitialize TLI registers */
```

```
tli_deinit();
```

### **tli\_init**

The description of tli\_init is shown as below:

**Table 3-811. Function tli\_init**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | tli_init   |
| <b>Function prototype</b>    | void tli_init(tli_parameter_struct *tli_struct); |
| <b>Function descriptions</b> | initialize TLI display timing parameters         |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>tli_struct</b>            | the data needed to initialize TLI                |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* initialize TLI display timing parameters */

tli_parameter_struct tli_init_struct;

/* TLI initialization */

tli_init_struct.signalpolarity_hs = TLI_HSYN_ACTLIVE_LOW;
tli_init_struct.signalpolarity_vs = TLI_VSYN_ACTLIVE_LOW;
tli_init_struct.signalpolarity_de = TLI_DE_ACTLIVE_LOW;
tli_init_struct.signalpolarity_pixelck = TLI_PIXEL_CLOCK_TLI;

/* LCD display timing configuration */

tli_init_struct.synpsz_hpsz = 40;
tli_init_struct.synpsz_vpsz = 9;
tli_init_struct.backpsz_hbpsz = 42;
tli_init_struct.backpsz_vbpsz = 11;
tli_init_struct.activesz_hasz = 42 + 480;
tli_init_struct.activesz_vasz = 11 + 272;
tli_init_struct.totalsz_htsz = 42 + 480 + 2;
tli_init_struct.totalsz_vtsz = 11 + 272 + 2;
```

```

/* LCD background color configure */

tli_init_struct.backcolor_red = 0xFF;
tli_init_struct.backcolor_green = 0xFF;
tli_init_struct.backcolor_blue = 0xFF;
tli_init (&tli_init_struct);

```

### **tli\_dither\_config**

The description of tli\_dither\_config is shown as below:

**Table 3-812. Function tli\_dither\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | tli_dither_config                           |
| <b>Function prototype</b>    | void tli_dither_config(uint8_t ditherstat); |
| <b>Function descriptions</b> | configure TLI dither function               |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| ditherstat                   | Dither status                               |
| <i>TLI_DITHER_ENABLE</i>     | TLI dither enable                           |
| <i>TLI_DITHER_DISABLE</i>    | TLI dither disable                          |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```

/* configure TLI dither function */

tli_dither_config(TLI_DITHER_ENABLE);

```

### **tli\_enable**

The description of tli\_enable is shown as below:

**Table 3-813. Function tli\_enable**

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

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

Example:

```
/* enable TLI */

tli_enable( );
```

### tli\_disable

The description of tli\_disable is shown as below:

**Table 3-814. Function tli\_disable**

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

Example:

```
/* disable TLI */

tli_disable( );
```

### tli\_reload\_config

The description of tli\_reload\_config is shown as below:

**Table 3-815. Function tli\_reload\_config**

|                                       |  |
|---------------------------------------|--|
| <b>Function name</b>                  | tli_reload_config  |
| <b>Function prototype</b>             | void tli_reload_config(uint8_t reloadmode);                  |
| <b>Function descriptions</b>          | configure TLI reload mode                                    |
| <b>Precondition</b>                   | -  |
| <b>The called functions</b>           | -  |
| <b>Input parameter{in}</b>            |  |
| <b>reloadmode</b>                     | Reload mode  |
| <b>TLI_FRAME_BLANK_R<br/>ELOAD_EN</b> | the layer configuration will be reloaded at frame blank      |
| <b>TLI_REQUEST_RELO</b>               | the layer configuration will be reloaded after this bit sets |

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

Example:

```
/* configure TLI reload mode */

tli_reload_config (TLI_FRAME_BLANK_RELOAD_EN);
```

### **tli\_line\_mark\_set**

The description of tli\_line\_mark\_set is shown as below:

**Table 3-816. Function tli\_line\_mark\_set**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | tli_line_mark_set                        |
| <b>Function prototype</b>    | void tli_line_mark_set(uint32_t linenum) |
| <b>Function descriptions</b> | set line mark value                      |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>linenum</b>               | line number                              |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* set line mark value */

tli_line_mark_set( 0x40);
```

### **tli\_current\_pos\_get**

The description of tli\_current\_pos\_get is shown as below:

**Table 3-817. Function tli\_current\_pos\_get**

|                              |                                     |
|------------------------------|-------------------------------------|
| <b>Function name</b>         | tli_current_pos_get                 |
| <b>Function prototype</b>    | uint32_t tli_current_pos_get(void); |
| <b>Function descriptions</b> | get current displayed position      |
| <b>Precondition</b>          | -                                   |
| <b>The called functions</b>  | -                                   |
| <b>Input parameter{in}</b>   |                                     |
| -                            | -                                   |
| <b>Output parameter{out}</b> |                                     |

|                     |                  |
|---------------------|------------------|
| -                   | -                |
| <b>Return value</b> |                  |
| uint32_t            | 0x0 – 0xFFFFFFFF |

Example:

```
/* get current displayed position */
uint32_t pos = tli_current_pos_get( );
```

### tli\_layer\_enable

The description of tli\_layer\_enable is shown as below:

**Table 3-818. Function tli\_layer\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | tli_layer_enable                        |
| <b>Function prototype</b>    | void tli_layer_enable(uint32_t layerx); |
| <b>Function descriptions</b> | TLI layer enable                        |
| <b>Precondition</b>          | -                                       |
| <b>The called functions</b>  | -                                       |
| <b>Input parameter{in}</b>   |   |
| <b>layerx</b>                | Layer base address                      |
| <i>LAYER0</i>                | Layer0 base address                     |
| <i>LAYER1</i>                | Layer1 base address                     |
| <b>Output parameter{out}</b> |   |
| -                            | -                                       |
| <b>Return value</b>          |   |
| -                            | -                                       |

Example:

```
/* TLI layer enable */
tli_layer_enable(LAYER0);
```

### tli\_layer\_disable

The description of tli\_layer\_disable is shown as below:

**Table 3-819. Function tli\_layer\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | tli_layer_disable                        |
| <b>Function prototype</b>    | void tli_layer_disable(uint32_t layerx); |
| <b>Function descriptions</b> | TLI layer disable                        |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>layerx</b>                | Layer base address                       |
| <i>LAYER0</i>                | Layer0 base address                      |

|                              |                     |
|------------------------------|---------------------|
| LAYER1                       | Layer1 base address |
| <b>Output parameter{out}</b> |                     |
| -                            | -                   |
| <b>Return value</b>          |                     |
| -                            | -                   |

Example:

```
/* TLI layer disable */

tli_layer_disable(LAYER0);
```

### **tli\_color\_key\_enable**

The description of tli\_color\_key\_enable is shown as below:

**Table 3-820. Function tli\_color\_key\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | tli_color_key_enable                        |
| <b>Function prototype</b>    | void tli_color_key_enable(uint32_t layerx); |
| <b>Function descriptions</b> | TLI layer color keying enable               |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| layerx                       | Layer base address                          |
| LAYER0                       | Layer0 base address                         |
| LAYER1                       | Layer1 base address                         |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* TLI layer color keying enable */

tli_color_key_enable(LAYER0);
```

### **tli\_color\_key\_disable**

The description of tli\_color\_key\_disable is shown as below:

**Table 3-821. Function tli\_color\_key\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | tli_color_key_disable                        |
| <b>Function prototype</b>    | void tli_color_key_disable(uint32_t layerx); |
| <b>Function descriptions</b> | TLI layer color keying disable               |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |

|                              |                     |
|------------------------------|---------------------|
| <b>layerx</b>                | Layer base address  |
| <i>LAYER0</i>                | Layer0 base address |
| <i>LAYER1</i>                | Layer1 base address |
| <b>Output parameter{out}</b> |                     |
| -                            | -                   |
| <b>Return value</b>          |                     |
| -                            | -                   |

Example:

```
/* TLI layer color keying disable */

tli_color_key_disable(LAYER0);
```

### tli\_lut\_enable

The description of tli\_lut\_enable is shown as below:

**Table 3-822. Function tli\_lut\_enable**

|                              |                                       |
|------------------------------|---------------------------------------|
| <b>Function name</b>         | tli_lut_enable                        |
| <b>Function prototype</b>    | void tli_lut_enable(uint32_t layerx); |
| <b>Function descriptions</b> | TLI layer LUT enable                  |
| <b>Precondition</b>          | -                                     |
| <b>The called functions</b>  | -                                     |
| <b>Input parameter{in}</b>   |                                       |
| <b>layerx</b>                | Layer base address                    |
| <i>LAYER0</i>                | Layer0 base address                   |
| <i>LAYER1</i>                | Layer1 base address                   |
| <b>Output parameter{out}</b> |                                       |
| -                            | -                                     |
| <b>Return value</b>          |                                       |
| -                            | -                                     |

Example:

```
/* TLI layer LUT enable */

tli_lut_enable(LAYER0);
```

### tli\_lut\_disable

The description of tli\_lut\_disable is shown as below:

**Table 3-823. Function tli\_lut\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | tli_lut_disable                        |
| <b>Function prototype</b>    | void tli_lut_disable(uint32_t layerx); |
| <b>Function descriptions</b> | TLI layer LUT disable                  |
| <b>Precondition</b>          | -                                      |

|                              |                     |
|------------------------------|---------------------|
| <b>The called functions</b>  | -                   |
| <b>Input parameter{in}</b>   |                     |
| <b>layerx</b>                | Layer base address  |
| <i>LAYER0</i>                | Layer0 base address |
| <i>LAYER1</i>                | Layer1 base address |
| <b>Output parameter{out}</b> |                     |
| -                            | -                   |
| <b>Return value</b>          |                     |
| -                            | -                   |

Example:

```
/* TLI layer LUT disable */
tli_lut_disable(LAYER0);
```

### tli\_layer\_init

The description of tli\_layer\_init is shown as below:

**Table 3-824. Function tli\_layer\_init**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | tli_layer_init  |
| <b>Function prototype</b>    | void tli_layer_init(uint32_t layerx,tli_layer_parameter_struct *layer_struct) |
| <b>Function descriptions</b> | TLI layer initialize  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>layerx</b>                | Layer base address  |
| <i>LAYER0</i>                | Layer0 base address   |
| <i>LAYER1</i>                | Layer1 base address   |
| <b>Input parameter{in}</b>   |   |
| <b>layer_struct</b>          | TLI Layer parameter struct  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* TLI layer initialize */
tli_layer_parameter_struct tli_layer_init_struct;
/* tli layer1 configuration */
/* tli window size configuration */
tli_layer_init_struct.layer_window_leftpos = 20 + 43;
```

```

tli_layer_init_struct.layer_window_rightpos = (20 + 140 + 43 - 1);

tli_layer_init_struct.layer_window_toppos = 30 + 12;

tli_layer_init_struct.layer_window_bottompos = (30 + 60 + 12 - 1);

/* tli window pixel format configuration */

tli_layer_init_struct.layer_ppf = LAYER_PPF_RGB565;

/* tli window specified alpha configuration */

tli_layer_init_struct.layer_sa = 255;

/* tli window blend configuration */

tli_layer_init_struct.layer_acf1 = LAYER_ACF1_PASA;

tli_layer_init_struct.layer_acf2 = LAYER_ACF2_PASA;

/* tli layer default alpha R,G,B value configuration */

tli_layer_init_struct.layer_default_alpha = 0;

tli_layer_init_struct.layer_default_blue = 0xFF;

tli_layer_init_struct.layer_default_green = 0xFF;

tli_layer_init_struct.layer_default_red = 0xFF;

/* tli layer frame buffer base address configuration */

tli_layer_init_struct.layer_frame_bufaddr = (uint32_t)&image_0;

tli_layer_init_struct.layer_frame_line_length = ((480 * 2) + 3);

tli_layer_init_struct.layer_frame_buf_stride_offset = (480 * 2);

tli_layer_init_struct.layer_frame_total_line_number = 60;

tli_layer_init(LAYER1, &tli_layer_init_struct);

```

### **tli\_layer\_window\_offset\_modify**

The description of tli\_layer\_window\_offset\_modify is shown as below:

**Table 3-825. Function tli\_layer\_window\_offset\_modify**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | tli_layer_window_offset_modify   |
| <b>Function prototype</b>    | void tli_layer_window_offset_modify(uint32_t layerx,uint32_t offset_x,uint32_t offset_y) |
| <b>Function descriptions</b> | reconfigure window position  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |

|                              |                       |
|------------------------------|-----------------------|
| <b>layerx</b>                | Layer base address    |
| <i>LAYER0</i>                | Layer0 base address   |
| <i>LAYER1</i>                | Layer1 base address   |
| <b>Input parameter{in}</b>   |                       |
| <i>offset_x</i>              | new horizontal offset |
| <b>Input parameter{in}</b>   |                       |
| <i>offset_y</i>              | new vertical offset   |
| <b>Output parameter{out}</b> |                       |
| -                            | -                     |
| <b>Return value</b>          |                       |
| -                            | -                     |

Example:

```
/* TLI layer initialize */

tli_layer_window_offset_modify(LAYER0, 0x40 , 0x40);
```

### tli\_lut\_init

The description of tli\_lut\_init is shown as below:

**Table 3-826. Function tli\_lut\_init**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | tli_lut_init  |
| <b>Function prototype</b>    | void tli_lut_init(uint32_t layerx,tli_layer_lut_parameter_struct *lut_struct) |
| <b>Function descriptions</b> | TLI layer LUT initialize  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>layerx</b>                | Layer base address  |
| <i>LAYER0</i>                | Layer0 base address   |
| <i>LAYER1</i>                | Layer1 base address   |
| <b>Input parameter{in}</b>   |   |
| <b>lut_struct</b>            | TLI layer LUT parameter struct  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* TLI layer initialize */

tli_layer_lut_parameter_struct lut_struct;
lut_struct.layer_table_addr = 0x20000400;
lut_struct.layer_lut_channel_red = 0x20;
```

```

lut_struct.layer_lut_channel_green = 0x40;
lut_struct.layer_lut_channel_blue = 0x40;
tli_layer_init(LAYER0, &lut_struct);
  
```

### **tli\_ckey\_init**

The description of tli\_ckey\_init is shown as below:

**Table 3-827. Function tli\_ckey\_init**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | tli_ckey_init  |
| <b>Function prototype</b>    | void tli_ckey_init(uint32_t layerx,uint32_t redkey,uint32_t greenkey,uint32_t bluekey) |
| <b>Function descriptions</b> | TLI layer key initialize   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>layerx</b>                | Layer base address   |
| <b>LAYER0</b>                | Layer0 base address  |
| <b>LAYER1</b>                | Layer1 base address  |
| <b>Input parameter{in}</b>   |  |
| <b>redkey</b>                | color key red  |
| <b>Input parameter{in}</b>   |  |
| <b>greenkey</b>              | color key green  |
| <b>Input parameter{in}</b>   |  |
| <b>bluekey</b>               | color key blue   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```

/* TLI layer key initialize */

tli_ckey_init(LAYER0, 0xAA, 0xFF, 0x00);
  
```

### **tli\_flag\_get**

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

**Table 3-828. Function tli\_flag\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | tli_flag_get                            |
| <b>Function prototype</b>    | FlagStatus tli_flag_get(uint32_t flag); |
| <b>Function descriptions</b> | get TLI flag or state                   |
| <b>Precondition</b>          | -                                       |

|                              |   |
|------------------------------|---|
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>flag</b>                  | TLI flags                                   |
| <i>TLI_FLAG_VDE</i>          | current VDE state                           |
| <i>TLI_FLAG_HDE</i>          | current HDE state                           |
| <i>TLI_FLAG_VS</i>           | current vs state                            |
| <i>TLI_FLAG_HS</i>           | current hs state                            |
| <i>TLI_FLAG_LM</i>           | line mark interrupt flag                    |
| <i>TLI_FLAG_FE</i>           | FIFO error interrupt flag                   |
| <i>TLI_FLAG_TE</i>           | transaction error interrupt flag            |
| <i>TLI_FLAG_LCR</i>          | layer configuration reloaded interrupt flag |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>FlagStatus</b>            | SET or RESET                                |

Example:

```
/* get TLI flag state */

FlagStatus status;

status = tli_flag_get (TLI_FLAG_VDE);
```

### tli\_interrupt\_enable

The description of `tli_interrupt_enable` is shown as below:

**Table 3-829. Function `tli_interrupt_enable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | tli_interrupt_enable                          |
| <b>Function prototype</b>    | void tli_interrupt_enable(uint32_t interrupt) |
| <b>Function descriptions</b> | enable TLI interrupt                          |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>intflag</b>               | TLI interrupt flags                           |
| <i>TLI_INT_LM</i>            | line mark interrupt                           |
| <i>TLI_INT_FE</i>            | FIFO error interrupt                          |
| <i>TLI_INT_TE</i>            | transaction error interrupt                   |
| <i>TLI_INT_LCR</i>           | layer configuration reloaded interrupt        |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable TLI interrupt */

tli_interrupt_enable (TLI_INT_LM);
```

### tli\_interrupt\_disable

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

**Table 3-830. Function tli\_interrupt\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | tli_interrupt_disable                          |
| <b>Function prototype</b>    | void tli_interrupt_disable(uint32_t interrupt) |
| <b>Function descriptions</b> | disable TLI interrupt                          |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| intflag                      | TLI interrupt flags                            |
| TLI_INT_LM                   | line mark interrupt                            |
| TLI_INT_FE                   | FIFO error interrupt                           |
| TLI_INT_TE:                  | transaction error interrupt                    |
| TLI_INT_LCR                  | layer configuration reloaded interrupt         |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable TLI interrupt */

tli_interrupt_disable (TLI_INT_LM);
```

### tli\_interrupt\_flag\_get

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

**Table 3-831. Function tli\_interrupt\_flag\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | tli_interrupt_flag_get                                |
| <b>Function prototype</b>    | FlagStatus tli_interrupt_flag_get(uint32_t int_flag); |
| <b>Function descriptions</b> | get TLI interrupt flag                                |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| int_flag                     | TLI interrupt flags                                   |
| TLI_INT_FLAG_LM              | line mark interrupt flag                              |
| TLI_INT_FLAG_FE              | FIFO error interrupt flag                             |
| TLI_INT_FLAG_TE              | transaction error interrupt flag                      |
| TLI_INT_FLAG_LCR:            | layer configuration reloaded interrupt flag           |

| Output parameter{out} |              |
|-----------------------|--------------|
| -                     | -            |
| Return value          |              |
| FlagStatus            | SET or RESET |

Example:

```
/* get TLI flag state */

FlagStatus status;

status = tli_interrupt_flag_get (TLI_INT_FLAG_LM);
```

### tli\_interrupt\_flag\_clear

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

**Table 3-832. Function tli\_interrupt\_flag\_clear**

| Function name         | tli_interrupt_flag_clear                         |  |
|-----------------------|--|--|
| Function prototype    | void tli_interrupt_flag_clear(uint32_t int_flag) |  |
| Function descriptions | clear TLI interrupt flag                         |  |
| Precondition          | -  |  |
| The called functions  | -  |  |
| Input parameter{in}   |  |  |
| int_flag              | TLI interrupt flags                              |  |
| TLI_INT_FLAG_LM       | line mark interrupt flag                         |  |
| TLI_INT_FLAG_FE       | FIFO error interrupt flag                        |  |
| TLI_INT_FLAG_TE       | transaction error interrupt flag                 |  |
| TLI_INT_FLAG_LCR:     | layer configuration reloaded interrupt flag      |  |
| Output parameter{out} |  |  |
| -                     | -  |  |
| Return value          |  |  |
| -                     | -  |  |

Example:

```
/* clear TLI flag state */

tli_interrupt_flag_clear (TLI_INT_FLAG_LM);
```

## 3.27. TRNG

The true random number generator (TRNG) module can generate a 32-bit value using continuous analog noise. The TRNG registers are listed in chapter [3.27.1](#), the TRNG firmware functions are introduced in chapter [3.27.2](#).

### 3.27.1. Descriptions of Peripheral registers

TRNG registers are listed in the table shown as below:

**Table 3-833. TRNG Registers**

| Registers | Descriptions          |
|-----------|-----------------------|
| TRNG_CTL  | TRNG control register |
| TRNG_STAT | TRNG status register  |
| TRNG_DATA | TRNG data register    |

### 3.27.2. Descriptions of Peripheral functions

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

**Table 3-834. TRNG firmware function**

| Function name             | Function description           |
|---------------------------|--------------------------------|
| trng_deinit               | deinitialize the TRNG          |
| trng_enable               | enable the TRNG interface      |
| trng_disable              | disable the TRNG interface     |
| trng_get_true_random_data | get the true random data       |
| trng_flag_get             | get the trng status flags      |
| trng_interrupt_enable     | the trng interrupt enable      |
| trng_interrupt_disable    | the trng interrupt disable     |
| trng_interrupt_flag_get   | get the trng interrupt flags   |
| trng_interrupt_flag_clear | clear the trng interrupt flags |

#### Enum trng\_flag\_enum

**Table 3-835. Enumeration trng\_flag\_enum**

| Member name    | Function description       |
|----------------|----------------------------|
| TRNG_FLAG_DRDY | random data ready status   |
| TRNG_FLAG_CECS | clock error current status |
| TRNG_FLAG_SECS | seed error current status  |

#### Enum trng\_int\_flag\_enum

**Table 3-836. Enumeration trng\_int\_flag\_enum**

| Member name      | Function description       |
|------------------|----------------------------|
| TRNG_INT_FLAG_CE | clock error interrupt flag |
| TRNG_INT_FLAG_SE | seed error interrupt flag  |

#### trng\_deinit

The description of trng\_deinit is shown as below:

**Table 3-837. Function trng\_deinit**

|                              |                          |
|------------------------------|--------------------------|
| <b>Function name</b>         | trng_deinit              |
| <b>Function prototype</b>    | void trng_deinit (void); |
| <b>Function descriptions</b> | TRNG deinit              |
| <b>Precondition</b>          | -                        |
| <b>The called functions</b>  | -                        |
| <b>Input parameter{in}</b>   |                          |
| -                            | -                        |
| <b>Output parameter{out}</b> |                          |
| -                            | -                        |
| <b>Return value</b>          |                          |
| -                            | -                        |

Example:

```
/* TRNG deinit */

trng_deinit();
```

### **trng\_enable**

The description of trng\_enable is shown as below:

**Table 3-838. Function trng\_enable**

|                              |                         |
|------------------------------|-------------------------|
| <b>Function name</b>         | trng_enable             |
| <b>Function prototype</b>    | void trng_enable(void); |
| <b>Function descriptions</b> | enable the TRNG         |
| <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 TRNG interface */

trng_enable();
```

### **trng\_disable**

The description of trng\_disable is shown as below:

**Table 3-839. Function trng\_disable**

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

|                              |                          |
|------------------------------|--------------------------|
| <b>Function prototype</b>    | void trng_disable(void); |
| <b>Function descriptions</b> | disable the TRNG         |
| <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 TRNG interface */

trng_disable();
```

### trng\_get\_true\_random\_data

The description of trng\_get\_true\_random\_data is shown as below:

**Table 3-840. Function trng\_get\_true\_random\_data**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | trng_get_true_random_data                 |
| <b>Function prototype</b>    | uint32_t trng_get_true_random_data(void); |
| <b>Function descriptions</b> | get the true random data                  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| -                            | -   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| uint32_t                     | 0x0 – 0xFFFFFFFF                          |

Example:

```
/* get the true random data */

uint32_t data = trng_get_true_random_data();
```

### trng\_flag\_get

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

**Table 3-841. Function trng\_flag\_get**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | trng_flag_get                                  |
| <b>Function prototype</b>    | FlagStatus trng_flag_get(trng_flag_enum flag); |
| <b>Function descriptions</b> | get the trng status flags                      |

|                              |  |
|------------------------------|--|
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>flag</b>                  | trng status flag, refer to <a href="#">Table 3-835. Enumeration trng_flag_enum</a> |
| <i>TRNG_FLAG_DRDY</i>        | random Data ready status   |
| <i>TRNG_FLAG_CECS</i>        | clock error current status   |
| <i>TRNG_FLAG_SECS</i>        | seed error current status  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>FlagStatus</b>            | SET or RESET   |

Example:

```
/* get the trng status flags */

FlagStatus trng_flag = RESET;

trng_flag = trng_flag_get(TRNG_FLAG_DRDY);
```

### **trng\_interrupt\_enable**

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

**Table 3-842. Function trng\_interrupt\_enable**

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

Example:

```
/* enable the TRNG interrupt */

trng_interrupt_enable();
```

### **trng\_interrupt\_disable**

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

**Table 3-843. Function trng\_interrupt\_disable**

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

Example:

```
/* disable the TRNG interrupt */

trng_interrupt_disable();
```

### trng\_interrupt\_flag\_get

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

**Table 3-844. Function trng\_interrupt\_flag\_get**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | trng_interrupt_flag_get   |
| <b>Function prototype</b>    | FlagStatus trng_interrupt_flag_get(trng_int_flag_enum int_flag);                          |
| <b>Function descriptions</b> | get the trng interrupt flag   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>int_flag</b>              | trng interrupt flag, refer to <a href="#">Table 3-836. Enumeration trng_int_flag_enum</a> |
| <i>TRNG_INT_FLAG_CE</i>      | clock error interrupt flag  |
| <i>TRNG_INT_FLAG_SE</i>      | seed error interrupt flag   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| <b>FlagStatus</b>            | SET or RESET  |

Example:

```
/* get the trng interrupt flag */

FlagStatus trng_flag = RESET;

trng_flag = trng_interrupt_flag_get(TRNG_INT_FLAG_CE);
```

### trng\_interrupt\_flag\_clear

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

**Table 3-845. Function trng\_interrupt\_flag\_clear**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | trng_interrupt_flag_clear   |
| <b>Function prototype</b>    | void trng_interrupt_flag_clear(trng_int_flag_enum int_flag);                              |
| <b>Function descriptions</b> | clear the trng interrupt flag   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>int_flag</b>              | trng interrupt flag, refer to <a href="#">Table 3-836. Enumeration trng_int_flag_enum</a> |
| <i>TRNG_INT_FLAG_CE</i>      | clock error interrupt flag  |
| <i>TRNG_INT_FLAG_SE</i>      | seed error interrupt flag   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/*clear the trng interrupt flag */
trng_interrupt_flag_clear(TRNG_INT_FLAG_CE);
```

## 3.28. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter [3.28.1](#), the USART firmware functions are introduced in chapter [3.28.2](#).

### 3.28.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

**Table 3-846. 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                |

| Registers   | Descriptions              |
|-------------|---------------------------|
| USART_RT    | receiver timeout register |
| USART_STAT1 | status register 1         |

### 3.28.2. Descriptions of Peripheral functions

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

**Table 3-847. USART firmware function**

| Function name                          | Function description                                |
|--|---|
| uart_deinit                            | reset USART/UART                                    |
| uart_baudrate_set                      | configure USART/UART baud rate value                |
| uart_parity_config                     | configure USART/UART parity                         |
| uart_word_length_set                   | configure USART/UART word length                    |
| uart_stop_bit_set                      | configure USART/UART stop bit length                |
| uart_enable                            | enable USART/UART                                   |
| uart_disable                           | disable USART/UART                                  |
| uart_transmit_config                   | configure USART/UART transmitter                    |
| uart_receive_config                    | configure USART/UART receiver                       |
| uart_data_first_config                 | data is transmitted/received with the LSB/MSB first |
| uart_invert_config                     | configure USART inversion                           |
| uart_receiver_timeout_enable           | enable receiver timeout                             |
| uart_receiver_timeout_disable          | disable receiver timeout                            |
| uart_receiver_timeout_threshold_config | configure receiver timeout threshold                |
| uart_data_transmit                     | USART/UART transmit data function                   |
| uart_data_receive                      | USART/UART receive data function                    |
| uart_address_config                    | configure address of the USART/UART                 |
| uart_mute_mode_enable                  | enable mute mode                                    |
| uart_mute_mode_disable                 | disable mute mode                                   |
| uart_mute_mode_wakeup_config           | configure wakeup method in mute mode                |
| uart_lin_mode_enable                   | enable LIN mode                                     |
| uart_lin_mode_disable                  | disable LIN mode                                    |
| uart_lin_break_dectcion_length_config  | configure LIN break frame length                    |
| uart_send_break                        | send break frame                                    |
| uart_halfduplex_enable                 | enable half duplex mode                             |
| uart_halfduplex_disable                | disable half duplex mode                            |
| uart_synchronous_clock_enable          | enable CK pin in synchronous mode                   |
| uart_synchronous_clock_disable         | disable CK pin in synchronous mode                  |
| uart_synchronous_clock_config          | configure USART synchronous mode parameters         |
| uart_guard_time_config                 | configure guard time value in smartcard mode        |
| uart_smartcard_mode_enable             | enable smartcard mode                               |

| Function name                    | Function description  |
|----------------------------------|---|
| uart_smartcard_mode_disable      | disable smartcard mode  |
| uart_smartcard_mode_nack_enable  | enable NACK in smartcard mode   |
| uart_smartcard_mode_nack_disable | disable NACK in smartcard mode  |
| uart_smartcard_autoretry_config  | configure smartcard auto-retry number   |
| uart_block_length_config         | configure block length  |
| uart_irda_mode_enable            | enable IrDA mode  |
| uart_irda_mode_disable           | disable IrDA mode   |
| uart_prescaler_config            | configure the peripheral clock prescaler in USART/UART<br>IrDA low-power mode |
| uart_irda_lowpower_config        | configure IrDA low-power  |
| uart_hardware_flow_rts_config    | configure hardware flow control RTS   |
| uart_hardware_flow_cts_config    | configure hardware flow control CTS   |
| uart_dma_enable                  | enable USART/UART DMA send or receive   |
| uart_dma_disable                 | disable USART/UART DMA send or receive  |
| uart_flag_get                    | get flag in STAT0/STAT1 register  |
| uart_flag_clear                  | clear flag in STAT0/STAT1 register  |
| uart_interrupt_enable            | enable USART/UART interrupt   |
| uart_interrupt_disable           | disable USART/UART interrupt  |
| uart_interrupt_flag_get          | get USART/UART interrupt flag status  |
| uart_interrupt_flag_clear        | clear USART/UART interrupt flag in STAT0/STAT1 register                       |

### Enum `uart_flag_enum`

**Table 3-848. `uart_flag_enum`**

| Member name      | Function description       |
|------------------|----------------------------|
| USART_FLAG_CTSF  | CTS change flag            |
| USART_FLAG_LBDF  | 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_IDLEF | IDLE frame 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      |

### Enum `uart_interrupt_flag_enum`

**Table 3-849. `uart_interrupt_flag_enum`**

| Member name                            | Function description  |
|--|---|
| <code>USART_INT_FLAG_PERR</code>       | parity error interrupt flag                                 |
| <code>USART_INT_FLAG_TBE</code>        | transmitter buffer empty interrupt flag                     |
| <code>USART_INT_FLAG_TC</code>         | transmission complete interrupt flag                        |
| <code>USART_INT_FLAG_RBNE</code>       | read data buffer not empty interrupt flag                   |
| <code>USART_INT_FLAG_RBNE_ORE_R</code> | read data buffer not empty interrupt and overrun error flag |
| <code>USART_INT_FLAG_IDLE</code>       | IDLE frame detected interrupt flag                          |
| <code>USART_INT_FLAG_LBD</code>        | LIN break detected interrupt flag                           |
| <code>USART_INT_FLAG_CTS</code>        | CTS interrupt flag  |
| <code>USART_INT_FLAG_ERR_ORER</code>   | overrun error interrupt flag                                |
| <code>USART_INT_FLAG_ERR_NERR</code>   | noise error interrupt flag                                  |
| <code>USART_INT_FLAG_ERR_FERR</code>   | frame error interrupt flag                                  |
| <code>USART_INT_FLAG_EB</code>         | end of block interrupt flag                                 |
| <code>USART_INT_FLAG_RT</code>         | receive timeout interrupt flag                              |

### Enum `uart_interrupt_enum`

**Table 3-850. `uart_interrupt_enum`**

| Member name                 | Function description   |
|-----------------------------|--|
| <code>USART_INT_PERR</code> | parity error interrupt   |
| <code>USART_INT_TBE</code>  | transmitter buffer empty interrupt                               |
| <code>USART_INT_TC</code>   | transmission complete interrupt                                  |
| <code>USART_INT_RBNE</code> | read data buffer not empty interrupt and overrun error interrupt |
| <code>USART_INT_IDLE</code> | IDLE line detected interrupt                                     |
| <code>USART_INT_LBD</code>  | LIN break detected interrupt                                     |
| <code>USART_INT_CTS</code>  | CTS interrupt  |
| <code>USART_INT_ERR</code>  | error interrupt  |
| <code>USART_INT_EB</code>   | end of block interrupt   |
| <code>USART_INT_RT</code>   | receive timeout interrupt  |

### Enum `uart_invert_enum`

**Table 3-851. `uart_invert_enum`**

| Member name                      | Function description         |
|----------------------------------|------------------------------|
| <code>USART_DINV_ENABLE</code>   | data bit level inversion     |
| <code>USART_DINV_DISABLE</code>  | data bit level not inversion |
| <code>USART_TXPIN_ENABLE</code>  | TX pin level inversion       |
| <code>USART_TXPIN_DISABLE</code> | TX pin level not inversion   |
| <code>USART_RXPIN_ENABLE</code>  | RX pin level inversion       |

| Member name         | Function description       |
|---------------------|----------------------------|
| USART_RXPIN_DISABLE | RX pin level not inversion |

### uart\_deinit

The description of `uart_deinit` is shown as below:

**Table 3-852. Function `uart_deinit`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | uart_deinit   |
| <b>Function prototype</b>    | <code>void usart_deinit(uint32_t usart_periph);</code>          |
| <b>Function descriptions</b> | reset USART/UART  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | <code>rcu_periph_reset_enable / rcu_periph_reset_disable</code> |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | uart peripheral   |
| <i>USARTx</i>                | $x=0,1,2,5$   |
| <i>UARTx</i>                 | $x=3,4,6,7$   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* reset USART0 */

uart_deinit(USART0);
```

### uart\_baudrate\_set

The description of `uart_baudrate_set` is shown as below:

**Table 3-853. Function `uart_baudrate_set`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | uart_baudrate_set  |
| <b>Function prototype</b>    | <code>void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);</code> |
| <b>Function descriptions</b> | configure USART/UART baud rate value   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | <code>rcu_clock_freq_get</code>  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | uart peripheral  |
| <i>USARTx</i>                | $x=0,1,2,5$  |
| <i>UARTx</i>                 | $x=3,4,6,7$  |
| <b>Input parameter{in}</b>   |  |
| <b>baudval</b>               | baud rate value  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |

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

Example:

```
/* configure USART0 baud rate value */
usart_baudrate_set(USART0, 115200);
```

### usart\_parity\_config

The description of usart\_parity\_config is shown as below:

**Table 3-854. 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/UART parity  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | uart peripheral  |
| <b>USARTx</b>                | x=0,1,2,5  |
| <b>UARTx</b>                 | x=3,4,6,7  |
| <b>Input parameter{in}</b>   |  |
| <b>paritycfg</b>             | USART/UART parity  |
| <b>USART_PM_NONE</b>         | no parity  |
| <b>USART_PM_ODD</b>          | odd parity   |
| <b>USART_PM EVEN</b>         | even parity  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure USART0 parity */
usart_parity_config(USART0, USART_PM_EVEN);
```

### usart\_word\_length\_set

The description of usart\_word\_length\_set is shown as below:

**Table 3-855. 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/UART word length                                  |
| <b>Precondition</b>          | -   |

|                              |                   |
|------------------------------|-------------------|
| <b>The called functions</b>  | -                 |
| <b>Input parameter{in}</b>   |                   |
| <b>usart_periph</b>          | uart peripheral   |
| <b>USARTx</b>                | x=0,1,2,5         |
| <b>UARTx</b>                 | x=3,4,6,7         |
| <b>Input parameter{in}</b>   |                   |
| <b>wlen</b>                  | USART word length |
| <b>USART_WL_8BIT</b>         | 8 bits            |
| <b>USART_WL_9BIT</b>         | 9 bits            |
| <b>Output parameter{out}</b> |                   |
| -                            | -                 |
| <b>Return value</b>          |                   |
| -                            | -                 |

Example:

```
/* configure USART0 word length */

uart_word_length_set(USART0, USART_WL_9BIT);
```

### **uart\_stop\_bit\_set**

The description of `uart_stop_bit_set` is shown as below:

**Table 3-856. Function `uart_stop_bit_set`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | uart_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/UART stop bit length                             |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | uart peripheral  |
| <b>USARTx</b>                | x=0,1,2,5  |
| <b>UARTx</b>                 | x=3,4,6,7  |
| <b>Input parameter{in}</b>   |  |
| <b>stblen</b>                | USART stop bit   |
| <b>USART_STB_1BIT</b>        | 1 bit  |
| <b>USART_STB_0_5BIT</b>      | 0.5 bit, not available for UARTx(x=3,4,6,7)                      |
| <b>USART_STB_2BIT</b>        | 2 bits   |
| <b>USART_STB_1_5BIT</b>      | 1.5 bits, not available for UARTx(x=3,4,6,7)                     |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```

/* configure USART0 stop bit length */

uart_stop_bit_set(USART0, USART_STB_1_5BIT);

```

### **uart\_enable**

The description of `uart_enable` is shown as below:

**Table 3-857. Function `uart_enable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | uart_enable                               |
| <b>Function prototype</b>    | void usart_enable(uint32_t usart_periph); |
| <b>Function descriptions</b> | enable USART/UART                         |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| usart_periph                 | uart peripheral                           |
| USARTx                       | x=0,1,2,5                                 |
| UARTx                        | x=3,4,6,7                                 |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```

/* enable USART0 */

usart_enable(USART0);

```

### **uart\_disable**

The description of `uart_disable` is shown as below:

**Table 3-858. Function `uart_disable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | uart_disable                               |
| <b>Function prototype</b>    | void usart_disable(uint32_t usart_periph); |
| <b>Function descriptions</b> | disable USART/UART                         |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| usart_periph                 | uart peripheral                            |
| USARTx                       | x=0,1,2,5                                  |
| UARTx                        | x=3,4,6,7                                  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable USART0 */

uart_disable(USART0);
```

### **uart\_transmit\_config**

The description of `uart_transmit_config` is shown as below:

**Table 3-859. Function `uart_transmit_config`**

|                               |   |
|-------------------------------|---|
| <b>Function name</b>          | uart_transmit_config  |
| <b>Function prototype</b>     | void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig); |
| <b>Function descriptions</b>  | configure USART/UART transmitter                                      |
| <b>Precondition</b>           | -   |
| <b>The called functions</b>   | -   |
| <b>Input parameter{in}</b>    |   |
| <b>usart_periph</b>           | uart peripheral   |
| <b>USARTx</b>                 | x=0,1,2,5   |
| <b>UARTx</b>                  | x=3,4,6,7   |
| <b>Input parameter{in}</b>    |   |
| <b>txconfig</b>               | enable or disable USART/UART transmitter                              |
| <b>USART_TRANSMIT_ENABLE</b>  | enable USART/UART transmission  |
| <b>USART_TRANSMIT_DISABLE</b> | enable USART/UART transmission  |
| <b>Output parameter{out}</b>  |   |
| -                             | -   |
| <b>Return value</b>           |   |
| -                             | -   |

Example:

```
/* configure USART0 transmitter */

uart_transmit_config(USART0, USART_TRANSMIT_ENABLE);
```

### **uart\_receive\_config**

The description of `uart_receive_config` is shown as below:

**Table 3-860. Function `uart_receive_config`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | uart_receive_config  |
| <b>Function prototype</b>    | void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig); |
| <b>Function descriptions</b> | configure USART/UART receiver  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |

| Input parameter{in}          |                                       |
|------------------------------|---------------------------------------|
| <b>usart_periph</b>          | uart peripheral                       |
| <b>USARTx</b>                | x=0,1,2,5                             |
| <b>UARTx</b>                 | x=3,4,6,7                             |
| Input parameter{in}          |                                       |
| <b>rxconfig</b>              | enable or disable USART/UART receiver |
| <b>USART_RECEIVE_ENABLE</b>  | enable USART/UART reception           |
| <b>USART_RECEIVE_DISABLE</b> | disable USART/UART reception          |
| Output parameter{out}        |                                       |
| -                            | -                                     |
| Return value                 |                                       |
| -                            | -                                     |

Example:

```
/* configure USART0 receiver */
uart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

### uart\_data\_first\_config

The description of `uart_data_first_config` is shown as below:

**Table 3-861. Function `uart_data_first_config`**

| <b>Function name</b>         | uart_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>  | -   |
| Input parameter{in}          |   |
| <b>usart_periph</b>          | uart peripheral   |
| <b>USARTx</b>                | x=0,1,2,5   |
| Input parameter{in}          |   |
| <b>msbf</b>                  | LSB/MSB   |
| <b>USART_MSBF_LSB</b>        | LSB first   |
| <b>USART_MSBF_MSB</b>        | MSB first   |
| Output parameter{out}        |   |
| -                            | -   |
| Return value                 |   |
| -                            | -   |

Example:

```
/* data is transmitted/received with the LSB first */
```

---

```
uart_data_first_config(USART0, USART_MSBF_LSB);
```

### **uart\_invert\_config**

The description of `uart_invert_config` is shown as below:

**Table 3-862. Function `uart_invert_config`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | uart_invert_config  |
| <b>Function prototype</b>    | void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);  |
| <b>Function descriptions</b> | configure USART inversion   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | uart peripheral   |
| <b>USARTx</b>                | x=0,1,2,5   |
| <b>Input parameter{in}</b>   |   |
| <b>invertpara</b>            | inversion parameters, refer to <a href="#">Table 3-851. usart_invert_enum</a> . |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* configure USART0 inversion */
uart_invert_config(USART0, USART_DINV_ENABLE);
```

### **uart\_receiver\_timeout\_enable**

The description of `uart_receiver_timeout_enable` is shown as below:

**Table 3-863. Function `uart_receiver_timeout_enable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | uart_receiver_timeout_enable                               |
| <b>Function prototype</b>    | void usart_receiver_timeout_enable(uint32_t usart_periph); |
| <b>Function descriptions</b> | enable receiver timeout                                    |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | uart peripheral  |
| <b>USARTx</b>                | x=0,1,2,5  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable USART0 receiver timeout */

uart_receiver_timeout_enable(USART0);
```

### **uart\_receiver\_timeout\_disable**

The description of `uart_receiver_timeout_disable` is shown as below:

**Table 3-864. Function `uart_receiver_timeout_disable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | uart_receiver_timeout_disable   |
| <b>Function prototype</b>    | void <code>uart_receiver_timeout_disable(uint32_t usart_periph);</code> |
| <b>Function descriptions</b> | disable receiver timeout  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <code>usart_periph</code>    | uart peripheral   |
| <code>USARTx</code>          | x=0,1,2,5   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable USART0 receiver timeout */

uart_receiver_timeout_disable(USART0);
```

### **uart\_receiver\_timeout\_threshold\_config**

The description of `uart_receiver_timeout_threshold_config` is shown as below:

**Table 3-865. Function `uart_receiver_timeout_threshold_config`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | uart_receiver_timeout_threshold_config  |
| <b>Function prototype</b>    | void <code>uart_receiver_timeout_threshold_config(uint32_t usart_periph,</code><br><code>                          uint32_t rtimeout);</code> |
| <b>Function descriptions</b> | configure the receiver timeout threshold  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <code>usart_periph</code>    | uart peripheral   |
| <code>USARTx</code>          | x=0,1,2,5   |
| <b>Input parameter{in}</b>   |   |
| <code>rtimeout</code>        | 0-0xFFFFFFF   |
| <b>Output parameter{out}</b> |   |

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

Example:

```
/* set the receiver timeout threshold of USART0 */

uart_receiver_timeout_threshold_config(USART0, 0xFFFF);
```

### **uart\_data\_transmit**

The description of `uart_data_transmit` is shown as below:

**Table 3-866. Function `uart_data_transmit`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | uart_data_transmit  |
| <b>Function prototype</b>    | void usart_data_transmit(uint32_t usart_periph, uint16_t data); |
| <b>Function descriptions</b> | USART/UART transmit data function                               |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | uart peripheral   |
| <b>USARTx</b>                | x=0,1,2,5   |
| <b>UARTx</b>                 | x=3,4,6,7   |
| <b>Input parameter{in}</b>   |   |
| <b>data</b>                  | data of transmission  |
| <b>0-0x1FF</b>               | data of transmission  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* USART0 transmit data */

uart_data_transmit(USART0, 0xAA);
```

### **uart\_data\_receive**

The description of `uart_data_receive` is shown as below:

**Table 3-867. Function `uart_data_receive`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | uart_data_receive                                   |
| <b>Function prototype</b>    | uint16_t usart_data_receive(uint32_t usart_periph); |
| <b>Function descriptions</b> | USART/UART receive data function                    |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |

| Input parameter{in}   |                           |
|-----------------------|---------------------------|
| uart_periph           | uart peripheral           |
| USARTx                | x=0,1,2,5                 |
| UARTx                 | x=3,4,6,7                 |
| Output parameter{out} |                           |
| -                     | -                         |
| Return value          |                           |
| uint16_t              | 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 3-868. Function usart\_address\_config**

| Function name         | usart_address_config  |
|-----------------------|---|
| Function prototype    | void usart_address_config(uint32_t usart_periph, uint8_t addr); |
| Function descriptions | configure address of the USART/UART                             |
| Precondition          | -   |
| The called functions  | -   |
| Input parameter{in}   |   |
| uart_periph           | uart peripheral   |
| USARTx                | x=0,1,2,5   |
| UARTx                 | x=3,4,6,7   |
| Input parameter{in}   |   |
| addr                  | address of USART/UART   |
| 0-0xFF                | address of USART/UART   |
| Output parameter{out} |   |
| -                     | -   |
| 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 3-869. Function usart\_mute\_mode\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | usart_mute_mode_enable                              |
| <b>Function prototype</b>    | void usart_mute_mode_enable(uint32_t usart_periph); |
| <b>Function descriptions</b> | enable mute mode                                    |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| usart_periph                 | uart peripheral                                     |
| USARTx                       | x=0,1,2,5   |
| UARTx                        | x=3,4,6,7   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable USART0 receiver mute mode */
usart_mute_mode_enable(USART0);
```

### **usart\_mute\_mode\_disable**

The description of usart\_mute\_mode\_disable is shown as below:

**Table 3-870. Function usart\_mute\_mode\_disable**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | usart_mute_mode_disable                              |
| <b>Function prototype</b>    | void usart_mute_mode_disable(uint32_t usart_periph); |
| <b>Function descriptions</b> | disable mute mode                                    |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| usart_periph                 | uart peripheral                                      |
| USARTx                       | x=0,1,2,5  |
| UARTx                        | x=3,4,6,7  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable USART0 receiver mute mode */
usart_mute_mode_disable(USART0);
```

### **uart\_mute\_mode\_wakeup\_config**

The description of `uart_mute_mode_wakeup_config` is shown as below:

**Table 3-871. Function `uart_mute_mode_wakeup_config`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | uart_mute_mode_wakeup_config   |
| <b>Function prototype</b>    | void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod); |
| <b>Function descriptions</b> | configure wakeup method in mute mode   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | uart peripheral  |
| <i>USARTx</i>                | x=0,1,2,5  |
| <i>UARTx</i>                 | x=3,4,6,7  |
| <b>Input parameter{in}</b>   |  |
| <b>wmethod</b>               | two methods be used to enter or exit the mute mode                           |
| <i>USART_WM_IDLE</i>         | idle line  |
| <i>USART_WM_ADDR</i>         | address mask   |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure USART0 wakeup method in mute mode */
uart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

### **uart\_lin\_mode\_enable**

The description of `uart_lin_mode_enable` is shown as below:

**Table 3-872. Function `uart_lin_mode_enable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | uart_lin_mode_enable                               |
| <b>Function prototype</b>    | void usart_lin_mode_enable(uint32_t usart_periph); |
| <b>Function descriptions</b> | enable LIN mode                                    |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | uart peripheral                                    |
| <i>USARTx</i>                | x=0,1,2,5  |
| <i>UARTx</i>                 | x=3,4,6,7  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |

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

Example:

```
/* enable USART0 LIN mode */
uart_lin_mode_enable(USART0);
```

### uart\_lin\_mode\_disable

The description of `uart_lin_mode_disable` is shown as below:

**Table 3-873. Function `uart_lin_mode_disable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | uart_lin_mode_disable                               |
| <b>Function prototype</b>    | void usart_lin_mode_disable(uint32_t usart_periph); |
| <b>Function descriptions</b> | disable LIN mode                                    |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | uart peripheral                                     |
| <b>USARTx</b>                | x=0,1,2,5   |
| <b>UARTx</b>                 | x=3,4,6,7   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable USART0 LIN mode */
uart_lin_mode_disable(USART0);
```

### uart\_lin\_break\_decton\_length\_config

The description of `uart_lin_break_decton_length_config` is shown as below:

**Table 3-874. Function `uart_lin_break_decton_length_config`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | uart_lin_break_decton_length_config   |
| <b>Function prototype</b>    | void usart_lin_break_decton_length_config(uint32_t usart_periph, uint32_t lrlen); |
| <b>Function descriptions</b> | configure lin break frame length  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | uart peripheral   |
| <b>USARTx</b>                | x=0,1,2,5   |

|                              |                        |
|------------------------------|------------------------|
| <i>UARTx</i>                 | x=3,4,6,7              |
| <b>Input parameter{in}</b>   |                        |
| <b>Iblen</b>                 | lin break frame length |
| <i>USART_LBLEN_10B</i>       | 10 bits                |
| <i>USART_LBLEN_11B</i>       | 11 bits                |
| <b>Output parameter{out}</b> |                        |
| -                            | -                      |
| <b>Return value</b>          |                        |
| -                            | -                      |

Example:

```
/* configure LIN break frame length */

uart_lin_break_decton_length_config(USART0, USART_LBLEN_10B);
```

### **uart\_send\_break**

The description of **uart\_send\_break** is shown as below:

**Table 3-875. Function `uart_send_break`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | uart_send_break                               |
| <b>Function prototype</b>    | void usart_send_break(uint32_t usart_periph); |
| <b>Function descriptions</b> | send break frame                              |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | usart peripheral                              |
| <i>USARTx</i>                | x=0,1,2,5                                     |
| <i>UARTx</i>                 | x=3,4,6,7                                     |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* USART0 send break frame */

uart_send_break(USART0);
```

### **uart\_halfduplex\_enable**

The description of **uart\_halfduplex\_enable** is shown as below:

**Table 3-876. Function `uart_halfduplex_enable`**

|                           |  |
|---------------------------|--|
| <b>Function name</b>      | uart_halfduplex_enable                               |
| <b>Function prototype</b> | void usart_halfduplex_enable(uint32_t usart_periph); |

|                              |                         |
|------------------------------|-------------------------|
| <b>Function descriptions</b> | enable half duplex mode |
| <b>Precondition</b>          | -                       |
| <b>The called functions</b>  | -                       |
| <b>Input parameter{in}</b>   |                         |
| <b>usart_periph</b>          | usart peripheral        |
| <b>USARTx</b>                | x=0,1,2,5               |
| <b>UARTx</b>                 | x=3,4,6,7               |
| <b>Output parameter{out}</b> |                         |
| -                            | -                       |
| <b>Return value</b>          |                         |
| -                            | -                       |

Example:

```
/* enable USART0 half duplex mode*/
uart_halfduplex_enable(USART0);
```

### **uart\_halfduplex\_disable**

The description of `uart_halfduplex_disable` is shown as below:

**Table 3-877. Function `uart_halfduplex_disable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | uart_halfduplex_disable                               |
| <b>Function prototype</b>    | void usart_halfduplex_disable(uint32_t usart_periph); |
| <b>Function descriptions</b> | disable half duplex mode                              |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | usart peripheral                                      |
| <b>USARTx</b>                | x=0,1,2,5   |
| <b>UARTx</b>                 | x=3,4,6,7   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable USART0 half duplex mode*/
uart_halfduplex_disable(USART0);
```

### **uart\_synchronous\_clock\_enable**

The description of `uart_synchronous_clock_enable` is shown as below:

**Table 3-878. 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>   |   |
| usart_periph                 | uart peripheral   |
| USARTx                       | x=0,1,2,5   |
| <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 3-879. 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>   |  |
| usart_periph                 | uart peripheral  |
| USARTx                       | x=0,1,2,5  |
| <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 3-880. 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>          | uart peripheral  |
| <b>USARTx</b>                | x=0,1,2,5  |
| <b>Input parameter{in}</b>   |  |
| <b>clen</b>                  | CK length  |
| <b>USART_CLEN_NONE</b>       | there are 7 CK pulses for an 8 bit frame and 8 CK pulses for a 9 bit frame                             |
| <b>USART_CLEN_EN</b>         | 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  |
| <b>USART_CPH_1CK</b>         | first clock transition is the first data capture edge  |
| <b>USART_CPH_2CK</b>         | second clock transition is the first data capture edge   |
| <b>Input parameter{in}</b>   |  |
| <b>cpl</b>                   | clock polarity   |
| <b>USART_CPL_LOW</b>         | steady low value on CK pin   |
| <b>USART_CPL_HIGH</b>        | 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 3-881. Function usart\_guard\_time\_config**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | usart_guard_time_config  |
| <b>Function prototype</b>    | void usart_guard_time_config(uint32_t usart_periph,uint32_t guat); |
| <b>Function descriptions</b> | configure guard time value in smartcard mode                       |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |

|                              |                  |
|------------------------------|------------------|
| <b>uart_periph</b>           | uart peripheral  |
| <b>USARTx</b>                | x=0,1,2,5        |
| <b>Input parameter{in}</b>   |                  |
| <b>guat</b>                  | guard time value |
| <b>0-0x000000FF</b>          | guard time value |
| <b>Output parameter{out}</b> |                  |
| -                            | -                |
| <b>Return value</b>          |                  |
| -                            | -                |

Example:

```
/* configure USART0 guard time value in smartcard mode */

uart_guard_time_config(USART0, 0x00000055);
```

### **uart\_smartcard\_mode\_enable**

The description of **uart\_smartcard\_mode\_enable** is shown as below:

**Table 3-882. Function `uart_smartcard_mode_enable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | uart_smartcard_mode_enable                              |
| <b>Function prototype</b>    | void uart_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>uart_periph</b>           | uart peripheral   |
| <b>USARTx</b>                | x=0,1,2,5   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable USART0 smartcard mode */

uart_smartcard_mode_enable(USART0);
```

### **uart\_smartcard\_mode\_disable**

The description of **uart\_smartcard\_mode\_disable** is shown as below:

**Table 3-883. Function `uart_smartcard_mode_disable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | uart_smartcard_mode_disable                              |
| <b>Function prototype</b>    | void uart_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>          | uart peripheral |
| <b>USARTx</b>                | x=0,1,2,5       |
| <b>Output parameter{out}</b> |                 |
| -                            | -               |
| <b>Return value</b>          |                 |
| -                            | -               |

Example:

```
/* disable USART0 smartcard mode */

uart_smartcard_mode_disable(USART0);
```

### **uart\_smartcard\_mode\_nack\_enable**

The description of **uart\_smartcard\_mode\_nack\_enable** is shown as below:

**Table 3-884. Function `uart_smartcard_mode_nack_enable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | uart_smartcard_mode_nack_enable                              |
| <b>Function prototype</b>    | void uart_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>          | uart peripheral  |
| <b>USARTx</b>                | x=0,1,2,5  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* enable USART0 NACK in smartcard mode */

uart_smartcard_mode_nack_enable(USART0);
```

### **uart\_smartcard\_mode\_nack\_disable**

The description of **uart\_smartcard\_mode\_nack\_disable** is shown as below:

**Table 3-885. Function `uart_smartcard_mode_nack_disable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | uart_smartcard_mode_nack_disable                              |
| <b>Function prototype</b>    | void uart_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>          | uart peripheral |
| <b>USARTx</b>                | x=0,1,2,5       |
| <b>Output parameter{out}</b> |                 |
| -                            | -               |
| <b>Return value</b>          |                 |
| -                            | -               |

Example:

```
/* disable USART0 NACK in smartcard mode */

uart_smartcard_mode_nack_disable(USART0);
```

### **uart\_smartcard\_autoretry\_config**

The description of **uart\_smartcard\_autoretry\_config** is shown as below:

**Table 3-886. Function `uart_smartcard_autoretry_config`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | uart_smartcard_autoretry_config  |
| <b>Function prototype</b>    | void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrnum); |
| <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>          | uart peripheral  |
| <b>USARTx</b>                | x=0,1,2,5  |
| <b>Input parameter{in}</b>   |  |
| <b>scrnum</b>                | smartcard auto-retry number  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* configure USART0 smartcard auto-retry number */

uart_smartcard_autoretry_config(USART0, 0x00000005);
```

### **uart\_block\_length\_config**

The description of **uart\_block\_length\_config** is shown as below:

**Table 3-887. Function usart\_block\_length\_config**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | usart_block_length_config   |
| <b>Function prototype</b>    | void usart_block_length_config(uint32_t usart_periph, uint32_t bl); |
| <b>Function descriptions</b> | configure block length  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| usart_periph                 | uart peripheral   |
| USARTx                       | x=0,1,2,5   |
| <b>Input parameter{in}</b>   |   |
| bl                           | block length  |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* configure USART0 block length in Smartcard T=1 reception */
usart_block_length_config(USART0, 0x00000005);
```

### **usart\_irda\_mode\_enable**

The description of usart\_irda\_mode\_enable is shown as below:

**Table 3-888. Function usart\_irda\_mode\_enable**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | usart_irda_mode_enable                              |
| <b>Function prototype</b>    | void usart_irda_mode_enable(uint32_t usart_periph); |
| <b>Function descriptions</b> | enable IrDA mode                                    |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| usart_periph                 | uart peripheral                                     |
| USARTx                       | x=0,1,2,5   |
| UARTx                        | x=3,4,6,7   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable USART0 IrDA mode */
usart_irda_mode_enable(USART0);
```

### **uart\_irda\_mode\_disable**

The description of `uart_irda_mode_disable` is shown as below:

**Table 3-889. Function `uart_irda_mode_disable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | uart_irda_mode_disable   |
| <b>Function prototype</b>    | void <code>uart_irda_mode_disable(uint32_t usart_periph);</code> |
| <b>Function descriptions</b> | disable IrDA mode  |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <code>usart_periph</code>    | uart peripheral  |
| <code>USARTx</code>          | x=0,1,2,5  |
| <code>UARTx</code>           | x=3,4,6,7  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* disable USART0 IrDA mode */
uart_irda_mode_disable(USART0);
```

### **uart\_prescaler\_config**

The description of `uart_prescaler_config` is shown as below:

**Table 3-890. Function `uart_prescaler_config`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | uart_prescaler_config  |
| <b>Function prototype</b>    | void <code>uart_prescaler_config(uint32_t usart_periph, uint8_t psc);</code> |
| <b>Function descriptions</b> | configure the peripheral clock prescaler in USART/UART IrDA low-power mode   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <code>usart_periph</code>    | uart peripheral  |
| <code>USARTx</code>          | x=0,1,2,5  |
| <code>UARTx</code>           | x=3,4,6,7  |
| <b>Input parameter{in}</b>   |  |
| <code>psc</code>             | clock prescaler  |
| <code>0x00-0xFF</code>       | clock prescaler  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |

|   |   |
|---|---|
| - | - |
|---|---|

Example:

```
/* configure the USART0 peripheral clock prescaler in USART IrDA low-power mode */
uart_prescaler_config(USART0, 0x00);
```

### uart\_irda\_lowpower\_config

The description of `uart_irda_lowpower_config` is shown as below:

**Table 3-891. Function `uart_irda_lowpower_config`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | uart_irda_lowpower_config  |
| <b>Function prototype</b>    | void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp); |
| <b>Function descriptions</b> | configure IrDA low-power   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | uart peripheral  |
| <i>USARTx</i>                | x=0,1,2,5  |
| <i>UARTx</i>                 | x=3,4,6,7  |
| <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 */
uart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

### uart\_hardware\_flow\_rts\_config

The description of `uart_hardware_flow_rts_config` is shown as below:

**Table 3-892. Function `uart_hardware_flow_rts_config`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | uart_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>uart_periph</b>           | uart peripheral       |
| <b>USARTx</b>                | x=0,1,2,5             |
| <b>Input parameter{in}</b>   |                       |
| <b>rtsconfig</b>             | enable or disable RTS |
| <b>USART_RTS_ENABLE</b>      | enable RTS            |
| <b>USART_RTS_DISABLE</b>     | disable RTS           |
| <b>Output parameter{out}</b> |                       |
| -                            | -                     |
| <b>Return value</b>          |                       |
| -                            | -                     |

Example:

```
/* configure USART0 hardware flow control RTS */
uart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);
```

### **uart\_hardware\_flow\_cts\_config**

The description of `uart_hardware_flow_cts_config` is shown as below:

**Table 3-893. Function `uart_hardware_flow_cts_config`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | uart_hardware_flow_cts_config   |
| <b>Function prototype</b>    | void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig); |
| <b>Function descriptions</b> | configure hardware flow control CTS   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>uart_periph</b>           | uart peripheral   |
| <b>USARTx</b>                | x=0,1,2,5   |
| <b>Input parameter{in}</b>   |   |
| <b>ctsconfig</b>             | enable or disable CTS   |
| <b>USART_CTS_ENABLE</b>      | enable CTS  |
| <b>USART_CTS_DISABLE</b>     | disable CTS   |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```

/* configure USART0 hardware flow control CTS */

uart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);

```

### **uart\_dma\_enable**

The description of `uart_dma_enable` is shown as below:

**Table 3-894. Function `uart_dma_enable`**

|                                 |   |
|---------------------------------|---|
| <b>Function name</b>            | uart_dma_enable   |
| <b>Function prototype</b>       | void <code>uart_dma_enable(uint32_t usart_periph, uint8_t dmacmd);</code> |
| <b>Function descriptions</b>    | enable USART/UART DMA send or receive                                     |
| <b>Precondition</b>             | -   |
| <b>The called functions</b>     | -   |
| <b>Input parameter{in}</b>      |   |
| <code>usart_periph</code>       | uart peripheral   |
| <code>USARTx</code>             | x=0,1,2,5   |
| <code>UARTx</code>              | x=3,4,6,7   |
| <b>Input parameter{in}</b>      |   |
| <code>dmacmd</code>             | DMA mode  |
| <code>USART_DMA_RECEIVE</code>  | receive data use DMA  |
| <code>USART_DMA_TRANSMIT</code> | transmit data use DMA   |
| <b>Output parameter{out}</b>    |   |
| -                               | -   |
| <b>Return value</b>             |   |
| -                               | -   |

Example:

```

/* enable USART0 DMA for reception */

uart_dma_enable(USART0, USART_DMA_RECEIVE);

```

### **uart\_dma\_disable**

The description of `uart_dma_disable` is shown as below:

**Table 3-895. Function `uart_dma_disable`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | uart_dma_disable   |
| <b>Function prototype</b>    | void <code>uart_dma_disable(uint32_t usart_periph, uint8_t dmacmd);</code> |
| <b>Function descriptions</b> | disable USART/UART DMA send or receive                                     |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <code>usart_periph</code>    | uart peripheral  |

|                              |                       |
|------------------------------|-----------------------|
| <i>USARTx</i>                | x=0,1,2,5             |
| <i>UARTx</i>                 | x=3,4,6,7             |
| <b>Input parameter{in}</b>   |                       |
| <b>dmacmd</b>                | DMA mode              |
| <i>USART_DMA_RECEIVE</i>     | receive data use DMA  |
| <i>USART_DMA_TRANS</i>       | transmit data use DMA |
| <b>Output parameter{out}</b> |                       |
| -                            | -                     |
| <b>Return value</b>          |                       |
| -                            | -                     |

Example:

```
/* disable USART0 DMA for reception */
uart_dma_disable(USART0, USART_DMA_RECEIVE);
```

### uart\_flag\_get

The description of `uart_flag_get` is shown as below:

**Table 3-896. Function `uart_flag_get`**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | <code>uart_flag_get</code>   |
| <b>Function prototype</b>    | <code>FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);</code> |
| <b>Function descriptions</b> | get flag in STAT0/STAT1 register   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | uart peripheral  |
| <i>USARTx</i>                | x=0,1,2,5  |
| <i>UARTx</i>                 | x=3,4,6,7  |
| <b>Input parameter{in}</b>   |  |
| <b>flag</b>                  | USART flags, refer to <a href="#">Table 3-848. usart_flag_enum</a> .                 |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| <b>FlagStatus</b>            | SET or RESET   |

Example:

```
/* get flag USART0 state */

FlagStatus status;

status = usart_flag_get(USART0, USART_FLAG_TBE);
```

### uart\_flag\_clear

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

**Table 3-897. 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>          | uart peripheral  |
| <i>USARTx</i>                | x=0,1,2,5  |
| <i>UARTx</i>                 | x=3,4,6,7  |
| <b>Input parameter{in}</b>   |  |
| <b>flag</b>                  | USART flags, refer to <a href="#">Table 3-848. usart_flag_enum</a> . |
| <i>USART_FLAG_CTS</i>        | CTS change flag  |
| <i>USART_FLAG_LBD</i>        | LIN break detected flag  |
| <i>USART_FLAG_TC</i>         | transmission complete  |
| <i>USART_FLAG_RBNE</i>       | read data buffer not empty   |
| <i>USART_FLAG_EB</i>         | end of block flag  |
| <i>USART_FLAG_RT</i>         | receiver timeout flag  |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* clear USART0 flag */
usart_flag_clear(USART0, USART_FLAG_TC);
```

### uart\_interrupt\_enable

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

**Table 3-898. 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/UART interrupt   |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |

|                              |   |
|------------------------------|---|
| <b>usart_periph</b>          | uart peripheral   |
| <i>USARTx</i>                | x=0,1,2,5   |
| <i>UARTx</i>                 | x=3,4,6,7   |
| <b>Input parameter{in}</b>   |   |
| <b>interrupt</b>             | USART interrupt, refer to <a href="#">Table 3-850. usart_interrupt_enum</a> . |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* enable USART0 TBE interrupt */

uart_interrupt_enable(USART0, USART_INT_TBE);
```

### **uart\_interrupt\_disable**

The description of **uart\_interrupt\_disable** is shown as below:

**Table 3-899. Function `uart_interrupt_disable`**

|                              |   |
|------------------------------|---|
| <b>Function name</b>         | uart_interrupt_disable  |
| <b>Function prototype</b>    | void uart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt); |
| <b>Function descriptions</b> | disable USART/UART interrupt  |
| <b>Precondition</b>          | -   |
| <b>The called functions</b>  | -   |
| <b>Input parameter{in}</b>   |   |
| <b>usart_periph</b>          | uart peripheral   |
| <i>USARTx</i>                | x=0,1,2,5   |
| <i>UARTx</i>                 | x=3,4,6,7   |
| <b>Input parameter{in}</b>   |   |
| <b>interrupt</b>             | USART interrupt, refer to <a href="#">Table 3-850. usart_interrupt_enum</a> .       |
| <b>Output parameter{out}</b> |   |
| -                            | -   |
| <b>Return value</b>          |   |
| -                            | -   |

Example:

```
/* disable USART0 TBE interrupt */

uart_interrupt_disable(USART0, USART_INT_TBE);
```

### **uart\_interrupt\_flag\_get**

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

**Table 3-900. 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/UART interrupt and flag status   |
| <b>Precondition</b>          | -  |
| <b>The called functions</b>  | -  |
| <b>Input parameter{in}</b>   |  |
| <b>usart_periph</b>          | usart peripheral   |
| <b>USARTx</b>                | x=0,1,2,5  |
| <b>UARTx</b>                 | x=3,4,6,7  |
| <b>Input parameter{in}</b>   |  |
| <b>int_flag</b>              | USART interrupt flag, refer to <a href="#">Table 3-849. usart_interrupt_flag_enum</a> .            |
| <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 3-901. Function usart\_interrupt\_flag\_clear**

|                                |  |
|--------------------------------|--|
| <b>Function name</b>           | usart_interrupt_flag_clear   |
| <b>Function prototype</b>      | void usart_interrupt_flag_clear(uint32_t usart_periph,<br>usart_interrupt_flag_enum int_flag); |
| <b>Function descriptions</b>   | clear USART/UART interrupt flag in STAT0/STAT1 register  |
| <b>Precondition</b>            | -  |
| <b>The called functions</b>    | -  |
| <b>Input parameter{in}</b>     |  |
| <b>usart_periph</b>            | usart peripheral   |
| <b>USARTx</b>                  | x=0,1,2,5  |
| <b>UARTx</b>                   | x=3,4,6,7  |
| <b>Input parameter{in}</b>     |  |
| <b>int_flag</b>                | USART interrupt flag, refer to <a href="#">Table 3-849. usart_interrupt_flag_enum</a> .        |
| <b>USART_INT_FLAG_CT<br/>S</b> | CTS interrupt and flag   |
| <b>USART_INT_FLAG_LB</b>       | LIN break detected interrupt and flag  |

|                              |  |
|------------------------------|--|
| <i>D</i>                     |  |
| <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_EB</i>     | interrupt enable bit of end of block event and flag    |
| <i>USART_INT_FLAG_RT</i>     | interrupt enable bit of receive timeout event and flag |
| <b>Output parameter{out}</b> |  |
| -                            | -  |
| <b>Return value</b>          |  |
| -                            | -  |

Example:

```
/* clear the USART0 interrupt flag status */
uart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

## 3.29. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.29.1](#), the WWDGT firmware functions are introduced in chapter [3.29.2](#).

### 3.29.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

**Table 3-902. WWDGT Registers**

| Registers  | Descriptions           |
|------------|------------------------|
| WWDGT_CTL  | control register       |
| WWDGT_CFG  | configuration register |
| WWDGT_STAT | status register        |

### 3.29.2. Descriptions of Peripheral functions

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

**Table 3-903. WWDGT firmware function**

| Function name        | Function description   |
|----------------------|--|
| wwdgt_deinit         | reset the window watchdog timer configuration                      |
| wwdgt_enable         | start the window watchdog timer counter                            |
| wwdgt_counter_update | configure the window watchdog timer counter value                  |
| wwdgt_config         | configure counter value, window value, and prescaler divider value |
| wwdgt_flag_get       | check early wakeup interrupt state of WWDGT                        |

| Function name            | Function description                        |
|--------------------------|---|
| wwdgt_flag_clear         | clear early wakeup interrupt state of WWDGT |
| wwdgt_interrupt_enable   | enable early wakeup interrupt of WWDGT      |
| wwdgt_interrupt_flag_get | get early wakeup interrupt flag of WWDGT    |

### wwdgt\_deinit

The description of wwdgt\_deinit is shown as below:

**Table 3-904. 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  | rcu_periph_reset_enable / rcu_periph_reset_disable |
| 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 3-905. 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 3-906. 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 3-907. 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   |
| <b>WWDGT_CFG_PSC_D<br/>IV1</b> | the time base of window watchdog counter = (PCLK1/4096)/1                 |
| <b>WWDGT_CFG_PSC_D<br/>IV2</b> | the time base of window watchdog counter = (PCLK1/4096)/2                 |

|                              |   |
|------------------------------|---|
| <i>WWDGT_CFG_PSC_DIV4</i>    | the time base of window watchdog counter = (PCLK1/4096)/4 |
| <i>WWDGT_CFG_PSC_DIV8</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\_flag\_get**

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

**Table 3-908. 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 reaches 0x40 or refreshes before it reaches the window value */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get ( );
```

### **wwdgt\_flag\_clear**

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

**Table 3-909. 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();
```

### **wwdgt\_interrupt\_enable**

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

**Table 3-910. 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\_interrupt\_flag\_get**

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

**Table 3-911. Function wwdgt\_interrupt\_flag\_get**

|                              |  |
|------------------------------|--|
| <b>Function name</b>         | wwdgt_interrupt_flag_get                   |
| <b>Function prototype</b>    | FlagStatus wwdgt_interrupt_flag_get(void); |
| <b>Function descriptions</b> | get early wakeup interrupt flag 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 reaches 0x40 or refreshes before it reaches the window value */
```

```
FlagStatus status;
```

```
status = wwdgt_interrupt_flag_get ( );
```

## 4. Revision history

**Table 4-1. Revision history**

| Revison No. | Description   | Date           |
|-------------|---|----------------|
| 1.0         | Initial Release   | Oct. 31th,2018 |
| 1.1         | Revision contents: gd32f20x has no usbd, the usbd chapter is deleted from the catalog; the folder description in section 2.1.2 is modified; the folder structure diagram in section 2.1.1 is replaced; the folder description in section 2.1.4 is modified; the header format of some drawings is modified  | Sep.30th,2020  |
| 1.2         | Rebase Release  | Oct. 31th,2021 |
| 1.3         | 1. Add function<br><code>exmc_sdram_struct_command_para_init()</code><br>2. Change function<br><code>void pmu_to_standbymode(uint8_t standbymodecmd)</code><br>3. Change all function <code>qspi_xxx()</code> to <code>spi_quad_xxx()</code><br>4. Add function<br><code>fwdgt_prescaler_value_config()</code><br><code>fwdgt_reload_value_config()</code><br>5. Delete USBFS chapter | Jul. 31th,2022 |

## 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.