

**GigaDevice Semiconductor Inc.**

**GD32A50x**

**Arm<sup>®</sup> Cortex<sup>®</sup>-M33 32-bit MCU**

**固件库  
使用指南**

1.5 版本

(2025 年 2 月)

# 目录

|                           |            |
|---------------------------|------------|
| 目录.....                   | 1          |
| 图索引 .....                 | 4          |
| 表索引 .....                 | 5          |
| <b>1. 介绍.....</b>         | <b>25</b>  |
| <b>1.1. 文档和固件库规则.....</b> | <b>25</b>  |
| 1.1.1. 外设缩写.....          | 25         |
| 1.1.2. 命名规则.....          | 26         |
| <b>2. 固件库概述.....</b>      | <b>27</b>  |
| <b>2.1. 文件组织结构.....</b>   | <b>27</b>  |
| 2.1.1. Examples 文件夹 ..... | 28         |
| 2.1.2. Firmware 文件夹 ..... | 28         |
| 2.1.3. Template 文件夹.....  | 28         |
| 2.1.4. Utilities 文件夹..... | 31         |
| <b>2.2. 固件库文件描述 .....</b> | <b>31</b>  |
| <b>3. 外设固件库.....</b>      | <b>32</b>  |
| <b>3.1. 外设固件库概述 .....</b> | <b>32</b>  |
| <b>3.2. ADC .....</b>     | <b>32</b>  |
| 3.2.1. 外设寄存器描述.....       | 32         |
| 3.2.2. 外设库函数说明 .....      | 33         |
| <b>3.3. BKP.....</b>      | <b>63</b>  |
| 3.3.1. 外设寄存器说明.....       | 63         |
| 3.3.2. 外设库函数说明 .....      | 63         |
| <b>3.4. CAN .....</b>     | <b>75</b>  |
| 3.4.1. 外设寄存器说明 .....      | 75         |
| 3.4.2. 外设库函数说明 .....      | 77         |
| <b>3.5. CMP .....</b>     | <b>121</b> |
| 3.5.1. 外设寄存器说明 .....      | 121        |
| 3.5.2. 外设库函数说明 .....      | 121        |
| <b>3.6. CRC .....</b>     | <b>131</b> |
| 3.6.1. 外设寄存器说明 .....      | 131        |
| 3.6.2. 外设库函数说明 .....      | 131        |
| <b>3.7. DBG .....</b>     | <b>139</b> |
| 3.7.1. 外设寄存器说明 .....      | 139        |

|              |                         |            |
|--------------|-------------------------|------------|
| 3.7.2.       | 外设库函数说明 .....           | 139        |
| <b>3.8.</b>  | <b>DAC .....</b>        | <b>143</b> |
| 3.8.1.       | 外设寄存器说明 .....           | 144        |
| 3.8.2.       | 外设库函数说明 .....           | 144        |
| <b>3.9.</b>  | <b>DMA/DMAMUX .....</b> | <b>159</b> |
| 3.9.1.       | 外设寄存器说明 .....           | 159        |
| 3.9.2.       | 外设库函数说明 .....           | 160        |
| <b>3.10.</b> | <b>EXTI.....</b>        | <b>208</b> |
| 3.10.1.      | 外设寄存器说明 .....           | 208        |
| 3.10.2.      | 外设库函数说明 .....           | 209        |
| <b>3.11.</b> | <b>FMC.....</b>         | <b>216</b> |
| 3.11.1.      | 外设寄存器说明 .....           | 216        |
| 3.11.2.      | 外设库函数说明 .....           | 217        |
| <b>3.12.</b> | <b>FWDGT.....</b>       | <b>253</b> |
| 3.12.1.      | 外设寄存器说明 .....           | 253        |
| 3.12.2.      | 外设库函数说明 .....           | 253        |
| <b>3.13.</b> | <b>GPIO.....</b>        | <b>258</b> |
| 3.13.1.      | 外设寄存器说明 .....           | 258        |
| 3.13.2.      | 外设库函数说明 .....           | 259        |
| <b>3.14.</b> | <b>I2C .....</b>        | <b>269</b> |
| 3.14.1.      | 外设寄存器说明 .....           | 269        |
| 3.14.2.      | 外设库函数说明 .....           | 270        |
| <b>3.15.</b> | <b>MFCOM .....</b>      | <b>308</b> |
| 3.15.1.      | 外设寄存器说明 .....           | 308        |
| 3.15.2.      | 外设库函数说明 .....           | 308        |
| <b>3.16.</b> | <b>MISC.....</b>        | <b>333</b> |
| 3.16.1.      | 外设寄存器说明 .....           | 333        |
| 3.16.2.      | 外设库函数说明 .....           | 333        |
| <b>3.17.</b> | <b>PMU.....</b>         | <b>340</b> |
| 3.17.1.      | 外设寄存器说明 .....           | 340        |
| 3.17.2.      | 外设库函数说明 .....           | 340        |
| <b>3.18.</b> | <b>RCU .....</b>        | <b>352</b> |
| 3.18.1.      | 外设寄存器说明 .....           | 352        |
| 3.18.2.      | 外设库函数说明 .....           | 353        |
| <b>3.19.</b> | <b>RTC .....</b>        | <b>388</b> |
| 3.19.1.      | 外设寄存器描述 .....           | 388        |
| 3.19.2.      | 外设库函数描述 .....           | 389        |
| <b>3.20.</b> | <b>SPI.....</b>         | <b>398</b> |
| 3.20.1.      | 外设寄存器说明 .....           | 398        |

|              |                     |            |
|--------------|---------------------|------------|
| 3.20.2.      | 外设库函数说明 .....       | 398        |
| <b>3.21.</b> | <b>SYSCFG .....</b> | <b>425</b> |
| 3.21.1.      | 外设寄存器说明 .....       | 425        |
| 3.21.2.      | 外设库函数说明 .....       | 425        |
| <b>3.22.</b> | <b>TIMER.....</b>   | <b>437</b> |
| 3.22.1.      | 外设寄存器说明 .....       | 437        |
| 3.22.2.      | 外设库函数说明 .....       | 438        |
| <b>3.23.</b> | <b>TRGSEL .....</b> | <b>517</b> |
| 3.23.1.      | 外设寄存器说明 .....       | 517        |
| 3.23.2.      | 外设库函数说明 .....       | 517        |
| <b>3.24.</b> | <b>USART.....</b>   | <b>524</b> |
| 3.24.1.      | 外设寄存器说明 .....       | 524        |
| 3.24.2.      | 外设库函数说明 .....       | 524        |
| <b>3.25.</b> | <b>WWDGT.....</b>   | <b>570</b> |
| 3.25.1.      | 外设寄存器说明 .....       | 570        |
| 3.25.2.      | 外设库函数说明 .....       | 570        |
| <b>4.</b>    | <b>版本历史 .....</b>   | <b>576</b> |

## 图索引

|                                 |    |
|---------------------------------|----|
| 图 2-1. GD32A50x 固件库文件组织结构 ..... | 27 |
| 图 2-2. 选择外设例程文件 .....           | 29 |
| 图 2-3. 拷贝外设例程文件 .....           | 29 |
| 图 2-4. 打开工程文件 .....             | 30 |
| 图 2-5. 配置工程文件 .....             | 30 |
| 图 2-6. 编译调试下载 .....             | 31 |

## 表索引

|  |    |
|--|----|
| 表 1-1. 外设缩写.....                                     | 25 |
| 表 2-1. 固件函数库文件描述.....                                | 31 |
| 表 3-1. 外设固件库函数描述格式 .....                             | 32 |
| 表 3-2. ADC 寄存器.....                                  | 32 |
| 表 3-3. ADC 库函数.....                                  | 33 |
| 表 3-4. 函数 adc_deinit .....                           | 34 |
| 表 3-5. 函数 adc_enable .....                           | 35 |
| 表 3-6. 函数 adc_disable .....                          | 35 |
| 表 3-7. 函数 adc_calibration_enable .....               | 36 |
| 表 3-8. 函数 adc_dma_mode_enable .....                  | 36 |
| 表 3-9. 函数 adc_dma_mode_disable .....                 | 37 |
| 表 3-10. 函数 adc_tempsensor_enable .....               | 37 |
| 表 3-11. 函数 adc_tempsensor_disable .....              | 38 |
| 表 3-12. 函数 adc_vrefint_enable .....                  | 38 |
| 表 3-13. 函数 adc_vrefint_disable .....                 | 39 |
| 表 3-14. 函数 adc_discontinuous_mode_config.....        | 39 |
| 表 3-15. 函数 adc_mode_config .....                     | 40 |
| 表 3-16. 函数 adc_special_function_config .....         | 41 |
| 表 3-17. 函数 adc_data_alignment_config .....           | 42 |
| 表 3-18. 函数 adc_channel_length_config .....           | 43 |
| 表 3-19. 函数 adc_regular_channel_config .....          | 43 |
| 表 3-20. 函数 adc_inserted_channel_config .....         | 44 |
| 表 3-21. 函数 adc_inserted_channel_offset_config .....  | 45 |
| 表 3-22. 函数 adc_external_trigger_config.....          | 46 |
| 表 3-23. 函数 adc_external_trigger_source_config.....   | 47 |
| 表 3-24. 函数 adc_software_trigger_enable .....         | 48 |
| 表 3-25. 函数 adc_regular_data_read .....               | 49 |
| 表 3-26. 函数 adc_inserted_data_read.....               | 49 |
| 表 3-27. 函数 adc_sync_mode_convert_value_read .....    | 50 |
| 表 3-28. 函数 adc_watchdog0_single_channel_enable ..... | 50 |
| 表 3-29. 函数 adc_watchdog0_group_channel_enable .....  | 51 |
| 表 3-30. 函数 adc_watchdog0_disable .....               | 52 |
| 表 3-31. 函数 adc_watchdog1_channel_config .....        | 52 |
| 表 3-32. 函数 adc_watchdog1_disable .....               | 53 |
| 表 3-33. 函数 adc_watchdog0_threshold_config .....      | 54 |
| 表 3-34. 函数 adc_watchdog1_threshold_config .....      | 54 |
| 表 3-35. 函数 adc_resolution_config .....               | 55 |
| 表 3-36. 函数 adc_oversample_mode_config .....          | 56 |
| 表 3-37. 函数 adc_oversample_mode_enable.....           | 58 |
| 表 3-38. 函数 adc_oversample_mode_disable.....          | 58 |

|  |    |
|--|----|
| 表 3-39. 函数 adc_flag_get .....                        | 59 |
| 表 3-40. 函数 adc_flag_clear .....                      | 59 |
| 表 3-41. 函数 adc_interrupt_enable .....                | 60 |
| 表 3-42. 函数 adc_interrupt_disable .....               | 61 |
| 表 3-43. 函数 adc_interrupt_flag_get .....              | 61 |
| 表 3-44. 函数 adc_interrupt_flag_clear .....            | 62 |
| 表 3-45. BKP 寄存器 .....                                | 63 |
| 表 3-46. BKP 库函数 .....                                | 63 |
| 表 3-47. 枚举类型 bkp_data_register_enum .....            | 64 |
| 表 3-48. 函数 bkp_deinit .....                          | 64 |
| 表 3-49. 函数 bkp_data_write .....                      | 65 |
| 表 3-50. 函数 bkp_data_read .....                       | 65 |
| 表 3-51. 函数 bkp_rtc_calibration_output_enable .....   | 66 |
| 表 3-52. 函数 bkp_rtc_calibration_output_disable .....  | 66 |
| 表 3-53. 函数 bkp_rtc_signal_output_enable .....        | 67 |
| 表 3-54. 函数 bkp_rtc_signal_output_disable .....       | 67 |
| 表 3-55. 函数 bkp_rtc_output_select .....               | 68 |
| 表 3-56. 函数 bkp_rtc_clock_output_select .....         | 69 |
| 表 3-57. 函数 bkp_rtc_clock_calibration_direction ..... | 69 |
| 表 3-58. 函数 bkp_rtc_calibration_value_set .....       | 70 |
| 表 3-59. 函数 bkp_osc32in_pin_select .....              | 70 |
| 表 3-60. 函数 bkp_tamper_detection_enable .....         | 71 |
| 表 3-61. 函数 bkp_tamper_detection_disable .....        | 71 |
| 表 3-62. 函数 bkp_tamper_active_level_set .....         | 72 |
| 表 3-63. 函数 bkp_tamper_interrupt_enable .....         | 72 |
| 表 3-64. 函数 bkp_tamper_interrupt_disable .....        | 73 |
| 表 3-65. 函数 bkp_flag_get .....                        | 73 |
| 表 3-66. 函数 bkp_flag_clear .....                      | 74 |
| 表 3-67. 函数 bkp_interrupt_flag_get .....              | 74 |
| 表 3-68. 函数 bkp_interrupt_flag_clear .....            | 75 |
| 表 3-69. CAN 寄存器 .....                                | 75 |
| 表 3-70. CAN 库函数 .....                                | 77 |
| 表 3-71. 结构体 can_error_counter_struct .....           | 78 |
| 表 3-72. 结构体 can_parameter_struct .....               | 78 |
| 表 3-73. 结构体 can_mailbox_descriptor_struct .....      | 79 |
| 表 3-74. 结构体 can_rx_fifo_struct .....                 | 79 |
| 表 3-75. 结构体 can_fd_parameter_struct .....            | 80 |
| 表 3-76. 结构体 can_rx_fifo_id_filter_struct .....       | 80 |
| 表 3-77. 结构体 can_fifo_parameter_struct .....          | 80 |
| 表 3-78. 结构体 can_pn_mode_filter_struct .....          | 81 |
| 表 3-79. 结构体 can_pn_mode_config_struct .....          | 81 |
| 表 3-80. 结构体 can_crc_struct .....                     | 81 |
| 表 3-81. 枚举类型 can_interrupt_enum .....                | 81 |
| 表 3-82. 枚举类型 can_flag_enum .....                     | 83 |

|  |     |
|--|-----|
| 表 3-83. 枚举类型 can_interrupt_flag_enum.....                | 85  |
| 表 3-84. 枚举类型 can_operation_modes_enum.....               | 87  |
| 表 3-85. 枚举类型 can_struct_type_enum .....                  | 88  |
| 表 3-86. 枚举类型 can_error_state_enum .....                  | 88  |
| 表 3-87. 函数 can_deinit .....                              | 88  |
| 表 3-88. 函数 can_software_reset .....                      | 89  |
| 表 3-89. 函数 can_init .....                                | 89  |
| 表 3-90. 函数 can_struct_para_init.....                     | 90  |
| 表 3-91. 函数 can_private_filter_config .....               | 91  |
| 表 3-92. 函数 can_operation_mode_enter .....                | 91  |
| 表 3-93. 函数 can_operation_mode_get .....                  | 92  |
| 表 3-94. 函数 can_inactive_mode_exit .....                  | 92  |
| 表 3-95. 函数 can_pn_mode_exit .....                        | 93  |
| 表 3-96. 函数 can_fd_config.....                            | 94  |
| 表 3-97. 函数 can_bitrate_switch_enable .....               | 94  |
| 表 3-98. 函数 can_bitrate_switch_disable .....              | 95  |
| 表 3-99. 函数 can_tdc_get .....                             | 95  |
| 表 3-100. 函数 can_tdc_enable .....                         | 96  |
| 表 3-101. 函数 can_tdc_disable .....                        | 96  |
| 表 3-102. 函数 can_rx_fifo_config .....                     | 97  |
| 表 3-103. 函数 can_rx_fifo_filter_table_config.....         | 97  |
| 表 3-104. 函数 can_rx_fifo_read .....                       | 98  |
| 表 3-105. 函数 can_rx_fifo_filter_matching_number_get ..... | 99  |
| 表 3-106. 函数 can_rx_fifo_clear .....                      | 99  |
| 表 3-107. 函数 can_ram_address_get.....                     | 100 |
| 表 3-108. 函数 can_mailbox_config .....                     | 100 |
| 表 3-109. 函数 can_mailbox_transmit_abort.....              | 101 |
| 表 3-110. 函数 can_mailbox_transmit_inactive.....           | 102 |
| 表 3-111. 函数 can_mailbox_receive_data_read .....          | 102 |
| 表 3-112. 函数 can_mailbox_receive_lock.....                | 103 |
| 表 3-113. 函数 can_mailbox_receive_unlock .....             | 103 |
| 表 3-114. 函数 can_mailbox_receive_inactive .....           | 104 |
| 表 3-115. 函数 can_mailbox_code_get .....                   | 105 |
| 表 3-116. 函数 can_error_counter_config .....               | 105 |
| 表 3-117. 函数 can_error_counter_get .....                  | 106 |
| 表 3-118. 函数 can_error_state_get.....                     | 106 |
| 表 3-119. 函数 can_crc_get .....                            | 107 |
| 表 3-120. 函数 can_pn_mode_config.....                      | 108 |
| 表 3-121. 函数 can_pn_mode_filter_config .....              | 108 |
| 表 3-122. 函数 can_pn_mode_num_of_match_get .....           | 109 |
| 表 3-123. 函数 can_pn_mode_data_read.....                   | 109 |
| 表 3-124. 函数 can_self_reception_enable.....               | 110 |
| 表 3-125. 函数 can_self_reception_disable.....              | 111 |
| 表 3-126. 函数 can_transmit_abort_enable .....              | 111 |



|  |     |
|--|-----|
| 表 3-127. 函数 can_transmit_abort_disable .....       | 112 |
| 表 3-128. 函数 can_auto_busoff_recovery_enable .....  | 112 |
| 表 3-129. 函数 can_auto_busoff_recovery_disable ..... | 113 |
| 表 3-130. 函数 can_time_sync_enable.....              | 113 |
| 表 3-131. 函数 can_time_sync_disable.....             | 114 |
| 表 3-132. 函数 can_edge_filter_mode_enable .....      | 114 |
| 表 3-133. 函数 can_edge_filter_mode_disable .....     | 115 |
| 表 3-134. 函数 can_ped_mode_enable .....              | 115 |
| 表 3-135. 函数 can_ped_mode_disable .....             | 116 |
| 表 3-136. 函数 can_arbitration_delay_bits_config..... | 116 |
| 表 3-137. 函数 can_bsp_mode_config.....               | 117 |
| 表 3-138. 函数 can_flag_get.....                      | 117 |
| 表 3-139. 函数 can_flag_clear.....                    | 118 |
| 表 3-140. 函数 can_interrupt_enable .....             | 119 |
| 表 3-141. 函数 can_interrupt_disable .....            | 119 |
| 表 3-142. 函数 can_interrupt_flag_get.....            | 120 |
| 表 3-143. 函数 can_interrupt_flag_clear.....          | 120 |
| 表 3-144. CMP 寄存器 .....                             | 121 |
| 表 3-145. CMP 库函数 .....                             | 121 |
| 表 3-146. 枚举类型 cmp_enum.....                        | 122 |
| 表 3-147. 函数 cmp_deinit.....                        | 122 |
| 表 3-148. 函数 cmp_mode_init.....                     | 122 |
| 表 3-149. 函数 cmp_noninverting_input_select.....     | 124 |
| 表 3-150. 函数 cmp_output_init.....                   | 125 |
| 表 3-151. 函数 cmp_blanking_init .....                | 126 |
| 表 3-152. 函数 cmp_enable .....                       | 127 |
| 表 3-153. 函数 cmp_disable .....                      | 127 |
| 表 3-154. 函数 cmp_lock_enable .....                  | 128 |
| 表 3-155. 函数 cmp_voltage_scaler_enable .....        | 128 |
| 表 3-156. 函数 cmp_voltage_scaler_disable .....       | 129 |
| 表 3-157. 函数 cmp_scaler_bridge_enable.....          | 129 |
| 表 3-158. 函数 cmp_scaler_bridge_disable.....         | 130 |
| 表 3-159. 函数 cmp_output_level_get.....              | 130 |
| 表 3-160. CRC 寄存器 .....                             | 131 |
| 表 3-161. CRC 库函数 .....                             | 131 |
| 表 3-162. 函数 crc_deinit.....                        | 131 |
| 表 3-163. 函数 crc_reverse_output_data_enable .....   | 132 |
| 表 3-164. 函数 crc_reverse_output_data_disable .....  | 132 |
| 表 3-165. 函数 crc_data_register_reset .....          | 133 |
| 表 3-166. 函数 crc_data_register_read .....           | 133 |
| 表 3-167. 函数 crc_free_data_register_read.....       | 134 |
| 表 3-168. 函数 crc_free_data_register_write.....      | 134 |
| 表 3-169. 函数 crc_init_data_register_write.....      | 135 |
| 表 3-170. 函数 crc_input_data_reverse_config.....     | 135 |

|  |     |
|--|-----|
| 表 3-171. 函数 crc_polynomial_size_set .....      | 136 |
| 表 3-172. 函数 crc_polynomial_set .....           | 137 |
| 表 3-173. 函数 crc_single_data_calculate .....    | 137 |
| 表 3-174. 函数 crc_block_data_calculate .....     | 138 |
| 表 3-175. DBG 寄存器 .....                         | 139 |
| 表 3-176. DBG 库函数 .....                         | 139 |
| 表 3-177. 枚举类型 dbg_periph_enum .....            | 139 |
| 表 3-178. 函数 dbg_deinit.....                    | 140 |
| 表 3-179. 函数 dbg_id_get.....                    | 140 |
| 表 3-180. 函数 dbg_low_power_enable .....         | 141 |
| 表 3-181. 函数 dbg_low_power_disable .....        | 142 |
| 表 3-182. 函数 dbg_periph_enable.....             | 142 |
| 表 3-183. 函数 dbg_periph_disable.....            | 143 |
| 表 3-184. DAC 寄存器 .....                         | 144 |
| 表 3-185. DAC 库函数 .....                         | 144 |
| 表 3-186. 函数 dac_deinit .....                   | 145 |
| 表 3-187. 函数 dac_enable .....                   | 145 |
| 表 3-188. 函数 dac_disable .....                  | 146 |
| 表 3-189. 函数 dac_dma_enable.....                | 146 |
| 表 3-190. 函数 dac_dma_disable.....               | 147 |
| 表 3-191. 函数 dac_gpio_connect_config .....      | 148 |
| 表 3-192. 函数 dac_output_buffer_enable.....      | 148 |
| 表 3-193. 函数 dac_output_buffer_disable.....     | 149 |
| 表 3-194. 函数 dac_output_value_get.....          | 149 |
| 表 3-195. 函数 dac_data_set .....                 | 150 |
| 表 3-196. 函数 dac_trigger_enable.....            | 151 |
| 表 3-197. 函数 dac_trigger_disable.....           | 151 |
| 表 3-198. 函数 dac_trigger_source_config.....     | 152 |
| 表 3-199. 函数 dac_software_trigger_enable .....  | 153 |
| 表 3-200. 函数 dac_wave_mode_config .....         | 153 |
| 表 3-201. 函数 dac_lfsr_noise_config.....         | 154 |
| 表 3-202. 函数 dac_triangle_noise_config .....    | 155 |
| 表 3-203. 函数 dac_flag_get.....                  | 156 |
| 表 3-204. 函数 dac_flag_clear.....                | 156 |
| 表 3-205. 函数 dac_interrupt_enable .....         | 157 |
| 表 3-206. 函数 dac_interrupt_disable .....        | 157 |
| 表 3-207. 函数 dac_interrupt_flag_get.....        | 158 |
| 表 3-208. 函数 dac_interrupt_flag_clear.....      | 159 |
| 表 3-209. DMA 寄存器 .....                         | 160 |
| 表 3-210. DMAMUX 寄存器.....                       | 160 |
| 表 3-211. DMA 库函数 .....                         | 160 |
| 表 3-212. DMAMUX 库函数.....                       | 161 |
| 表 3-213. 结构体 dma_parameter_struct .....        | 162 |
| 表 3-214. 结构体 dmamux_sync_parameter_struct..... | 162 |

|  |     |
|--|-----|
| 表 3-215. 结构体 dmamux_gen_parameter_struct .....             | 162 |
| 表 3-216. 枚举 dma_channel_enum .....                         | 163 |
| 表 3-217. 枚举 dmamux_multiplexer_channel_enum .....          | 163 |
| 表 3-218. 枚举 dmamux_generator_channel_enum .....            | 164 |
| 表 3-219. 枚举 dmamux_interrupt_enum.....                     | 164 |
| 表 3-220. 枚举 dmamux_flag_enum .....                         | 165 |
| 表 3-221. 枚举 dmamux_interrupt_flag_enum .....               | 165 |
| 表 3-222. 函数 dma_deinit.....                                | 166 |
| 表 3-223. 函数 dma_struct_para_init .....                     | 167 |
| 表 3-224. 函数 dma_init .....                                 | 167 |
| 表 3-225. 函数 dma_circulation_enable .....                   | 168 |
| 表 3-226. 函数 dma_circulation_disable.....                   | 169 |
| 表 3-227. 函数 dma_memory_to_memory_enable .....              | 170 |
| 表 3-228. 函数 dma_memory_to_memory_disable .....             | 170 |
| 表 3-229. 函数 dma_channel_enable .....                       | 171 |
| 表 3-230. 函数 dma_channel_disable .....                      | 171 |
| 表 3-231. 函数 dma_periph_address_config.....                 | 172 |
| 表 3-232. 函数 dma_memory_address_config.....                 | 173 |
| 表 3-233. 函数 dma_transfer_number_config.....                | 173 |
| 表 3-234. 函数 dma_transfer_number_get .....                  | 174 |
| 表 3-235. 函数 dma_priority_config .....                      | 175 |
| 表 3-236. 函数 dma_memory_width_config .....                  | 175 |
| 表 3-237. 函数 dma_periph_width_config .....                  | 176 |
| 表 3-238. 函数 dma_memory_increase_enable .....               | 177 |
| 表 3-239. 函数 dma_memory_increase_disable .....              | 178 |
| 表 3-240. 函数 dma_periph_increase_enable .....               | 178 |
| 表 3-241. 函数 dma_periph_increase_disable .....              | 179 |
| 表 3-242. 函数 dma_transfer_direction_config.....             | 179 |
| 表 3-243. 函数 dma_flag_get.....                              | 180 |
| 表 3-244. 函数 dma_flag_clear.....                            | 181 |
| 表 3-245. 函数 dma_interrupt_enable .....                     | 182 |
| 表 3-246. 函数 dma_interrupt_disable .....                    | 182 |
| 表 3-247. 函数 dma_interrupt_flag_get.....                    | 183 |
| 表 3-248. 函数 dma_interrupt_flag_clear.....                  | 184 |
| 表 3-249. 函数 dmamux_sync_struct_para_init.....              | 185 |
| 表 3-250. 函数 dmamux_synchronization_init.....               | 185 |
| 表 3-251. 函数 dmamux_synchronization_enable .....            | 186 |
| 表 3-252. 函数 dmamux_synchronization_disable.....            | 187 |
| 表 3-253. 函数 dmamux_event_generation_enable.....            | 187 |
| 表 3-254. 函数 dmamux_event_generation_disable.....           | 188 |
| 表 3-255. 函数 dmamux_gen_struct_para_init.....               | 188 |
| 表 3-256. 函数 dmamux_request_generator_init.....             | 189 |
| 表 3-257. 函数 dmamux_request_generator_channel_enable .....  | 189 |
| 表 3-258. 函数 dmamux_request_generator_channel_disable ..... | 190 |

|  |     |
|--|-----|
| 表 3-259. 函数 dmamux_synchronization_polarity_config ..... | 191 |
| 表 3-260. 函数 dmamux_request_forward_number_config .....   | 191 |
| 表 3-261. 函数 dmamux_sync_id_config .....                  | 192 |
| 表 3-262. 函数 dmamux_request_id_config .....               | 194 |
| 表 3-263. 函数 dma_interrupt_disable .....                  | 198 |
| 表 3-264. 函数 dmamux_request_generate_number_config .....  | 199 |
| 表 3-265. 函数 dmamux_trigger_id_config .....               | 199 |
| 表 3-266. 函数 dmamux_flag_get .....                        | 201 |
| 表 3-267. 函数 dmamux_flag_clear .....                      | 202 |
| 表 3-268. 函数 dmamux_interrupt_enable .....                | 203 |
| 表 3-269. 函数 dmamux_interrupt_disable .....               | 204 |
| 表 3-270. 函数 dmamux_interrupt_flag_get .....              | 206 |
| 表 3-271. 函数 dmamux_interrupt_flag_clear .....            | 207 |
| 表 3-272. EXTI 寄存器 .....                                  | 208 |
| 表 3-273. EXTI 库函数 .....                                  | 209 |
| 表 3-274. 枚举类型 exti_line_enum .....                       | 209 |
| 表 3-275. 枚举类型 exti_mode_enum .....                       | 210 |
| 表 3-276. 枚举类型 exti_trig_type_enum .....                  | 210 |
| 表 3-277. 函数 exti_deinit .....                            | 210 |
| 表 3-278. 函数 exti_init .....                              | 211 |
| 表 3-279. 函数 exti_interrupt_enable .....                  | 211 |
| 表 3-280. 函数 exti_interrupt_disable .....                 | 212 |
| 表 3-281. 函数 exti_event_enable .....                      | 212 |
| 表 3-282. 函数 exti_event_disable .....                     | 213 |
| 表 3-283. 函数 exti_software_interrupt_enable .....         | 213 |
| 表 3-284. 函数 exti_software_interrupt_disable .....        | 214 |
| 表 3-285. 函数 exti_flag_get .....                          | 214 |
| 表 3-286. 函数 exti_flag_clear .....                        | 215 |
| 表 3-287. 函数 exti_interrupt_flag_get .....                | 215 |
| 表 3-288. 函数 exti_interrupt_flag_clear .....              | 216 |
| 表 3-289. FMC 寄存器 .....                                   | 216 |
| 表 3-290. FMC 库函数 .....                                   | 217 |
| 表 3-291. 枚举类型 fmc_state_enum .....                       | 218 |
| 表 3-292. 枚举类型 fmc_sram_mode_enum .....                   | 218 |
| 表 3-293. 枚举类型 fmc_area_enum .....                        | 219 |
| 表 3-294. 枚举类型 fmc_flag_enum .....                        | 219 |
| 表 3-295. 枚举类型 fmc_interrupt_flag_enum .....              | 220 |
| 表 3-296. 枚举类型 fmc_interrupt_enum .....                   | 221 |
| 表 3-297. 函数 fmc_unlock .....                             | 222 |
| 表 3-298. 函数 fmc_bank0_unlock .....                       | 222 |
| 表 3-299. 函数 fmc_bank1_unlock .....                       | 223 |
| 表 3-300. 函数 fmc_lock .....                               | 223 |
| 表 3-301. 函数 fmc_bank0_lock .....                         | 224 |
| 表 3-302. 函数 fmc_bank1_lock .....                         | 224 |

|   |     |
|---|-----|
| 表 3-303. 函数 fmc_wscnt_set .....                 | 225 |
| 表 3-304. 函数 fmc_prefetch_enable .....           | 225 |
| 表 3-305. 函数 fmc_prefetch_disable .....          | 226 |
| 表 3-306. 函数 fmc_cache_enable .....              | 226 |
| 表 3-307. 函数 fmc_cache_disable .....             | 227 |
| 表 3-308. 函数 fmc_cache_reset_enable .....        | 227 |
| 表 3-309. 函数 fmc_cache_reset_disable .....       | 228 |
| 表 3-310. 函数 fmc_powerdown_mode_set .....        | 228 |
| 表 3-311. 函数 fmc_sleep_mode_set .....            | 229 |
| 表 3-312. 函数 fmc_sram_mode_config .....          | 229 |
| 表 3-313. 函数 fmc_sram_mode_get .....             | 230 |
| 表 3-314. 函数 fmc_blank_check .....               | 230 |
| 表 3-315. 函数 fmc_page_erase .....                | 231 |
| 表 3-316. 函数 fmc_bank0_mass_erase .....          | 232 |
| 表 3-317. 函数 fmc_bank1_mass_erase .....          | 233 |
| 表 3-318. 函数 fmc_dflash_mass_erase .....         | 233 |
| 表 3-319. 函数 fmc_mass_erase .....                | 234 |
| 表 3-320. 函数 fmc_doubleword_program .....        | 235 |
| 表 3-321. 函数 fmc_fast_program .....              | 236 |
| 表 3-322. 函数 otp_doubleword_program .....        | 237 |
| 表 3-323. 函数 ob_unlock .....                     | 238 |
| 表 3-324. 函数 ob_lock .....                       | 238 |
| 表 3-325. 函数 ob_reset .....                      | 239 |
| 表 3-326. 函数 ob_erase .....                      | 239 |
| 表 3-327. 函数 ob_write_protection_enable .....    | 240 |
| 表 3-328. 函数 ob_security_protection_config ..... | 241 |
| 表 3-329. 函数 ob_user_write .....                 | 242 |
| 表 3-330. 函数 ob_data_program .....               | 243 |
| 表 3-331. 函数 ob_user_get .....                   | 244 |
| 表 3-332. 函数 ob_data_get .....                   | 244 |
| 表 3-333. 函数 ob_write_protection_get .....       | 245 |
| 表 3-334. 函数 ob_bk1_write_protection_get .....   | 245 |
| 表 3-335. 函数 ob_df_write_protection_get .....    | 246 |
| 表 3-336. 函数 ob_plevel_get .....                 | 246 |
| 表 3-337. 函数 ob1_lock_config .....               | 247 |
| 表 3-338. 函数 ob1_parameter_config .....          | 248 |
| 表 3-339. 函数 dflash_size_get .....               | 248 |
| 表 3-340. 函数 fmc_flag_get .....                  | 249 |
| 表 3-341. 函数 fmc_flag_clear .....                | 249 |
| 表 3-342. 函数 fmc_interrupt_enable .....          | 251 |
| 表 3-343. 函数 fmc_interrupt_disable .....         | 251 |
| 表 3-344. 函数 fmc_interrupt_flag_get .....        | 252 |
| 表 3-345. 函数 fmc_interrupt_flag_clear .....      | 252 |
| 表 3-346. FWDGT 寄存器 .....                        | 253 |

|  |     |
|--|-----|
| 表 3-347. FWDGT 库函数 .....                             | 253 |
| 表 3-348. 函数 fwdgt_write_enable .....                 | 253 |
| 表 3-349. 函数 fwdgt_write_disable .....                | 254 |
| 表 3-350. 函数 fwdgt_enable .....                       | 254 |
| 表 3-351. 函数 fwdgt_prescaler_value_config .....       | 255 |
| 表 3-352. 函数 fwdgt_reload_value_config .....          | 255 |
| 表 3-353. 函数 fwdgt_window_value_config .....          | 256 |
| 表 3-354. 函数 fwdgt_counter_reload .....               | 256 |
| 表 3-355. 函数 fwdgt_config .....                       | 257 |
| 表 3-356. 函数 fwdgt_flag_get .....                     | 258 |
| 表 3-357. GPIO 寄存器 .....                              | 259 |
| 表 3-358. GPIO 库函数 .....                              | 259 |
| 表 3-359. 函数 gpio_deinit .....                        | 259 |
| 表 3-360. 函数 gpio_mode_set .....                      | 260 |
| 表 3-361. 函数 gpio_output_options_set .....            | 261 |
| 表 3-362. 函数 gpio_bit_set .....                       | 262 |
| 表 3-363. 函数 gpio_bit_reset .....                     | 263 |
| 表 3-364. 函数 gpio_bit_write .....                     | 263 |
| 表 3-365. 函数 gpio_port_write .....                    | 264 |
| 表 3-366. 函数 gpio_input_bit_get .....                 | 264 |
| 表 3-367. 函数 gpio_input_port_get .....                | 265 |
| 表 3-368. 函数 gpio_output_bit_get .....                | 266 |
| 表 3-369. 函数 gpio_output_port_get .....               | 266 |
| 表 3-370. 函数 gpio_af_set .....                        | 267 |
| 表 3-371. 函数 gpio_pin_lock .....                      | 268 |
| 表 3-372. 函数 gpio_bit_toggle .....                    | 268 |
| 表 3-373. 函数 gpio_port_toggle .....                   | 269 |
| 表 3-374. I2C 寄存器 .....                               | 270 |
| 表 3-375. I2C 库函数 .....                               | 270 |
| 表 3-376. 枚举类型 i2c_interrupt_flag_enum .....          | 272 |
| 表 3-377. 函数 i2c_deinit .....                         | 272 |
| 表 3-378. 函数 i2c_timing_config .....                  | 273 |
| 表 3-379. 函数 i2c_digital_noise_filter_config .....    | 273 |
| 表 3-380. 函数 i2c_analog_noise_filter_enable .....     | 274 |
| 表 3-381. 函数 i2c_analog_noise_filter_disable .....    | 275 |
| 表 3-382. 函数 i2c_master_clock_config .....            | 275 |
| 表 3-383. 函数 i2c_master_addressing .....              | 276 |
| 表 3-384. 函数 i2c_address10_header_enable .....        | 277 |
| 表 3-385. 函数 i2c_address10_header_disable .....       | 277 |
| 表 3-386. 函数 i2c_address10_enable .....               | 278 |
| 表 3-387. 函数 i2c_address10_disable .....              | 278 |
| 表 3-388. 函数 i2c_automatic_end_enable .....           | 279 |
| 表 3-389. 函数 i2c_automatic_end_disable .....          | 279 |
| 表 3-390. 函数 i2c_slave_response_to_gcall_enable ..... | 280 |

|  |     |
|--|-----|
| 表 3-391. 函数 i2c_slave_response_to_gcall_disable..... | 280 |
| 表 3-392. 函数 i2c_stretch_scl_low_enable .....         | 281 |
| 表 3-393. 函数 i2c_stretch_scl_low_disable .....        | 281 |
| 表 3-394. 函数 i2c_address_config .....                 | 282 |
| 表 3-395. 函数 i2c_address_bit_compare_config.....      | 282 |
| 表 3-396. 函数 i2c_address_disable.....                 | 283 |
| 表 3-397. 函数 i2c_second_address_config.....           | 284 |
| 表 3-398. 函数 i2c_second_address_disable .....         | 285 |
| 表 3-399. 函数 i2c_receved_address_get .....            | 285 |
| 表 3-400. 函数 i2c_slave_byte_control_enable .....      | 286 |
| 表 3-401. 函数 i2c_slave_byte_control_disable .....     | 286 |
| 表 3-402. 函数 i2c_nack_enable .....                    | 287 |
| 表 3-403. 函数 i2c_nack_disable .....                   | 287 |
| 表 3-404. 函数 i2c_enable.....                          | 288 |
| 表 3-405. 函数 i2c_disable.....                         | 288 |
| 表 3-406. 函数 i2c_start_on_bus .....                   | 289 |
| 表 3-407. 函数 i2c_stop_on_bus.....                     | 289 |
| 表 3-408. 函数 i2c_data_transmit .....                  | 290 |
| 表 3-409. 函数 i2c_data_receive .....                   | 290 |
| 表 3-410. 函数 i2c_reload_enable.....                   | 291 |
| 表 3-411. 函数 i2c_reload_disable .....                 | 291 |
| 表 3-412. 函数 i2c_transfer_byte_number_config .....    | 292 |
| 表 3-413. 函数 i2c_dma_enable .....                     | 292 |
| 表 3-414. 函数 i2c_dma_disable .....                    | 293 |
| 表 3-415. 函数 i2c_pec_transfer .....                   | 294 |
| 表 3-416. 函数 i2c_pec_enable .....                     | 294 |
| 表 3-417. 函数 i2c_pec_disable .....                    | 295 |
| 表 3-418. 函数 i2c_pec_value_get .....                  | 295 |
| 表 3-419. 函数 i2c_smbus_alert_enable .....             | 296 |
| 表 3-420. 函数 i2c_smbus_alert_disable .....            | 296 |
| 表 3-421. 函数 i2c_smbus_default_addr_enable .....      | 297 |
| 表 3-422. 函数 i2c_smbus_default_addr_disable .....     | 297 |
| 表 3-423. 函数 i2c_smbus_host_addr_enable .....         | 298 |
| 表 3-424. 函数 i2c_smbus_host_addr_disable .....        | 298 |
| 表 3-425. 函数 i2c_extented_clock_timeout_enable .....  | 299 |
| 表 3-426. 函数 i2c_extented_clock_timeout_disable ..... | 299 |
| 表 3-427. 函数 i2c_clock_timeout_enable .....           | 300 |
| 表 3-428. 函数 i2c_clock_timeout_disable .....          | 300 |
| 表 3-429. 函数 i2c_bus_timeout_b_config .....           | 301 |
| 表 3-430. 函数 i2c_bus_timeout_a_config.....            | 301 |
| 表 3-431. 函数 i2c_idle_clock_timeout_config.....       | 302 |
| 表 3-432. 函数 i2c_flag_get .....                       | 302 |
| 表 3-433. 函数 i2c_flag_clear .....                     | 303 |
| 表 3-434. 函数 i2c_interrupt_enable.....                | 304 |



|   |     |
|---|-----|
| 表 3-435. 函数 i2c_interrupt_disable.....                  | 305 |
| 表 3-436. 函数 i2c_interrupt_flag_get .....                | 306 |
| 表 3-437. 函数 i2c_interrupt_flag_clear .....              | 307 |
| 表 3-438. MFCOM 寄存器.....                                 | 308 |
| 表 3-439. MFCOM 库函数.....                                 | 308 |
| 表 3-440. 结构体 rtc_parameter_struct .....                 | 309 |
| 表 3-441. 结构体 mfcom_shifter_parameter_struct .....       | 310 |
| 表 3-442. 函数 mfcom_deinit.....                           | 310 |
| 表 3-443. 函数 mfcom_software_reset.....                   | 311 |
| 表 3-444. 函数 mfcom_enable.....                           | 311 |
| 表 3-445. 函数 mfcom_disable.....                          | 312 |
| 表 3-446. 函数 mfcom_timer_struct_para_init .....          | 312 |
| 表 3-447. 函数 mfcom_shifter_struct_para_init .....        | 313 |
| 表 3-448. 函数 mfcom_timer_init.....                       | 313 |
| 表 3-449. 函数 mfcom_shifter_init.....                     | 314 |
| 表 3-450. 函数 mfcom_timer_pin_config .....                | 315 |
| 表 3-451. 函数 mfcom_shifter_pin_config.....               | 316 |
| 表 3-452. 函数 mfcom_timer_enable.....                     | 317 |
| 表 3-453. 函数 mfcom_shifter_enable.....                   | 317 |
| 表 3-454. 函数 mfcom_timer_disable.....                    | 318 |
| 表 3-455. 函数 mfcom_shifter_disable.....                  | 319 |
| 表 3-456. 函数 mfcom_timer_cmpvalue_set .....              | 319 |
| 表 3-457. 函数 mfcom_timer_cmpvalue_get .....              | 320 |
| 表 3-458. 函数 mfcom_timer_dismode_set .....               | 320 |
| 表 3-459. 函数 mfcom_shifter_stopbit_set.....              | 321 |
| 表 3-460. 函数 mfcom_buffer_write .....                    | 322 |
| 表 3-461. 函数 mfcom_buffer_read .....                     | 323 |
| 表 3-462. 函数 mfcom_shifter_flag_get .....                | 323 |
| 表 3-463. 函数 mfcom_shifter_error_flag_get.....           | 324 |
| 表 3-464. 函数 mfcom_timer_flag_get .....                  | 324 |
| 表 3-465. 函数 mfcom_shifter_interrupt_flag_get .....      | 325 |
| 表 3-466. 函数 mfcom_shifter_error_interrupt_flag_get..... | 325 |
| 表 3-467. 函数 mfcom_timer_interrupt_flag_get .....        | 326 |
| 表 3-468. 函数 mfcom_shifter_flag_clear .....              | 326 |
| 表 3-469. 函数 mfcom_shifter_error_flag_clear.....         | 327 |
| 表 3-470. 函数 mfcom_timer_flag_clear .....                | 327 |
| 表 3-471. 函数 mfcom_shifter_interrupt_enable.....         | 328 |
| 表 3-472. 函数 mfcom_shifter_error_interrupt_enable .....  | 329 |
| 表 3-473. 函数 mfcom_timer_interrupt_enable .....          | 329 |
| 表 3-474. 函数 mfcom_shifter_dma_enable .....              | 330 |
| 表 3-475. 函数 mfcom_shifter_interrupt_disable.....        | 330 |
| 表 3-476. 函数 mfcom_shifter_error_interrupt_disable ..... | 331 |
| 表 3-477. 函数 mfcom_timer_interrupt_disable .....         | 331 |
| 表 3-478. 函数 mfcom_shifter_dma_disable .....             | 332 |



|   |     |
|---|-----|
| 表 3-479. NVIC 寄存器.....                            | 333 |
| 表 3-480. SysTick 寄存器.....                         | 333 |
| 表 3-481. MISC 库函数 .....                           | 333 |
| 表 3-482. 枚举类型 IRQn_Type .....                     | 334 |
| 表 3-483. 函数 nvic_priority_group_set.....          | 335 |
| 表 3-484. 函数 nvic_irq_enable.....                  | 336 |
| 表 3-485. 函数 nvic_irq_disable.....                 | 337 |
| 表 3-486. 函数 nvic_system_reset.....                | 337 |
| 表 3-487. 函数 nvic_vector_table_set .....           | 338 |
| 表 3-488. 函数 system_lowpower_set.....              | 338 |
| 表 3-489. 函数 system_lowpower_reset .....           | 339 |
| 表 3-490. 函数 systick_clksource_set.....            | 340 |
| 表 3-491. PMU 寄存器 .....                            | 340 |
| 表 3-492. PMU 库函数 .....                            | 340 |
| 表 3-493. 函数 pmu_deinit.....                       | 341 |
| 表 3-494. 函数 pmu_lvd_select.....                   | 342 |
| 表 3-495. 函数 pmu_lvd_disable.....                  | 342 |
| 表 3-496. 函数 pmu_ovd_select.....                   | 343 |
| 表 3-497. 函数 pmu_ovd_disable .....                 | 343 |
| 表 3-498. 函数 pmu_lowdriver_mode_enable .....       | 344 |
| 表 3-499. 函数 pmu_lowdriver_mode_disable .....      | 344 |
| 表 3-500. 函数 pmu_sram1_poweroff_mode_enable .....  | 345 |
| 表 3-501. 函数 pmu_sram1_poweroff_mode_disable ..... | 345 |
| 表 3-502. 函数 pmu_sram2_poweroff_mode_enable .....  | 346 |
| 表 3-503. 函数 pmu_sram2_poweroff_mode_disable ..... | 346 |
| 表 3-504. 函数 pmu_to_sleepmode .....                | 347 |
| 表 3-505. 函数 pmu_to_deepsleepmode.....             | 347 |
| 表 3-506. 函数 pmu_to_standbymode .....              | 348 |
| 表 3-507. 函数 pmu_wakeup_pin_enable .....           | 349 |
| 表 3-508. 函数 pmu_wakeup_pin_disable .....          | 349 |
| 表 3-509. 函数 pmu_backup_write_enable.....          | 350 |
| 表 3-510. 函数 pmu_backup_write_disable.....         | 350 |
| 表 3-511. 函数 pmu_flag_get.....                     | 351 |
| 表 3-512. 函数 pmu_flag_clear .....                  | 351 |
| 表 3-513. RCU 寄存器 .....                            | 352 |
| 表 3-514. RCU 库函数 .....                            | 353 |
| 表 3-515. 枚举类型 rcu_periph_enum .....               | 354 |
| 表 3-516. 枚举类型 rcu_periph_sleep_enum .....         | 355 |
| 表 3-517. 枚举类型 rcu_periph_reset_enum.....          | 355 |
| 表 3-518. 枚举类型 rcu_flag_enum .....                 | 356 |
| 表 3-519. 枚举类型 rcu_int_flag_enum.....              | 357 |
| 表 3-520. 枚举类型 rcu_int_flag_clear_enum .....       | 357 |
| 表 3-521. 枚举类型 rcu_int_enum .....                  | 357 |
| 表 3-522. 枚举类型 rcu_osci_type_enum.....             | 358 |

|  |     |
|--|-----|
| 表 3-523. 枚举类型 rcu_clock_freq_enum .....            | 358 |
| 表 3-524. 函数 rcu_deinit.....                        | 358 |
| 表 3-525. 函数 rcu_periph_clock_enable.....           | 359 |
| 表 3-526. 函数 rcu_periph_clock_disable.....          | 360 |
| 表 3-527. 函数 rcu_periph_reset_enable .....          | 360 |
| 表 3-528. 函数 rcu_periph_reset_disable .....         | 361 |
| 表 3-529. 函数 rcu_periph_clock_sleep_enable .....    | 362 |
| 表 3-530. 函数 rcu_periph_clock_sleep_disable .....   | 363 |
| 表 3-531. 函数 rcu_bkp_reset_enable.....              | 363 |
| 表 3-532. 函数 rcu_bkp_reset_disable .....            | 364 |
| 表 3-533. 函数 rcu_system_clock_source_config .....   | 364 |
| 表 3-534. 函数 rcu_system_clock_source_get .....      | 365 |
| 表 3-535. 函数 rcu_ahb_clock_config.....              | 365 |
| 表 3-536. 函数 rcu_apb1_clock_config.....             | 366 |
| 表 3-537. 函数 rcu_apb2_clock_config.....             | 367 |
| 表 3-538. 函数 rcu_ckout_config.....                  | 367 |
| 表 3-539. 函数 rcu_pll_config .....                   | 368 |
| 表 3-540. 函数 rcu_double_pll_enable.....             | 369 |
| 表 3-541. 函数 rcu_double_pll_disable.....            | 369 |
| 表 3-542. 函数 rcu_system_reset_enable.....           | 370 |
| 表 3-543. 函数 rcu_system_reset_disable.....          | 370 |
| 表 3-544. 函数 rcu_adc_clock_config .....             | 371 |
| 表 3-545. 函数 rcu_rtc_clock_config.....              | 372 |
| 表 3-546. 函数 rcu_usart_clock_config .....           | 372 |
| 表 3-547. 函数 rcu_can_clock_config .....             | 373 |
| 表 3-548. 函数 rcu_lxtal_drive_capability_config..... | 374 |
| 表 3-549. 函数 rcu_osci_stab_wait.....                | 375 |
| 表 3-550. 函数 rcu_osci_on.....                       | 375 |
| 表 3-551. 函数 rcu_osci_off.....                      | 376 |
| 表 3-552. 函数 rcu_osci_bypass_mode_enable.....       | 376 |
| 表 3-553. 函数 rcu_osci_bypass_mode_disable.....      | 377 |
| 表 3-554. 函数 rcu_hxtal_frequency_scale_select ..... | 377 |
| 表 3-555. 函数 rcu_hxtal_prediv_config .....          | 378 |
| 表 3-556. 函数 rcu_irc8m_adjust_value_set.....        | 379 |
| 表 3-557. 函数 rcu_hxtal_clock_monitor_enable.....    | 379 |
| 表 3-558. 函数 rcu_hxtal_clock_monitor_disable.....   | 380 |
| 表 3-559. 函数 rcu_lxtal_clock_monitor_enable .....   | 380 |
| 表 3-560. 函数 rcu_lxtal_clock_monitor_disable .....  | 381 |
| 表 3-561. 函数 rcu_voltage_key_unlock .....           | 381 |
| 表 3-562. 函数 rcu_deepsleep_voltage_set.....         | 382 |
| 表 3-563. 函数 rcu_clock_freq_get.....                | 382 |
| 表 3-564. 函数 rcu_flag_get.....                      | 383 |
| 表 3-565. 函数 rcu_all_reset_flag_clear.....          | 384 |
| 表 3-566. 函数 rcu_interrupt_flag_get.....            | 385 |

|  |     |
|--|-----|
| 表 3-567. 函数 rcu_interrupt_flag_clear.....          | 386 |
| 表 3-568. 函数 rcu_interrupt_enable .....             | 387 |
| 表 3-569. 函数 rcu_interrupt_disable .....            | 387 |
| 表 3-570. RTC 寄存器.....                              | 388 |
| 表 3-571. RTC 库函数.....                              | 389 |
| 表 3-572. 函数 rtc_configuration_mode_enter.....      | 389 |
| 表 3-573. 函数 rtc_configuration_mode_exit .....      | 390 |
| 表 3-574. 函数 rtc_lwoff_wait.....                    | 390 |
| 表 3-575. 函数 rtc_register_sync_wait.....            | 391 |
| 表 3-576. 函数 rtc_counter_get.....                   | 391 |
| 表 3-577. Function rtc_counter_set.....             | 392 |
| 表 3-578. 函数 rtc_prescaler_set .....                | 392 |
| 表 3-579. 函数 rtc_alarm_config.....                  | 393 |
| 表 3-580. 函数 rtc_divider_get .....                  | 393 |
| 表 3-581. 函数 rtc_interrupt_enable .....             | 394 |
| 表 3-582. 函数 rtc_interrupt_disable .....            | 394 |
| 表 3-583. 函数 rtc_flag_get.....                      | 395 |
| 表 3-584. 函数 rtc_flag_clear.....                    | 396 |
| 表 3-585. 函数 rtc_interrupt_flag_get.....            | 396 |
| 表 3-586. 函数 rtc_interrupt_flag_clear.....          | 397 |
| 表 3-587. SPI/I2S 寄存器 .....                         | 398 |
| 表 3-588. SPI/I2S 库函数 .....                         | 398 |
| 表 3-589. 结构体 spi_parameter_struct.....             | 399 |
| 表 3-590. 函数 spi_i2s_deinit .....                   | 400 |
| 表 3-591. 函数 spi_struct_para_init.....              | 400 |
| 表 3-592. 函数 spi_init .....                         | 401 |
| 表 3-593. 函数 spi_enable .....                       | 402 |
| 表 3-594. 函数 spi_disable .....                      | 402 |
| 表 3-595. 函数 i2s_init.....                          | 403 |
| 表 3-596. 函数 i2s_psc_config .....                   | 404 |
| 表 3-597. 函数 i2s_enable.....                        | 405 |
| 表 3-598. 函数 i2s_disable.....                       | 406 |
| 表 3-599. 函数 spi_nss_output_enable .....            | 406 |
| 表 3-600. 函数 spi_nss_output_disable .....           | 407 |
| 表 3-601. 函数 spi_nss_internal_high.....             | 407 |
| 表 3-602. 函数 spi_nss_internal_low .....             | 408 |
| 表 3-603. 函数 spi_dma_enable .....                   | 408 |
| 表 3-604. 函数 spi_dma_disable .....                  | 409 |
| 表 3-605. 函数 spi_i2s_data_frame_format_config ..... | 410 |
| 表 3-606. 函数 spi_i2s_data_transmit.....             | 410 |
| 表 3-607. 函数 spi_i2s_data_receive .....             | 411 |
| 表 3-608. 函数 spi_bidirectional_transfer_config..... | 411 |
| 表 3-609. 函数 spi_crc_error_clear .....              | 412 |
| 表 3-610. 函数 spi_crc_polynomial_set .....           | 412 |

|  |     |
|--|-----|
| 表 3-611. 函数 spi_crc_polynomial_get .....           | 413 |
| 表 3-612. 函数 spi_crc_on .....                       | 414 |
| 表 3-613. 函数 spi_crc_off .....                      | 414 |
| 表 3-614. 函数 spi_crc_next .....                     | 415 |
| 表 3-615. 函数 spi_crc_get .....                      | 415 |
| 表 3-616. 函数 spi_crc_error_clear .....              | 416 |
| 表 3-617. 函数 spi_ti_mode_enable .....               | 416 |
| 表 3-618. 函数 spi_ti_mode_disable .....              | 417 |
| 表 3-619. 函数 spi_nssp_mode_enable .....             | 417 |
| 表 3-620. 函数 spi_nssp_mode_disable .....            | 418 |
| 表 3-621. 函数 spi_quad_enable .....                  | 418 |
| 表 3-622. 函数 spi_quad_disable .....                 | 419 |
| 表 3-623. 函数 spi_quad_write_enable .....            | 419 |
| 表 3-624. 函数 spi_quad_read_enable .....             | 420 |
| 表 3-625. 函数 spi_quad_io23_output_enable .....      | 420 |
| 表 3-626. 函数 spi_quad_io23_output_disable .....     | 421 |
| 表 3-627. 函数 spi_i2s_interrupt_enable .....         | 421 |
| 表 3-628. 函数 spi_i2s_interrupt_disable .....        | 422 |
| 表 3-629. 函数 spi_i2s_interrupt_flag_get .....       | 422 |
| 表 3-630. 函数 spi_i2s_flag_get .....                 | 423 |
| 表 3-631. SYSCFG 寄存器 .....                          | 425 |
| 表 3-632. SYSCFG 库函数 .....                          | 425 |
| 表 3-633. 函数 syscfg_deinit .....                    | 426 |
| 表 3-634. 函数 syscfg_exti_line_config .....          | 426 |
| 表 3-635. 函数 syscfg_pin_remap_enable .....          | 427 |
| 表 3-636. 函数 syscfg_pin_remap_disable .....         | 428 |
| 表 3-637. 函数 syscfg_adc_ch_remap_config .....       | 428 |
| 表 3-638. 函数 syscfg_timer_eti_sel .....             | 429 |
| 表 3-639. 函数 syscfg_timer_bkin_select_trigsel ..... | 430 |
| 表 3-640. 函数 syscfg_timer_bkin_select_gpio .....    | 430 |
| 表 3-641. 函数 syscfg_timer7_ch0n_select .....        | 431 |
| 表 3-642. 函数 syscfg_lock_config .....               | 431 |
| 表 3-643. 函数 syscfg_flag_get .....                  | 432 |
| 表 3-644. 函数 syscfg_flag_clear .....                | 433 |
| 表 3-645. 函数 syscfg_interrupt_enable .....          | 433 |
| 表 3-646. 函数 syscfg_interrupt_disable .....         | 434 |
| 表 3-647. 函数 syscfg_interrupt_flag_get .....        | 435 |
| 表 3-648. 函数 syscfg_bootmode_get .....              | 435 |
| 表 3-649. 函数 syscfg_sram_ecc_address_get .....      | 436 |
| 表 3-650. 函数 syscfg_sram_ecc_bit_get .....          | 437 |
| 表 3-651. TIMER 寄存器 .....                           | 437 |
| 表 3-652. TIMER 库函数 .....                           | 439 |
| 表 3-653. 结构体 timer_parameter_struct .....          | 441 |
| 表 3-654. 结构体 timer_break_parameter_struct .....    | 442 |

|   |     |
|---|-----|
| 表 3-655. 结构体 timer_oc_parameter_struct.....                 | 442 |
| 表 3-656. 结构体 timer_omc_parameter_struct .....               | 443 |
| 表 3-657. 结构体 timer_ic_parameter_struct.....                 | 443 |
| 表 3-658. 结构体 timer_break_ext_input_struct.....              | 443 |
| 表 3-659. 结构体 timer_free_complementary_parameter_struct..... | 443 |
| 表 3-660. 函数 timer_deinit.....                               | 444 |
| 表 3-661. 函数 timer_struct_para_init .....                    | 444 |
| 表 3-662. 函数 timer_init.....                                 | 445 |
| 表 3-663. 函数 timer_enable.....                               | 445 |
| 表 3-664. 函数 timer_disable.....                              | 446 |
| 表 3-665. 函数 timer_auto_reload_shadow_enable.....            | 447 |
| 表 3-666. 函数 timer_auto_reload_shadow_disable.....           | 447 |
| 表 3-667. 函数 timer_update_event_enable .....                 | 448 |
| 表 3-668. 函数 timer_update_event_disable .....                | 448 |
| 表 3-669. 函数 timer_counter_alignment.....                    | 449 |
| 表 3-670. 函数 timer_counter_up_direction .....                | 449 |
| 表 3-671. 函数 timer_counter_down_direction .....              | 450 |
| 表 3-672. 函数 timer_prescaler_config .....                    | 450 |
| 表 3-673. 函数 timer_repetition_value_config.....              | 451 |
| 表 3-674. 函数 timer_autoreload_value_config.....              | 452 |
| 表 3-675. 函数 timer_counter_value_config.....                 | 452 |
| 表 3-676. 函数 timer_counter_read .....                        | 453 |
| 表 3-677. 函数 timer_prescaler_read .....                      | 454 |
| 表 3-678. 函数 timer_single_pulse_mode_config.....             | 454 |
| 表 3-679. 函数 timer_update_source_config.....                 | 455 |
| 表 3-680. 函数 timer_channel_control_shadow_config.....        | 455 |
| 表 3-681. 函数 timer_channel_control_shadow_update_config..... | 456 |
| 表 3-682. 函数 timer_dma_enable .....                          | 457 |
| 表 3-683. 函数 timer_dma_disable .....                         | 458 |
| 表 3-684. 函数 timer_channel_dma_request_source_select.....    | 459 |
| 表 3-685. 函数 timer_dma_transfer_config .....                 | 460 |
| 表 3-686. 函数 timer_event_software_generate.....              | 462 |
| 表 3-687. 函数 timer_break_struct_para_init .....              | 463 |
| 表 3-688. 函数 timer_break_config.....                         | 464 |
| 表 3-689. 函数 timer_break_enable .....                        | 465 |
| 表 3-690. 函数 timer_break_disable .....                       | 465 |
| 表 3-691. 函数 timer_automatic_output_enable .....             | 466 |
| 表 3-692. 函数 timer_automatic_output_disable .....            | 466 |
| 表 3-693. 函数 timer_primary_output_config.....                | 467 |
| 表 3-694. 函数 timer_channel_output_struct_para_init .....     | 468 |
| 表 3-695. 函数 timer_channel_output_config .....               | 468 |
| 表 3-696. 函数 timer_channel_output_mode_config.....           | 469 |
| 表 3-697. 函数 timer_channel_output_pulse_value_config.....    | 470 |
| 表 3-698. 函数 timer_channel_output_shadow_config.....         | 471 |

|  |     |
|--|-----|
| 表 3-699. 函数 timer_channel_output_clear_config .....                          | 472 |
| 表 3-700. 函数 timer_channel_output_polarity_config .....                       | 473 |
| 表 3-701. 函数 timer_channel_complementary_output_polarity_config .....         | 474 |
| 表 3-702. 函数 timer_channel_output_state_config .....                          | 475 |
| 表 3-703. 函数 timer_channel_complementary_output_state_config .....            | 476 |
| 表 3-704. 函数 timer_channel_input_struct_para_init .....                       | 477 |
| 表 3-705. 函数 timer_input_capture_config .....                                 | 478 |
| 表 3-706. 函数 timer_channel_input_capture_prescaler_config .....               | 479 |
| 表 3-707. 函数 timer_channel_capture_value_register_read .....                  | 480 |
| 表 3-708. 函数 timer_input_pwm_capture_config .....                             | 480 |
| 表 3-709. 函数 timer_hall_mode_config .....                                     | 481 |
| 表 3-710. 函数 timer_multi_mode_channel_output_parameter_struct_init .....      | 482 |
| 表 3-711. 函数 timer_multi_mode_channel_output_config .....                     | 483 |
| 表 3-712. 函数 timer_multi_mode_channel_mode_config .....                       | 483 |
| 表 3-713. 函数 timer_input_trigger_source_select .....                          | 484 |
| 表 3-714. 函数 timer_master_output_trigger_source_select .....                  | 486 |
| 表 3-715. 函数 timer_slave_mode_select .....                                    | 487 |
| 表 3-716. 函数 timer_master_slave_mode_config .....                             | 488 |
| 表 3-717. 函数 timer_external_trigger_config .....                              | 488 |
| 表 3-718. 函数 timer_quadrature_decoder_mode_config .....                       | 489 |
| 表 3-719. 函数 timer_internal_clock_config .....                                | 490 |
| 表 3-720. 函数 timer_internal_trigger_as_external_clock_config .....            | 491 |
| 表 3-721. 函数 timer_external_trigger_as_external_clock_config .....            | 492 |
| 表 3-722. 函数 timer_external_clock_mode0_config .....                          | 493 |
| 表 3-723. 函数 timer_external_clock_mode1_config .....                          | 494 |
| 表 3-724. 函数 timer_external_clock_mode1_disable .....                         | 495 |
| 表 3-725. 函数 timer_channel_remap_config .....                                 | 495 |
| 表 3-726. 函数 timer_write_chxval_register_config .....                         | 496 |
| 表 3-727. 函数 timer_output_value_selection_config .....                        | 497 |
| 表 3-728. 函数 timer_output_match_pulse_select .....                            | 497 |
| 表 3-729. 函数 timer_channel_composite_pwm_mode_config .....                    | 498 |
| 表 3-730. 函数 timer_channel_composite_pwm_mode_output_pulse_value_config ..... | 499 |
| 表 3-731. 函数 timer_channel_additional_compare_value_config .....              | 500 |
| 表 3-732. 函数 timer_channel_additional_output_shadow_config .....              | 501 |
| 表 3-733. 函数 timer_break_external_input_struct_para_init .....                | 502 |
| 表 3-734. 函数 timer_break_external_input_config .....                          | 502 |
| 表 3-735. 函数 timer_break_external_input_enable .....                          | 503 |
| 表 3-736. 函数 timer_break_external_input_disable .....                         | 504 |
| 表 3-737. 函数 timer_break_external_input_polarity_config .....                 | 505 |
| 表 3-738. 函数 timer_channel_break_control_config .....                         | 506 |
| 表 3-739. 函数 timer_channel_dead_time_config .....                             | 507 |
| 表 3-740. 函数 timer_free_complementary_struct_para_init .....                  | 507 |
| 表 3-741. 函数 timer_channel_free_complementary_config .....                    | 508 |
| 表 3-742. 函数 timer_flag_get .....   | 509 |



|   |     |
|---|-----|
| 表 3-743. 函数 timer_flag_clear .....                        | 510 |
| 表 3-744. 函数 timer_interrupt_enable .....                  | 512 |
| 表 3-745. 函数 timer_interrupt_disable .....                 | 513 |
| 表 3-746. 函数 timer_interrupt_flag_get .....                | 514 |
| 表 3-747. 函数 timer_interrupt_flag_clear .....              | 515 |
| 表 3-748. TRIGSEL 寄存器.....                                 | 517 |
| 表 3-749. TRIGSEL 库函数.....                                 | 517 |
| 表 3-750. 枚举类型 trigel_source_enum .....                    | 518 |
| 表 3-751. 枚举类型 trigel_periph_enum .....                    | 519 |
| 表 3-752. 函数 trigel_init.....                              | 521 |
| 表 3-753. 函数 trigel_trigger_source_get .....               | 521 |
| 表 3-754. 函数 trigel_trigger_source_set.....                | 522 |
| 表 3-755. 函数 trigel_trigger_lock_get .....                 | 522 |
| 表 3-756. USART 寄存器.....                                   | 524 |
| 表 3-757. USART 库函数.....                                   | 524 |
| 表 3-758. 枚举类型 usart_flag_enum.....                        | 526 |
| 表 3-759. 枚举类型 usart_interrupt_flag_enum .....             | 527 |
| 表 3-760. 枚举类型 usart_interrupt_enum .....                  | 527 |
| 表 3-761. 枚举类型 usart_invert_enum .....                     | 528 |
| 表 3-762. 函数 usart_deinit.....                             | 528 |
| 表 3-763. 函数 usart_baudrate_set.....                       | 529 |
| 表 3-764. 函数 usart_parity_config .....                     | 529 |
| 表 3-765. 函数 usart_word_length_set .....                   | 530 |
| 表 3-766. 函数 usart_stop_bit_set.....                       | 530 |
| 表 3-767. 函数 usart_enable.....                             | 531 |
| 表 3-768. 函数 usart_disable.....                            | 532 |
| 表 3-769. 函数 usart_transmit_config .....                   | 532 |
| 表 3-770. 函数 usart_receive_config .....                    | 533 |
| 表 3-771. 函数 usart_data_first_config .....                 | 534 |
| 表 3-772. 函数 usart_invert_config .....                     | 534 |
| 表 3-773. 函数 usart_overrun_enable .....                    | 535 |
| 表 3-774. 函数 usart_overrun_disable .....                   | 535 |
| 表 3-775. 函数 usart_oversample_config .....                 | 536 |
| 表 3-776. 函数 usart_sample_bit_config .....                 | 536 |
| 表 3-777. 函数 usart_receiver_timeout_enable .....           | 537 |
| 表 3-778. 函数 usart_receiver_timeout_disable .....          | 537 |
| 表 3-779. 函数 usart_receiver_timeout_threshold_config ..... | 538 |
| 表 3-780. 函数 usart_data_transmit .....                     | 539 |
| 表 3-781. 函数 usart_data_receive .....                      | 539 |
| 表 3-782. 函数 usart_command_enable.....                     | 540 |
| 表 3-783. 函数 usart_address_config .....                    | 540 |
| 表 3-784. 函数 usart_address_detection_mode_config.....      | 541 |
| 表 3-785. 函数 usart_mute_mode_enable .....                  | 542 |
| 表 3-786. 函数 usart_mute_mode_disable .....                 | 542 |

|  |     |
|--|-----|
| 表 3-787. 函数 usart_mute_mode_wakeup_config.....           | 543 |
| 表 3-788. 函数 usart_lin_mode_enable.....                   | 543 |
| 表 3-789. 函数 usart_lin_mode_disable.....                  | 544 |
| 表 3-790. 函数 usart_lin_break_dection_length_config.....   | 544 |
| 表 3-791. 函数 usart_halfduplex_enable.....                 | 545 |
| 表 3-792. 函数 usart_halfduplex_disable.....                | 545 |
| 表 3-793. 函数 usart_clock_enable .....                     | 546 |
| 表 3-794. 函数 usart_clock_disable .....                    | 547 |
| 表 3-795. 函数 usart_synchronous_clock_config .....         | 547 |
| 表 3-796. 函数 usart_guard_time_config.....                 | 548 |
| 表 3-797. 函数 usart_smartcard_mode_enable.....             | 548 |
| 表 3-798. 函数 usart_smartcard_mode_disable.....            | 549 |
| 表 3-799. 函数 usart_smartcard_mode_nack_enable .....       | 549 |
| 表 3-800. 函数 usart_smartcard_mode_nack_disable .....      | 550 |
| 表 3-801. 函数 usart_smartcard_mode_early_nack_enable.....  | 551 |
| 表 3-802. 函数 usart_smartcard_mode_early_nack_disable..... | 551 |
| 表 3-803. 函数 usart_smartcard_autoretry_config .....       | 552 |
| 表 3-804. 函数 usart_block_length_config.....               | 552 |
| 表 3-805. 函数 usart_irda_mode_enable .....                 | 553 |
| 表 3-806. 函数 usart_irda_mode_disable .....                | 553 |
| 表 3-807. 函数 usart_prescaler_config .....                 | 554 |
| 表 3-808. 函数 usart_irda_lowpower_config.....              | 554 |
| 表 3-809. 函数 usart_hardware_flow_rts_config .....         | 555 |
| 表 3-810. 函数 usart_hardware_flow_cts_config.....          | 555 |
| 表 3-811. 函数 usart_hardware_flow_coherence_config.....    | 556 |
| 表 3-812. 函数 usart_rs485_driver_enable .....              | 557 |
| 表 3-813. 函数 usart_rs485_driver_disable .....             | 557 |
| 表 3-814. 函数 usart_driver_asserttime_config .....         | 558 |
| 表 3-815. 函数 usart_driver_deasserttime_config.....        | 558 |
| 表 3-816. 函数 usart_depolarity_config .....                | 559 |
| 表 3-817. 函数 usart_dma_receive_config .....               | 560 |
| 表 3-818. 函数 usart_dma_transmit_config.....               | 560 |
| 表 3-819. 函数 usart_reception_error_dma_disable .....      | 561 |
| 表 3-820. 函数 usart_reception_error_dma_enable .....       | 561 |
| 表 3-821. 函数 usart_wakeup_enable .....                    | 562 |
| 表 3-822. 函数 usart_wakeup_disable .....                   | 562 |
| 表 3-823. 函数 usart_wakeup_mode_config .....               | 563 |
| 表 3-824. 函数 usart_receive_fifo_enable.....               | 564 |
| 表 3-825. 函数 usart_receive_fifo_disable.....              | 564 |
| 表 3-826. 函数 usart_receive_fifo_counter_number .....      | 565 |
| 表 3-827. 函数 usart_flag_get .....                         | 565 |
| 表 3-828. 函数 usart_flag_clear .....                       | 566 |
| 表 3-829. 函数 usart_interrupt_enable.....                  | 567 |
| 表 3-830. 函数 usart_interrupt_disable.....                 | 567 |



|  |     |
|--|-----|
| 表 3-831. 函数 usart_interrupt_flag_get .....   | 568 |
| 表 3-832. 函数 usart_interrupt_flag_clear ..... | 569 |
| 表 3-833. WWDGT 寄存器.....                      | 570 |
| 表 3-834. WWDGT 库函数.....                      | 570 |
| 表 3-835. 函数 wwdgt_deinit .....               | 571 |
| 表 3-836. 函数 wwdgt_enable .....               | 571 |
| 表 3-837. 函数 wwdgt_counter_update.....        | 572 |
| 表 3-838. 函数 wwdgt_config .....               | 572 |
| 表 3-839. 函数 wwdgt_interrupt_enable .....     | 573 |
| 表 3-840. 函数 wwdgt_flag_get.....              | 574 |
| 表 3-841. 函数 wwdgt_flag_clear .....           | 574 |
| 表 4-1. 版本历史.....                             | 576 |

## 1. 介绍

本手册介绍了32位基于ARM微控制器GD32A50x固件库。

该固件库是一个固件函数包，它由程序、数据结构和宏组成，包括了GD32A50x所有外设的性能特征。该固件库还包括每一个外设的驱动描述和基于评估板的固件库使用例程。通过使用本固件库，用户无需深入掌握细节，也可以轻松应用每一个外设。使用本固件库可以大大减少用户的编程时间，从而降低开发成本。

每个外设驱动都由一组函数组成，这组函数覆盖了该外设所有功能。可以通过调用一组通用API(application programming interface应用编程界面)来实现对外设的驱动，这些API的结构、函数名称和参数名称都进行了标准化规范。

所有的驱动源代码都符合“MISRA-C:2004”标准（例程文件符合扩充ANSI-C标准），不会受到来自开发环境差异带来的影响。仅有启动文件取决于开发环境。

因为该固件库是通用的，并且包括了所有外设的功能，所以应用程序代码的大小和执行速度可能不是最优的。对大多数应用程序来说，用户可以直接使用之，对于那些在代码大小和执行速度方面有严格要求的应用程序，该固件库可以作为如何设置外设的一份参考资料，可以根据实际需求对其进行调整。

此份固件库使用手册的整体架构如下：

- 文档和固件库规则；
- 固件库概述；
- 外设固件库具体描述，外设固件库例程使用说明。

### 1.1. 文档和固件库规则

#### 1.1.1. 外设缩写

表 1-1. 外设缩写

| 外设缩写   | 说明         |
|--------|------------|
| ADC    | 模数转换器      |
| BKP    | 备份域寄存器     |
| CAN    | 控制器局域网络    |
| CMP    | 比较器        |
| CRC    | 循环冗余校验计算单元 |
| DBG    | 调试模块       |
| DAC    | 数模转换器      |
| DMA    | 直接存储器访问控制器 |
| DMAMUX | DMA请求多路复用器 |
| EXTI   | 外部中断事件控制器  |

| 外设缩写      | 说明            |
|-----------|---------------|
| FMC       | 闪存控制器         |
| FWDGT     | 独立看门狗         |
| GPIO/AFIO | 通用和备用输入/输出接口  |
| I2C       | 内部集成电路总线接口    |
| MFCOM     | 多功能通信接口       |
| MISC      | 嵌套中断向量列表控制器   |
| PMU       | 电源管理单元        |
| RCU       | 复位和时钟单元       |
| RTC       | 实时时钟          |
| SPI/I2S   | 串行外设接口/片上音频接口 |
| SYSCFG    | 系统配置          |
| TIMER     | 定时器           |
| TRIGSEL   | 触发选择控制器       |
| USART     | 通用同步异步收发器     |
| WWDGT     | 窗口看门狗         |

### 1.1.2. 命名规则

固件库遵从以下命名规则：

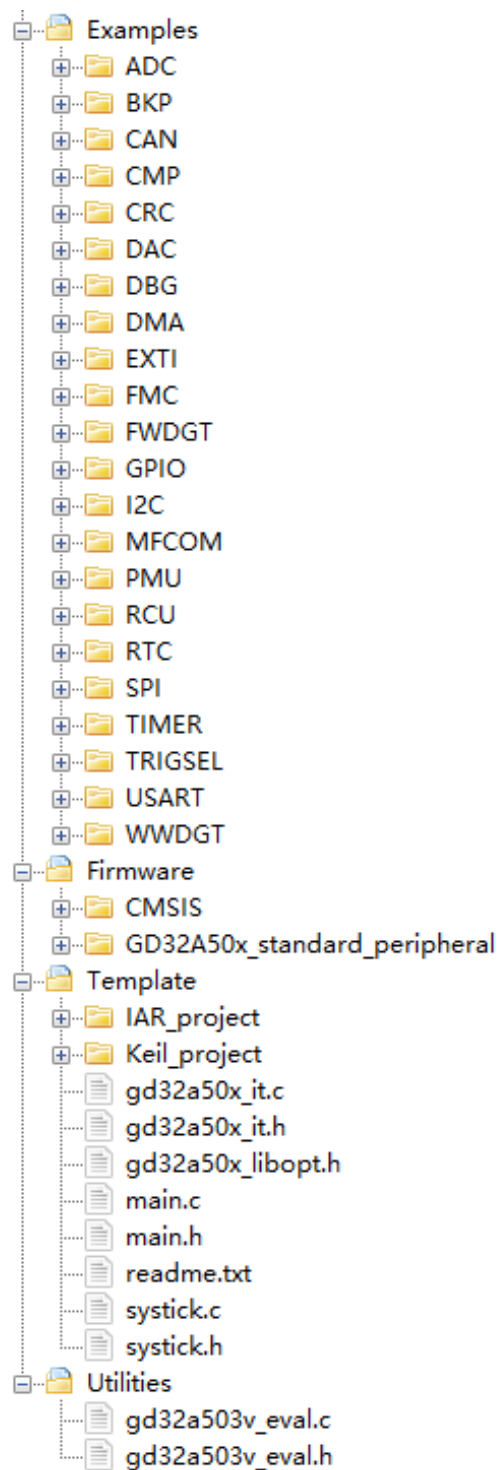
- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；
- 源文件和头文件命名都以“gd32a50x\_”作为开头，例如：gd32a50x\_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

## 2. 固件库概述

### 2.1. 文件组织结构

GD32A50x\_Firmware\_Library, 文件组织结构见下图:

图 2-1. GD32A50x 固件库文件组织结构



### 2.1.1. Examples 文件夹

文件夹**Examples**，对应每一个GD32外设均包含一个子文件夹。每个子文件夹包含了关于本外设的一个或多个例程，来示范如何使用对应外设。每个例程子文件夹包含如下文件：

- **readme.txt**: 关于本例程的简单描述和使用说明；
- **gd32a50x\_libopt.h**: 该头文件可以设置例程所使用到的外设，由不同的“**DEFINE**”语句组成（默认情况下，所有外设均打开）；
- **gd32a50x\_it.c**: 该源文件包含了所有的中断处理程序（如果未使用到中断，则所有的函数体都为空）；
- **gd32a50x\_it.h**: 该头文件包含了所有的中断处理程序的原形；
- **systick.c**: 该源文件包含了使用**systick**的精准延时程序；
- **systick.h**: 该头文件包含了使用**systick**的精准延时程序的原形；
- **main.c**: 例程代码注：所有的例程的使用，都不受不同软件开发环境的影响。

### 2.1.2. Firmware 文件夹

**Firmware**文件夹包含组成固件库核心的所有子文件夹和文件：

- **CMSIS**子文件夹包含有**Cortex M33**内核的支持文件、基于**Cortex M33**内核处理器的启动代码和库引导文件以及基于**GD32A50x**的全局头文件和系统配置文件；
- **GD32A50x\_standard\_peripheral**子文件夹：
  - **Include**子文件夹包含了固件函数库所需的头文件，用户无需修改该文件夹；
  - **Source**子文件夹包含了固件函数库所需的源文件，用户无需修改该文件夹；

注：所有代码都按照**MISRA-C:2004**标准书写，都不受不同软件开发环境的影响。

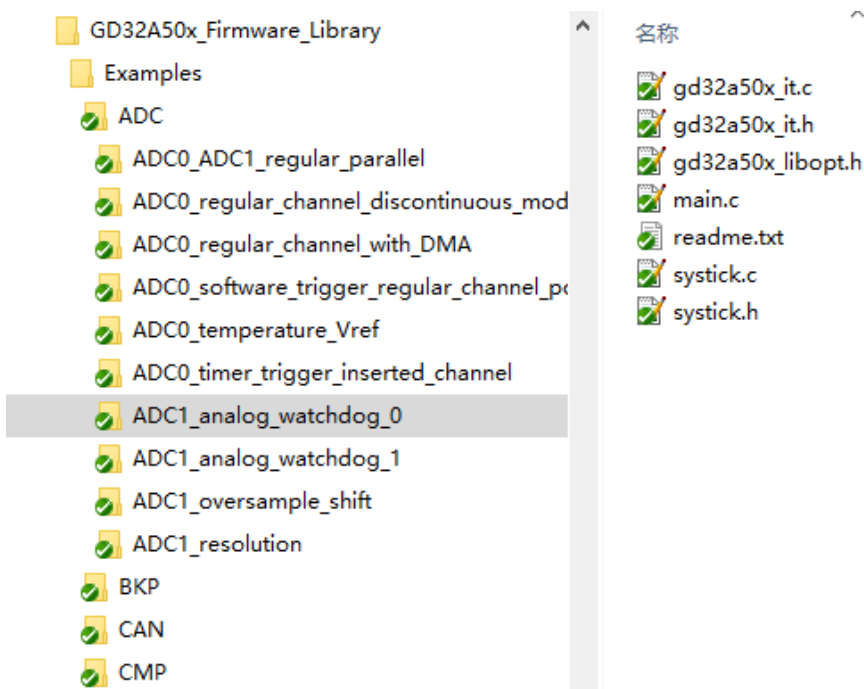
### 2.1.3. Template 文件夹

**Template**文件夹包含一个关于使用**LED**、**USART**打印、按键控制的简单例程，（**IAR\_project**用于**IAR**编译环境，**Keil\_project**用于**Keil5**编译环境）。用户可以使用该工程模板进行固件库例程的移植编译，具体使用方法见下：

#### 选择文件

打开“**Examples**”文件夹，选择需要测试的模块，如**ADC**，打开“**ADC**”文件夹，选择**ADC**的一个例程，如“**Analog\_watchdog**”，如下图所示：

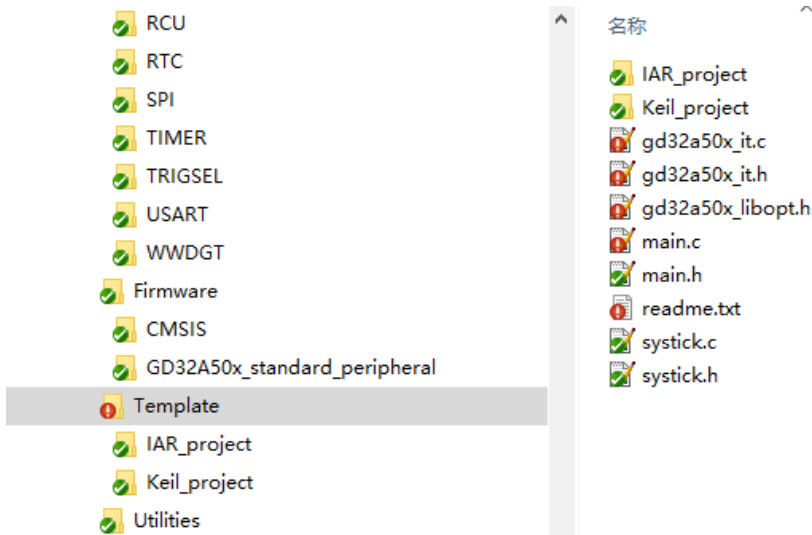
图 2-2. 选择外设例程文件



### 拷贝文件

打开“Template”文件夹，将“IAR\_project”和“Keil\_project”两个文件夹保留，其他文件都删除，然后将“Analog\_watchdog”文件夹中的所有文件拷到“Template”文件夹子目录下，如下图所示：

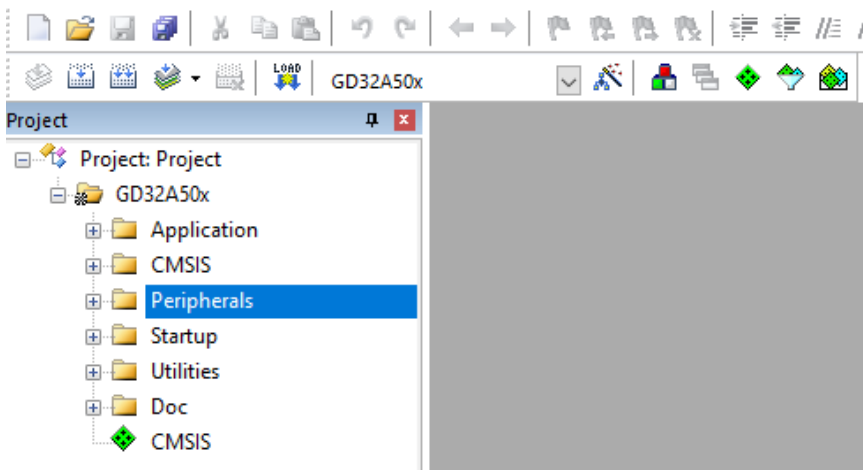
图 2-3. 拷贝外设例程文件



### 打开工程

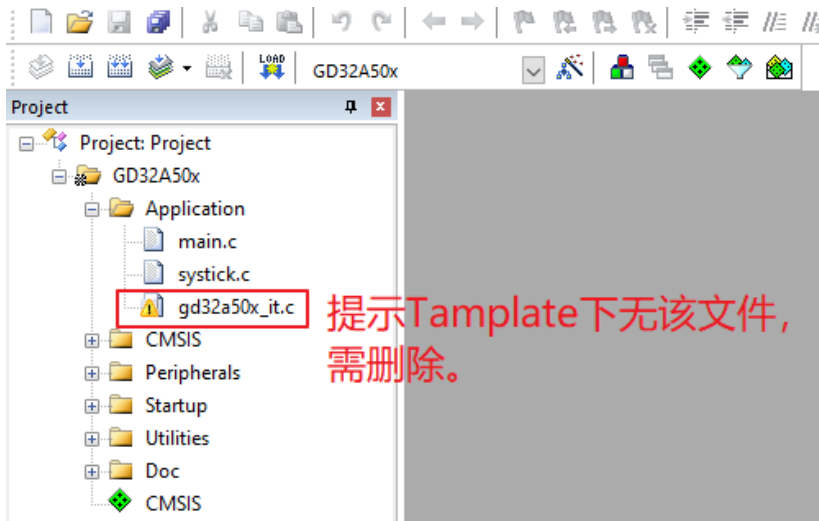
GD提供Keil和IAR两种版本的工程，根据客户所安装的软件，打开不同的project，如“Keil\_project”，打开Template\Keil\_project\Project.uvprojx，如下图所示：

图 2-4. 打开工程文件



由于不同的模块、不同的功能，会使用到不同的文件，需要根据客户选择拷贝的文件，对工程里的文件进行增加或删除，如下图所示：

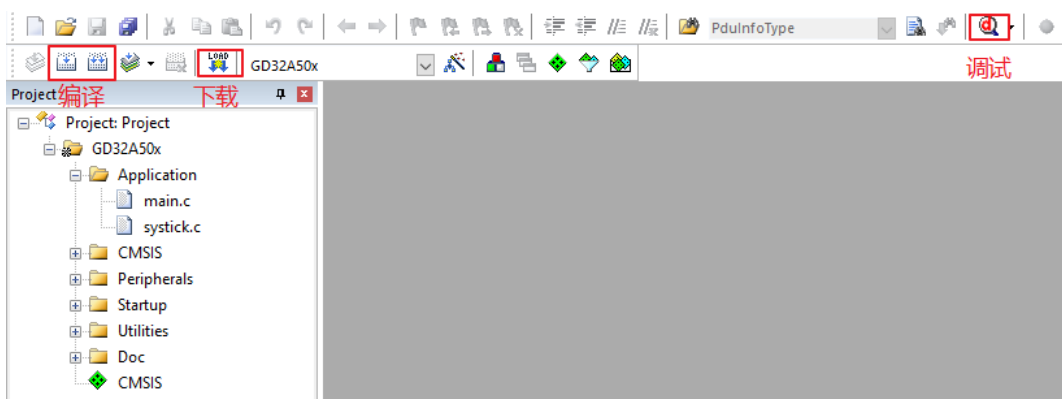
图 2-5. 配置工程文件



### 编译调试下载

首先编译整个工程，如果无错误，按照readme中的介绍，选择正确的跳线及连线，然后再将程序下载到目标板上，则会有如readme中描述的现象。IDE的具体使用，请参考相应的软件使用说明。如客户使用的是Keil，可见下图所示：

图 2-6. 编译调试下载



#### 2.1.4. Utilities 文件夹

Utilities文件夹包含运行固件库例程评估板的文件：

- gd32a503v\_eval.h文件是运行固件库例程所需关于评估板的头文件；
- gd32a503v\_eval.c文件是运行固件库例程所需关于评估板的源文件。

注：所有代码都按照MISRA-C:2004标准书写，都不受不同软件开发环境的影响。

## 2.2. 固件库文件描述

下表列举和描述了固件库使用的主要文件。

表 2-1. 固件函数库文件描述

| 文件名                | 描述   |
|--------------------|--|
| gd32a50xx_libopt.h | 包含了所有外设的头文件的头文件。它是唯一一个用户需要包括在自己应用中的文件，起到应用和库之间界面的作用。                                     |
| main.c             | 主函数体示例。  |
| gd32a50xx_it.h     | 头文件，包含所有中断处理函数原形。  |
| gd32a50xx_it.c     | 外设中断函数文件。用户可以加入自己的中断程序代码。对于指向同一个中断向量的多个不同中断请求，可以利用函数通过判断外设的中断标志位来确定准确的中断源。固件库提供了这些函数的名称。 |
| gd32a50xx_xxx.h    | 外设xxx的头文件。包含外设xxx函数的定义，以及这些函数使用的变量。  |
| gd32a50xx_xxx.c    | 由C语言编写的外设xxx的驱动源程序文件。  |
| systick.h          | systick.c的头文件。包含systick配置函数的定义，以及外部用延时函数的定义。   |
| systick.c          | systick配置与延时函数源文件。   |
| readme.txt         | 固件库例程使用及配置说明文档。  |



### 3. 外设固件库

#### 3.1. 外设固件库概述

外设固件库函数的描述格式如下表：

表 3-1. 外设固件库函数描述格式

|           |              |
|-----------|--------------|
| 函数名称      | 外设函数的名称      |
| 函数原型      | 原型声明         |
| 功能描述      | 简要解释函数是如何执行的 |
| 先决条件      | 调用函数前应满足的要求  |
| 被调用函数     | 其他被该函数调用的库函数 |
| 输入参数{in}  |              |
| XXX       | 输入参数描述       |
| Xx        | 输入参数可选宏描述    |
| 输出参数{out} |              |
| XXX       | 输出参数描述       |
| 返回值       |              |
| XXX       | 函数的返回值       |

#### 3.2. ADC

12位ADC是一种采用逐次逼近方式的模拟数字转换器。章节[3.2.1](#)描述了ADC的寄存器列表，章节[3.2.2](#)对ADC库函数进行说明。

##### 3.2.1. 外设寄存器描述

ADC寄存器列表如下表所示：

表 3-2. ADC 寄存器

| 寄存器名称                 | 寄存器描述        |
|-----------------------|--------------|
| ADC_STAT              | 状态寄存器        |
| ADC_CTL0              | 控制寄存器0       |
| ADC_CTL1              | 控制寄存器1       |
| ADC_SAMPT0            | 采样时间寄存器0     |
| ADC_SAMPT1            | 采样时间寄存器1     |
| ADC_IOFFx<br>(x=0..3) | 注入通道数据偏移寄存器x |
| ADC_WDHT0             | 看门狗0高阈值寄存器   |
| ADC_WDLT0             | 看门狗0低阈值寄存器   |
| ADC_RSQ0              | 规则序列寄存器0     |

| 寄存器名称                  | 寄存器描述       |
|------------------------|-------------|
| ADC_RSQ1               | 规则序列寄存器1    |
| ADC_RSQ2               | 规则序列寄存器2    |
| ADC_ISQ                | 注入序列寄存器     |
| ADC_IDATAx<br>(x=0..3) | 注入数据寄存器x    |
| ADC_RDATA              | 规则数据寄存器     |
| ADC_OVSAMPCTL          | 过采样控制寄存器    |
| ADC_WD1SR              | 看门狗1通道选择寄存器 |
| ADC_WDT1               | 看门狗1阈值寄存器   |

### 3.2.2. 外设库函数说明

ADC库函数列表如下表所示：

**表 3-3. ADC 库函数**

| 库函数名称                               | 库函数描述                      |
|-------------------------------------|----------------------------|
| adc_deinit                          | 复位ADC外设                    |
| adc_enable                          | 使能ADC外设                    |
| adc_disable                         | 禁能ADC外设                    |
| adc_calibration_enable              | ADC校准复位                    |
| adc_dma_mode_enable                 | ADC DMA请求使能                |
| adc_dma_mode_disable                | ADC DMA请求禁能                |
| adc_tempsensor_enable               | 温度传感器通道使能                  |
| adc_tempsensor_disable              | 温度传感器通道禁能                  |
| adc_vrefint_enable                  | vrefint通道使能                |
| adc_vrefint_disable                 | vrefint通道禁能                |
| adc_discontinuous_mode_config       | 配置ADC间断模式                  |
| adc_mode_config                     | 配置ADC同步模式                  |
| adc_special_function_config         | 使能或禁能ADC特殊功能               |
| adc_data_alignment_config           | 配置ADC数据对齐方式                |
| adc_channel_length_config           | 配置规则通道组或注入通道组的长度           |
| adc_regular_channel_config          | 配置ADC规则通道组                 |
| adc_inserted_channel_config         | 配置ADC注入通道组                 |
| adc_inserted_channel_offset_config  | 配置ADC注入通道组数据偏移值            |
| adc_external_trigger_config         | 配置ADC外部触发                  |
| adc_external_trigger_source_config  | 配置ADC外部触发源                 |
| adc_software_trigger_enable         | ADC软件触发使能                  |
| adc_regular_data_read               | 读ADC规则组数据寄存器               |
| adc_inserted_data_read              | 读ADC注入组数据寄存器               |
| adc_sync_mode_convert_value_read    | 在同步模式下，读ADC0和ADC1最近的一次转换结果 |
| adc_watchdog0_single_channel_enable | 配置ADC模拟看门狗0单通道有效           |

| 库函数名称                              | 库函数描述             |
|------------------------------------|-------------------|
| adc_watchdog0_group_channel_enable | 配置ADC模拟看门狗0在通道组有效 |
| adc_watchdog0_disable              | ADC模拟看门狗0禁能       |
| adc_watchdog1_channel_config       | 配置ADC模拟看门狗1通道     |
| adc_watchdog1_disable              | ADC模拟看门狗1禁能       |
| adc_watchdog0_threshold_config     | 配置ADC模拟看门狗0阈值     |
| adc_watchdog1_threshold_config     | 配置ADC模拟看门狗1阈值     |
| adc_resolution_config              | 配置ADC分辨率          |
| adc_oversample_mode_config         | 配置ADC过采样模式        |
| adc_oversample_mode_enable         | 使能ADC过采样          |
| adc_oversample_mode_disable        | 禁能ADC过采样          |
| adc_flag_get                       | 获取ADC标志位          |
| adc_flag_clear                     | 清除ADC标志位          |
| adc_interrupt_flag_get             | 获取ADC中断标志位        |
| adc_interrupt_flag_clear           | 清除ADC中断标志位        |
| adc_interrupt_enable               | ADC中断使能           |
| adc_interrupt_disable              | ADC中断禁能           |

## 函数 adc\_deinit

函数adc\_deinit描述见下表：

表 3-4. 函数 adc\_deinit

|             |  |
|-------------|--|
| 函数名称        | adc_deinit   |
| 函数原形        | void adc_deinit(uint32_t adc_periph);              |
| 功能描述        | 复位ADC外设  |
| 先决条件        | -  |
| 被调用函数       | rcu_periph_reset_enable / rcu_periph_reset_disable |
| 输入参数{in}    |  |
| adc_periph  | ADC外设  |
| ADCx(x=0,1) | ADC外设选择  |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如：

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

## 函数 adc\_enable

函数adc\_enable描述见下表：

表 3-5. 函数 adc\_enable

|             |                                       |
|-------------|---------------------------------------|
| 函数名称        | adc_enable                            |
| 函数原形        | void adc_enable(uint32_t adc_periph); |
| 功能描述        | 使能ADC外设                               |
| 先决条件        | -                                     |
| 被调用函数       | -                                     |
| 输入参数{in}    |                                       |
| adc_periph  | ADC外设                                 |
| ADCx(x=0,1) | ADC外设选择                               |
| 输出参数{out}   |                                       |
| -           | -                                     |
| 返回值         |                                       |
| -           | -                                     |

例如：

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

## 函数 adc\_disable

函数adc\_disable描述见下表：

表 3-6. 函数 adc\_disable

|             |  |
|-------------|--|
| 函数名称        | adc_disable                            |
| 函数原形        | void adc_disable(uint32_t adc_periph); |
| 功能描述        | 禁能ADC外设                                |
| 先决条件        | -                                      |
| 被调用函数       | -                                      |
| 输入参数{in}    |  |
| adc_periph  | ADC外设                                  |
| ADCx(x=0,1) | ADC外设选择                                |
| 输出参数{out}   |  |
| -           | -                                      |
| 返回值         |  |
| -           | -                                      |

例如：

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

## 函数 adc\_calibration\_enable

函数adc\_calibration\_enable描述见下表:

表 3-7. 函数 adc\_calibration\_enable

|             |   |
|-------------|---|
| 函数名称        | adc_calibration_enable                            |
| 函数原形        | void adc_calibration_enable(uint32_t adc_periph); |
| 功能描述        | ADC校准复位   |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| adc_periph  | ADC外设   |
| ADCx(x=0,1) | ADC外设选择   |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如:

```
/* ADC0 calibration and reset calibration */
```

```
adc_calibration_enable(ADC0);
```

## 函数 adc\_dma\_mode\_enable

函数adc\_dma\_mode\_enable描述见下表:

表 3-8. 函数 adc\_dma\_mode\_enable

|             |  |
|-------------|--|
| 函数名称        | adc_dma_mode_enable                            |
| 函数原形        | void adc_dma_mode_enable(uint32_t adc_periph); |
| 功能描述        | ADC DMA请求使能                                    |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| adc_periph  | ADC外设  |
| ADCx(x=0,1) | ADC外设选择  |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如:

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

```
adc_dma_mode_enable(ADC0);
```

## 函数 adc\_dma\_mode\_disable

函数adc\_dma\_mode\_disable描述见下表:

表 3-9. 函数 adc\_dma\_mode\_disable

|             |   |
|-------------|---|
| 函数名称        | adc_dma_mode_disable                            |
| 函数原形        | void adc_dma_mode_disable(uint32_t adc_periph); |
| 功能描述        | ADC DMA请求禁能                                     |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| adc_periph  | ADC外设   |
| ADCx(x=0,1) | ADC外设选择   |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如:

```
/* disable ADC0 DMA request */
adc_dma_mode_disable(ADC0);
```

## 函数 adc\_tempsensor\_enable

函数adc\_tempsensor\_enable描述见下表:

表 3-10. 函数 adc\_tempsensor\_enable

|           |                                   |
|-----------|-----------------------------------|
| 函数名称      | adc_tempsensor_enable             |
| 函数原形      | void adc_tempsensor_enable(void); |
| 功能描述      | 温度传感器通道使能                         |
| 先决条件      | -                                 |
| 被调用函数     | -                                 |
| 输入参数{in}  |                                   |
| -         | -                                 |
| 输出参数{out} |                                   |
| -         | -                                 |
| 返回值       |                                   |
| -         | -                                 |

例如:

```
/* enable the temperature sensor channel */
adc_tempsensor_enable();
```

## 函数 adc\_tempsensor\_disable

函数adc\_tempsensor\_disable描述见下表:

表 3-11. 函数 adc\_tempsensor\_disable

|           |                                    |
|-----------|------------------------------------|
| 函数名称      | adc_tempsensor_disable             |
| 函数原形      | void adc_tempsensor_disable(void); |
| 功能描述      | 温度传感器通道禁能                          |
| 先决条件      | -                                  |
| 被调用函数     | -                                  |
| 输入参数{in}  |                                    |
| -         | -                                  |
| 输出参数{out} |                                    |
| -         | -                                  |
| 返回值       |                                    |
| -         | -                                  |

例如:

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

```
adc_tempsensor_disable();
```

## 函数 adc\_vrefint\_enable

函数adc\_vrefint\_enable描述见下表:

表 3-12. 函数 adc\_vrefint\_enable

|           |                                |
|-----------|--------------------------------|
| 函数名称      | adc_vrefint_enable             |
| 函数原形      | void adc_vrefint_enable(void); |
| 功能描述      | vrefint通道使能                    |
| 先决条件      | -                              |
| 被调用函数     | -                              |
| 输入参数{in}  |                                |
| -         | -                              |
| 输出参数{out} |                                |
| -         | -                              |
| 返回值       |                                |
| -         | -                              |

例如:

```
/* enable the vrefint channel */
```

```
adc_vrefint_enable();
```

## 函数 `adc_vrefint_disable`

函数`adc_vrefint_disable`描述见下表：

**表 3-13. 函数 `adc_vrefint_disable`**

|           |  |
|-----------|--|
| 函数名称      | <code>adc_vrefint_disable</code>             |
| 函数原形      | <code>void adc_vrefint_disable(void);</code> |
| 功能描述      | vrefint通道禁能                                  |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如：

```
/* disable the vrefint channel */
```

```
adc_vrefint_disable();
```

## 函数 `adc_discontinuous_mode_config`

函数`adc_discontinuous_mode_config`描述见下表：

**表 3-14. 函数 `adc_discontinuous_mode_config`**

|   |  |
|---|--|
| 函数名称                                    | <code>adc_discontinuous_mode_config</code>   |
| 函数原形                                    | <code>void adc_discontinuous_mode_config(uint32_t adc_periph, uint8_t adc_channel_group, uint8_t length);</code> |
| 功能描述                                    | 配置ADC间断模式  |
| 先决条件                                    | -  |
| 被调用函数                                   | -  |
| 输入参数{in}                                |  |
| <b>adc_periph</b>                       | ADC外设  |
| <code>ADCx(x=0,1)</code>                | ADC外设选择  |
| 输入参数{in}                                |  |
| <b>adc_channel_group</b>                | 通道组选择  |
| <code>ADC_REGULAR_CHANNEL</code>        | 规则通道组  |
| <code>ADC_INSERTED_CHANNEL</code>       | 注入通道组  |
| <code>ADC_CHANNEL_DISCON_DISABLE</code> | 规则通道组和注入通道组间断模式禁能  |



| 输入参数{in}      |                                    |
|---------------|------------------------------------|
| <b>length</b> | 中断模式下的转换数目，规则通道组取值为1..8，注入通道组取值无意义 |
| 输出参数{out}     |                                    |
| -             | -                                  |
| 返回值           |                                    |
| -             | -                                  |

例如：

```
/* configure ADC0 discontinuous mode */
```

```
adc_discontinuous_mode_config(ADC0, ADC_REGULAR_CHANNEL, 6);
```

### 函数 adc\_mode\_config

函数adc\_mode\_config描述见下表：

**表 3-15. 函数 adc\_mode\_config**

| <b>函数名称</b>                                       | adc_mode_config                      |
|---|--------------------------------------|
| <b>函数原形</b>                                       | void adc_mode_config(uint32_t mode); |
| <b>功能描述</b>                                       | 配置ADC同步模式                            |
| <b>先决条件</b>                                       | -                                    |
| <b>被调用函数</b>                                      | -                                    |
| 输入参数{in}  |                                      |
| <b>mode</b>                                       | ADC运行模式                              |
| ADC_MODE_FREE                                     | 所有ADC运行于独立模式                         |
| ADC_DAUL_REGULAR_PARALLEL_INSERTED_PARALLEL       | ADC0和ADC1运行在规则并行+注入并行组合模式            |
| ADC_DAUL_REGULAR_PARALLEL_INSERTED_ROTATION       | ADC0和ADC1运行在规则并行+交替触发组合模式            |
| ADC_DAUL_INSERTED_PARALLEL_REGULAR_FOLLOW_UP_FAST | ADC0和ADC1运行在注入并行+快速交叉组合模式            |
| ADC_DAUL_INSERTED_PARALLEL_REGULAR_FOLLOW_UP_SLOW | ADC0和ADC1运行在注入并行+慢速交叉组合模式            |
| ADC_DAUL_INSERTED_PARALLEL                        | ADC0和ADC1运行在注入并行模式                   |
| ADC_DAUL_REGULAR                                  | ADC0和ADC1运行在规则并行模式                   |

|   |                    |
|---|--------------------|
| LAL_PARALLEL                            |                    |
| ADC_DAU_L_REGU<br>LAL_FOLLOWUP_F<br>AST | ADC0和ADC1运行在快速交叉模式 |
| ADC_DAU_L_REGU<br>LAL_FOLLOWUP_S<br>LOW | ADC0和ADC1运行在慢速交叉模式 |
| ADC_DAU_L_INSERTED_TRIGGER_ROTATION     | ADC0和ADC1运行在交替触发模式 |
| 输出参数{out}                               |                    |
| -                                       | -                  |
| 返回值                                     |                    |
| -                                       | -                  |

例如:

```
/* configure the ADC sync mode */
```

```
adc_mode_config(ADC_MODE_FREE);
```

### 函数 adc\_special\_function\_config

函数adc\_special\_function\_config描述见下表:

表 3-16. 函数 adc\_special\_function\_config

|                           |   |
|---------------------------|---|
| 函数名称                      | adc_special_function_config   |
| 函数原形                      | void adc_special_function_config(uint32_t adc_periph, uint32_t function, ControlStatus newvalue); |
| 功能描述                      | 使能或禁能ADC特殊功能  |
| 先决条件                      | -   |
| 被调用函数                     | -   |
| 输入参数{in}                  |   |
| adc_periph                | ADC外设   |
| ADCx(x=0,1)               | ADC外设选择   |
| 输入参数{in}                  |   |
| function                  | 功能配置  |
| ADC_SCAN_MODE             | 扫描模式选择  |
| ADC_INSERTED_CHANNEL_AUTO | 注入组自动转换   |
| ADC_CONTINUOUS_MODE       | 连续模式选择  |
| 输入参数{in}                  |   |
| newvalue                  | 功能使能禁能  |
| ENABLE                    | 使能  |

|                |    |
|----------------|----|
| <i>DISABLE</i> | 禁能 |
| 输出参数{out}      |    |
| -              | -  |
| 返回值            |    |
| -              | -  |

例如：

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

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

### 函数 **adc\_data\_alignment\_config**

函数adc\_alignment\_config描述见下表：

**表 3-17. 函数 **adc\_data\_alignment\_config****

|                     |   |
|---------------------|---|
| 函数名称                | adc_data_alignment_config   |
| 函数原形                | void adc_data_alignment_config(uint32_t adc_periph, uint32_t data_alignment); |
| 功能描述                | 配置ADC数据对齐方式   |
| 先决条件                | -   |
| 被调用函数               | -   |
| 输入参数{in}            |   |
| adc_periph          | ADC外设   |
| ADCx(x=0,1)         | ADC外设选择   |
| 输入参数{in}            |   |
| data_alignment      | 数据对齐方式选择  |
| ADC_DATAALIGN_RIGHT | 右对齐   |
| ADC_DATAALIGN_LEFT  | 左对齐   |
| 输出参数{out}           |   |
| -                   | -   |
| 返回值                 |   |
| -                   | -   |

例如：

```
/* configure ADC0 data alignment */
```

```
adc_data_alignment_config(ADC0, ADC_DATAALIGN_RIGHT);
```

### 函数 **adc\_channel\_length\_config**

函数adc\_channel\_length\_config描述见下表：

表 3-18. 函数 `adc_channel_length_config`

|                                   |   |
|-----------------------------------|---|
| 函数名称                              | <code>adc_channel_length_config</code>  |
| 函数原形                              | <code>void adc_channel_length_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t length);</code> |
| 功能描述                              | 配置规则通道组或注入通道组的长度  |
| 先决条件                              | -   |
| 被调用函数                             | -   |
| 输入参数{in}                          |   |
| <code>adc_periph</code>           | ADC外设   |
| <code>ADCx(x=0,1)</code>          | ADC外设选择   |
| 输入参数{in}                          |   |
| <code>adc_channel_group</code>    | 通道组选择   |
| <code>ADC_REGULAR_CHANNEL</code>  | 规则通道组   |
| <code>ADC_INSERTED_CHANNEL</code> | 注入通道组   |
| 输入参数{in}                          |   |
| <code>length</code>               | 通道长度，规则通道组为1-16，注入通道组为1-4   |
| 输出参数{out}                         |   |
| -                                 | -   |
| 返回值                               |   |
| -                                 | -   |

例如：

```
/* configure the length of ADC0 regular channel */
```

```
adc_channel_length_config(ADC0, ADC_REGULAR_CHANNEL, 4);
```

### 函数 `adc_regular_channel_config`

函数`adc_regular_channel_config`描述见下表：

表 3-19. 函数 `adc_regular_channel_config`

|                          |   |
|--------------------------|---|
| 函数名称                     | <code>adc_regular_channel_config</code>   |
| 函数原形                     | <code>void adc_regular_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);</code> |
| 功能描述                     | 配置ADC规则通道组  |
| 先决条件                     | -   |
| 被调用函数                    | -   |
| 输入参数{in}                 |   |
| <code>adc_periph</code>  | ADC外设   |
| <code>ADCx(x=0,1)</code> | ADC外设选择   |
| 输入参数{in}                 |   |

|                              |                                       |
|------------------------------|---------------------------------------|
| <b>rank</b>                  | 规则组通道序列，取值范围为0~15                     |
| <b>输入参数{in}</b>              |                                       |
| <b>adc_channel</b>           | ADC通道选择                               |
| ADC_CHANNEL_x<br>(x=0..17)   | ADC 通道x (x=0..17)(只有ADC0，可取值x=16和17 ) |
| <b>输入参数{in}</b>              |                                       |
| <b>sample_time</b>           | 采样时间                                  |
| ADC_SAMPLETIME<br>_2POINT5   | 2.5周期                                 |
| ADC_SAMPLETIME<br>_14POINT5  | 14.5周期                                |
| ADC_SAMPLETIME<br>_27POINT5  | 27.5周期                                |
| ADC_SAMPLETIME<br>_55POINT5  | 55.5周期                                |
| ADC_SAMPLETIME<br>_83POINT5  | 83.5周期                                |
| ADC_SAMPLETIME<br>_111POINT5 | 111.5周期                               |
| ADC_SAMPLETIME<br>_143POINT5 | 143.5周期                               |
| ADC_SAMPLETIME<br>_479POINT5 | 479.5周期                               |
| <b>输出参数{out}</b>             |                                       |
| -                            | -                                     |
| <b>返回值</b>                   |                                       |
| -                            | -                                     |

例如：

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

```
adc_regular_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_2POINT5);
```

### 函数 adc\_inserted\_channel\_config

函数adc\_inserted\_channel\_config描述见下表：

**表 3-20. 函数 adc\_inserted\_channel\_config**

|              |   |
|--------------|---|
| <b>函数名称</b>  | adc_inserted_channel_config   |
| <b>函数原形</b>  | void adc_inserted_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time); |
| <b>功能描述</b>  | 配置ADC注入通道组  |
| <b>先决条件</b>  | -   |
| <b>被调用函数</b> | -   |

| 输入参数{in}                          |                                      |
|-----------------------------------|--------------------------------------|
| <b>adc_periph</b>                 | ADC外设                                |
| <i>ADCx(x=0,1)</i>                | ADC外设选择                              |
| 输入参数{in}                          |                                      |
| <b>rank</b>                       | 注入组通道序列，取值范围为0~3                     |
| 输入参数{in}                          |                                      |
| <b>adc_channel</b>                | ADC通道选择                              |
| <i>ADC_CHANNEL_x</i><br>(x=0..17) | ADC 通道x (x=0..17)(只有ADC0，可取值x=16和17) |
| 输入参数{in}                          |                                      |
| <b>sample_time</b>                | 采样时间                                 |
| <i>ADC_SAMPLETIME_2POINT5</i>     | 2.5周期                                |
| <i>ADC_SAMPLETIME_14POINT5</i>    | 14.5周期                               |
| <i>ADC_SAMPLETIME_27POINT5</i>    | 27.5周期                               |
| <i>ADC_SAMPLETIME_55POINT5</i>    | 55.5周期                               |
| <i>ADC_SAMPLETIME_83POINT5</i>    | 83.5周期                               |
| <i>ADC_SAMPLETIME_111POINT5</i>   | 111.5周期                              |
| <i>ADC_SAMPLETIME_143POINT5</i>   | 143.5周期                              |
| <i>ADC_SAMPLETIME_479POINT5</i>   | 479.5周期                              |
| 输出参数{out}                         |                                      |
| -                                 | -                                    |
| 返回值                               |                                      |
| -                                 | -                                    |

例如：

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

```
adc_inserted_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_2POINT5);
```

### 函数 **adc\_inserted\_channel\_offset\_config**

函数adc\_inserted\_channel\_offset\_config描述见下表：

**表 3-21. 函数 **adc\_inserted\_channel\_offset\_config****

|      |  |
|------|--|
| 函数名称 | adc_inserted_channel_offset_config                                   |
| 函数原形 | void adc_inserted_channel_offset_config(uint32_t adc_periph, uint8_t |

|                                |                                     |
|--------------------------------|-------------------------------------|
|                                | inserted_channel, uint16_t offset); |
| 功能描述                           | 配置ADC注入通道组数据偏移值                     |
| 先决条件                           | -                                   |
| 被调用函数                          | -                                   |
| 输入参数{in}                       |                                     |
| adc_periph                     | ADC外设                               |
| ADCx(x=0,1)                    | ADC外设选择                             |
| 输入参数{in}                       |                                     |
| inserted_channel               | 注入通道选择                              |
| ADC_INSERTED_CHANNEL_x(x=0..3) | 注入通道, x=0,1,2,3                     |
| 输入参数{in}                       |                                     |
| offset                         | 数据偏移值, 取值范围为0~4095                  |
| 输出参数{out}                      |                                     |
| -                              | -                                   |
| 返回值                            |                                     |
| -                              | -                                   |

例如:

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

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

### 函数 adc\_external\_trigger\_config

函数adc\_external\_trigger\_config描述见下表:

表 3-22. 函数 adc\_external\_trigger\_config

|                      |   |
|----------------------|---|
| 函数名称                 | adc_external_trigger_config   |
| 函数原形                 | void adc_external_trigger_config(uint32_t adc_periph, uint8_t adc_channel_group, ControlStatus newvalue); |
| 功能描述                 | 配置ADC外部触发   |
| 先决条件                 | -   |
| 被调用函数                | -   |
| 输入参数{in}             |   |
| adc_periph           | ADC外设   |
| ADCx(x=0,1)          | ADC外设选择   |
| 输入参数{in}             |   |
| adc_channel_group    | 通道组选择   |
| ADC_REGULAR_CHANNEL  | 规则通道组   |
| ADC_INSERTED_CHANNEL | 注入通道组   |

| 输入参数{in}        |         |
|-----------------|---------|
| <b>newvalue</b> | 通道使能/禁能 |
| <i>ENABLE</i>   | 使能      |
| <i>DISABLE</i>  | 禁能      |
| 输出参数{out}       |         |
| -               | -       |
| 返回值             |         |
| -               | -       |

例如：

```
/* enable ADC0 inserted channel group external trigger */
```

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

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

函数adc\_external\_trigger\_source\_config描述见下表：

**表 3-23. 函数 **adc\_external\_trigger\_source\_config****

| 函数名称                                  | adc_external_trigger_source_config   |
|---------------------------------------|--|
| 函数原形                                  | void adc_external_trigger_source_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t external_trigger_source); |
| 功能描述                                  | 配置ADC外部触发源   |
| 先决条件                                  | -  |
| 被调用函数                                 | -  |
| 输入参数{in}                              |  |
| <b>adc_periph</b>                     | ADC外设  |
| <i>ADCx(x=0,1)</i>                    | ADC外设选择  |
| 输入参数{in}                              |  |
| <b>adc_channel_group</b>              | 通道组选择  |
| <i>ADC_REGULAR_CHANNEL</i>            | 规则通道组  |
| <i>ADC_INSERTED_CHANNEL</i>           | 注入通道组  |
| 输入参数{in}                              |  |
| <b>external_trigger_source</b>        | 规则通道组或注入通道组触发源   |
| <i>ADC0_1_EXTTRIG_REGULAR_TRIGSEL</i> | TRIGSEL事件（规则组）   |
| <i>ADC0_1_EXTTRIG_REGULAR_NONE</i>    | 软件事件（规则组）  |
| <i>ADC0_1_EXTTRIG</i>                 | TRIGSEL事件（注入组）   |



|   |           |
|---|-----------|
| <code>_INSERTED_TRIGSEL</code>            |           |
| <code>ADC0_1_EXTTRIG_INSERTED_NONE</code> | 软件事件（注入组） |
| 输出参数{out}                                 |           |
| -   | -         |
| 返回值                                       |           |
| -   | -         |

例如：

```
/* configure ADC0 regular channel trigger source: TRIGSEL */
```

```
adc_external_trigger_source_config(ADC0,ADC_REGULAR_CHANNEL,
ADC0_1_EXTTRIG_REGULAR_TRIGSEL);
```

### 函数 `adc_software_trigger_enable`

函数`adc_software_trigger_enable`描述见下表：

**表 3-24. 函数 `adc_software_trigger_enable`**

|                                   |  |
|-----------------------------------|--|
| 函数名称                              | <code>adc_software_trigger_enable</code>   |
| 函数原形                              | <code>void adc_software_trigger_enable(uint32_t adc_periph, uint8_t adc_channel_group);</code> |
| 功能描述                              | ADC软件触发使能  |
| 先决条件                              | -  |
| 被调用函数                             | -  |
| 输入参数{in}                          |  |
| <b>adc_periph</b>                 | ADC外设  |
| <code>ADCx(x=0,1)</code>          | ADC外设选择  |
| 输入参数{in}                          |  |
| <b>adc_channel_group</b>          | 通道组选择  |
| <code>ADC_REGULAR_CHANNEL</code>  | 规则通道组  |
| <code>ADC_INSERTED_CHANNEL</code> | 注入通道组  |
| 输出参数{out}                         |  |
| -                                 | -  |
| 返回值                               |  |
| -                                 | -  |

例如：

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

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

函数 `adc_regular_data_read`

函数`adc_inserted_regular_data_read`描述见下表:

表 3-25. 函数 `adc_regular_data_read`

|                          |   |
|--------------------------|---|
| 函数名称                     | <code>adc_regular_data_read</code>                                |
| 函数原形                     | <code>uint16_t adc_regular_data_read(uint32_t adc_periph);</code> |
| 功能描述                     | 读ADC规则组数据寄存器  |
| 先决条件                     | -   |
| 被调用函数                    | -   |
| 输入参数{in}                 |   |
| <code>adc_periph</code>  | ADC外设   |
| <code>ADCx(x=0,1)</code> | ADC外设选择   |
| 输出参数{out}                |   |
| -                        | -   |
| 返回值                      |   |
| <code>uint16_t</code>    | ADC转换值(0~0xFFFF)  |

例如:

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

```
uint16_t adc_value = 0;
```

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

函数 `adc_inserted_data_read`

函数`adc_inserted_regular_data_read`描述见下表:

表 3-26. 函数 `adc_inserted_data_read`

|   |  |
|---|--|
| 函数名称  | <code>adc_inserted_data_read</code>  |
| 函数原形  | <code>uint16_t adc_inserted_data_read(uint32_t adc_periph, uint8_t inserted_channel);</code> |
| 功能描述  | 读ADC注入组数据寄存器   |
| 先决条件  | -  |
| 被调用函数                                       | -  |
| 输入参数{in}                                    |  |
| <code>adc_periph</code>                     | ADC外设  |
| <code>ADCx(x=0,1)</code>                    | ADC外设选择  |
| 输入参数{in}                                    |  |
| <code>inserted_channel</code>               | 注入通道选择   |
| <code>ADC_INSERTED_CHANNEL_x(x=0..3)</code> | 注入通道x, x=0,1,2,3   |
| 输出参数{out}                                   |  |
| -   | -  |

| 返回值      |                  |
|----------|------------------|
| uint16_t | ADC转换值(0~0xFFFF) |

例如:

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

```
uint16_t adc_value = 0;
```

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

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

函数adc\_sync\_mode\_convert\_value\_read描述见下表:

表 3-27. 函数 adc\_sync\_mode\_convert\_value\_read

|           |  |
|-----------|--|
| 函数名称      | adc_sync_mode_convert_value_read                 |
| 函数原形      | uint32_t adc_sync_mode_convert_value_read(void); |
| 功能描述      | 在同步模式下，读ADC0和ADC1最近的一次转换结果                       |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| uint32_t  | ADC转换值 (0-0xFFFFFFFF)                            |

例如:

```
/* 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 ();
```

### 函数 adc\_watchdog0\_single\_channel\_enable

函数adc\_watchdog0\_single\_channel\_enable描述见下表:

表 3-28. 函数 adc\_watchdog0\_single\_channel\_enable

|          |   |
|----------|---|
| 函数名称     | adc_watchdog0_single_channel_enable   |
| 函数原形     | void adc_watchdog0_single_channel_enable(uint32_t adc_periph, uint8_t adc_channel); |
| 功能描述     | 配置ADC模拟看门狗0单通道有效  |
| 先决条件     | -   |
| 被调用函数    | -   |
| 输入参数{in} |   |

|  |  |
|--|--|
| <b>adc_periph</b>                          | ADC外设  |
| <i>ADCx(x=0,1)</i>                         | ADC外设选择  |
| <b>输入参数{in}</b>                            |  |
| <b>adc_channel</b>                         | 选择ADC通道  |
| <i>ADC_CHANNEL_x</i><br>( <i>x=0..17</i> ) | ADC通道 <i>x</i> ( <i>x=0..17</i> ) (只有ADC0, 可取值 <i>x=16</i> 和 <i>17</i> ) |
| <b>输出参数{out}</b>                           |  |
| -  | -  |
| <b>返回值</b>                                 |  |
| -  | -  |

例如:

```
/* configure ADC0 analog watchdog 0 single channel */
```

```
adc_watchdog0_single_channel_enable(ADC0, ADC_CHANNEL_1);
```

### 函数 **adc\_watchdog0\_group\_channel\_enable**

函数adc\_watchdog0\_group\_channel\_enable描述见下表:

**表 3-29. 函数 **adc\_watchdog0\_group\_channel\_enable****

|                                     |  |
|-------------------------------------|--|
| <b>函数名称</b>                         | adc_watchdog0_group_channel_enable   |
| <b>函数原形</b>                         | void adc_watchdog0_group_channel_enable(uint32_t adc_periph, uint8_t adc_channel_group); |
| <b>功能描述</b>                         | 配置ADC模拟看门狗0在通道组有效  |
| <b>先决条件</b>                         | -  |
| <b>被调用函数</b>                        | -  |
| <b>输入参数{in}</b>                     |  |
| <b>adc_periph</b>                   | ADC外设  |
| <i>ADCx(x=0,1)</i>                  | ADC外设选择  |
| <b>输入参数{in}</b>                     |  |
| <b>adc_channel_group</b>            | 通道组使用模拟看门狗   |
| <i>ADC_REGULAR_CHANNEL</i>          | 规则通道组  |
| <i>ADC_INSERTED_CHANNEL</i>         | 注入通道组  |
| <i>ADC_REGULAR_INSERTED_CHANNEL</i> | 规则和注入通道组   |
| <b>输出参数{out}</b>                    |  |
| -                                   | -  |
| <b>返回值</b>                          |  |
| -                                   | -  |

例如:

```
/* configure ADC0 analog watchdog 0 group channel */
```

```
adc_watchdog0_group_channel_enable(ADC0, ADC_REGULAR_CHANNEL);
```

### 函数 `adc_watchdog0_disable`

函数`adc_watchdog0_disable`描述见下表:

表 3-30. 函数 `adc_watchdog0_disable`

|             |  |
|-------------|--|
| 函数名称        | adc_watchdog0_disable                            |
| 函数原形        | void adc_watchdog0_disable(uint32_t adc_periph); |
| 功能描述        | ADC模拟看门狗0禁能                                      |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| adc_periph  | ADC外设  |
| ADCx(x=0,1) | ADC外设选择  |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如:

```
/* disable ADC0 analog watchdog 0 */
```

```
adc_watchdog0_disable(ADC0);
```

### 函数 `adc_watchdog1_channel_config`

函数`adc_watchdog1_channel_config`描述见下表:

表 3-31. 函数 `adc_watchdog1_channel_config`

|               |   |
|---------------|---|
| 函数名称          | adc_watchdog1_channel_config  |
| 函数原形          | void adc_watchdog1_channel_config(uint32_t adc_periph, uint32_t adc_channel, ControlStatus newvalue); |
| 功能描述          | 配置ADC模拟看门狗1单通道有效  |
| 先决条件          | -   |
| 被调用函数         | -   |
| 输入参数{in}      |   |
| adc_periph    | ADC外设   |
| ADCx(x=0,1)   | ADC外设选择   |
| 输入参数{in}      |   |
| adc_channel   | 选择ADC通道   |
| ADC_AWD1_SELE | ADC通道模拟看门狗1/2选择   |

|   |                              |
|---|------------------------------|
| <i>CTION_CHANNEL_</i><br><i>x (x=0..17),</i><br><i>ADC_AWD1_SELE</i><br><i>CTION_CHANNEL_</i><br><i>ALL</i> | (x=0..17, 只有ADC0可取值x=16和17)) |
| 输入参数{in}  |                              |
| <b>newvalue</b>   | 使能/禁能控制                      |
| <i>ENABLE</i>   | 使能                           |
| <i>DISABLE</i>  | 禁能                           |
| 输出参数{out}   |                              |
| -   | -                            |
| 返回值   |                              |
| -   | -                            |

例如:

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

```
adc_watchdog1_channel_config(ADC0, ADC_AWD1_SELECTION_CHANNEL_1,  
ENABLE);
```

### 函数 **adc\_watchdog1\_disable**

函数adc\_watchdog1\_disable描述见下表:

**表 3-32. 函数 **adc\_watchdog1\_disable****

|                    |  |
|--------------------|--|
| 函数名称               | adc_watchdog1_disable                            |
| 函数原形               | void adc_watchdog1_disable(uint32_t adc_periph); |
| 功能描述               | ADC模拟看门狗1禁能                                      |
| 先决条件               | -  |
| 被调用函数              | -  |
| 输入参数{in}           |  |
| <b>adc_periph</b>  | ADC外设  |
| <i>ADCx(x=0,1)</i> | ADC外设选择  |
| 输出参数{out}          |  |
| -                  | -  |
| 返回值                |  |
| -                  | -  |

例如:

```
/* disable ADC0 analog watchdog 1 */
```

```
adc_watchdog1_disable(ADC0);
```

**函数 adc\_watchdog0\_threshold\_config**

函数adc\_watchdog0\_threshold\_config描述见下表:

**表 3-33. 函数 adc\_watchdog0\_threshold\_config**

|                |  |
|----------------|--|
| 函数名称           | adc_watchdog0_threshold_config   |
| 函数原形           | void adc_watchdog0_threshold_config(uint32_t adc_periph, uint16_t low_threshold, uint16_t high_threshold); |
| 功能描述           | 配置ADC模拟看门狗0阈值  |
| 先决条件           | -  |
| 被调用函数          | -  |
| 输入参数{in}       |  |
| adc_periph     | ADC外设  |
| ADCx(x=0,1)    | ADC外设选择  |
| 输入参数{in}       |  |
| low_threshold  | 模拟看门狗0低阈值, 0..4095   |
| 输入参数{in}       |  |
| high_threshold | 模拟看门狗0高阈值, 0..4095   |
| 输出参数{out}      |  |
| -              | -  |
| 返回值            |  |
| -              | -  |

例如:

```
/* configure ADC0 analog watchdog 0 threshold */
```

```
adc_watchdog0_threshold_config(ADC0, 0x0400, 0x0A00);
```

**函数 adc\_watchdog1\_threshold\_config**

函数adc\_watchdog1\_threshold\_config描述见下表:

**表 3-34. 函数 adc\_watchdog1\_threshold\_config**

|               |  |
|---------------|--|
| 函数名称          | adc_watchdog1_threshold_config   |
| 函数原形          | void adc_watchdog1_threshold_config(uint32_t adc_periph, uint8_t low_threshold, uint8_t high_threshold); |
| 功能描述          | 配置ADC模拟看门狗1阈值  |
| 先决条件          | -  |
| 被调用函数         | -  |
| 输入参数{in}      |  |
| adc_periph    | ADC外设  |
| ADCx(x=0,1)   | ADC外设选择  |
| 输入参数{in}      |  |
| low_threshold | 模拟看门狗1低阈值, 0..255  |
| 输入参数{in}      |  |

|                       |                  |
|-----------------------|------------------|
| <b>high_threshold</b> | 模拟看门狗1高阈值，0..255 |
| <b>输出参数{out}</b>      |                  |
| -                     | -                |
| <b>返回值</b>            |                  |
| -                     | -                |

例如：

```
/* configure ADC0 analog watchdog 1 threshold */
```

```
adc_watchdog1_threshold_config(ADC0, 0x40, 0xA0);
```

### 函数 adc\_resolution\_config

函数adc\_resolution\_config描述见下表：

表 3-35. 函数 adc\_resolution\_config

|                    |  |
|--------------------|--|
| <b>函数名称</b>        | adc_resolution_config  |
| <b>函数原形</b>        | void adc_resolution_config(uint32_t adc_periph , uint32_t resolution); |
| <b>功能描述</b>        | 配置ADC分辨率   |
| <b>先决条件</b>        | -  |
| <b>被调用函数</b>       | -  |
| <b>输入参数{in}</b>    |  |
| <b>adc_periph</b>  | ADC外设  |
| ADCx(x=0,1)        | ADC外设选择  |
| <b>输入参数{in}</b>    |  |
| <b>resolution</b>  | ADC分辨率   |
| ADC_RESOLUTION_12B | 12位分辨率   |
| ADC_RESOLUTION_10B | 10位分辨率   |
| ADC_RESOLUTION_8B  | 8位分辨率  |
| ADC_RESOLUTION_6B  | 6位分辨率  |
| <b>输出参数{out}</b>   |  |
| -                  | -  |
| <b>返回值</b>         |  |
| -                  | -  |

Example:

```
/* configure ADC0 resolution */
```

```
adc_resolution_config(ADC0, ADC_RESOLUTION_10B);
```



## 函数 `adc_oversample_mode_config`

函数 `adc_oversample_mode_config` 描述见下表：

**表 3-36. 函数 `adc_oversample_mode_config`**

|   |  |
|---|--|
| 函数名称                                      | <code>adc_oversample_mode_config</code>  |
| 函数原形                                      | <code>void adc_oversample_mode_config(uint32_t adc_periph, uint32_t mode, uint16_t shift, uint8_t ratio);</code> |
| 功能描述                                      | 配置ADC过采样模式   |
| 先决条件                                      | -  |
| 被调用函数                                     | -  |
| 输入参数{in}                                  |  |
| <code>adc_periph</code>                   | ADC外设  |
| <code>ADCx(x=0,1)</code>                  | ADC外设选择  |
| 输入参数{in}                                  |  |
| <code>mode</code>                         | ADC过采样触发模式   |
| <code>ADC_OVERSAMPLING_ALL_CONVERT</code> | 在一个触发之后，对一个通道连续进行过采样转换   |
| <code>ADC_OVERSAMPLING_ONE_CONVERT</code> | 在一个触发之后，对一个通道只进行一次过采样转换  |
| 输入参数{in}                                  |  |
| <code>shift</code>                        | ADC过滤采样移位  |
| <code>ADC_OVERSAMPLING_SHIFT_NONE</code>  | 不移位  |
| <code>ADC_OVERSAMPLING_SHIFT_1B</code>    | 移1位  |
| <code>ADC_OVERSAMPLING_SHIFT_2B</code>    | 移2位  |
| <code>ADC_OVERSAMPLING_SHIFT_3B</code>    | 移3位  |
| <code>ADC_OVERSAMPLING_SHIFT_4B</code>    | 移4位  |
| <code>ADC_OVERSAMPLING_SHIFT_5B</code>    | 移5位  |
| <code>ADC_OVERSAMPLING_SHIFT_6B</code>    | 移6位  |

|                                       |         |
|---------------------------------------|---------|
| MPLING_SHIFT<br>_6B                   |         |
| ADC_OVERSA<br>MPLING_SHIFT<br>_7B     | 移7位     |
| ADC_OVERSA<br>MPLING_SHIFT<br>_8B     | 移8位     |
| 输入参数{in}                              |         |
| ratio                                 | ADC过采样率 |
| ADC_OVERSA<br>MPLING_RATIO<br>_MUL2   | 2x      |
| ADC_OVERSA<br>MPLING_RATIO<br>_MUL4   | 4x      |
| ADC_OVERSA<br>MPLING_RATIO<br>_MUL8   | 8x      |
| ADC_OVERSA<br>MPLING_RATIO<br>_MUL16  | 16x     |
| ADC_OVERSA<br>MPLING_RATIO<br>_MUL32  | 32x     |
| ADC_OVERSA<br>MPLING_RATIO<br>_MUL64  | 64x     |
| ADC_OVERSA<br>MPLING_RATIO<br>_MUL128 | 128x    |
| ADC_OVERSA<br>MPLING_RATIO<br>_MUL256 | 256x    |
| 输出参数{out}                             |         |
| -                                     | -       |
| 返回值                                   |         |
| -                                     | -       |

Example:

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

```
adc_oversample_mode_config(ADC1, ADC_OVERSAMPLING_ALL_CONVERT,
```

ADC\_OVERSAMPLING\_SHIFT\_4B, ADC\_OVERSAMPLING\_RATIO\_MUL16);

### 函数 `adc_oversample_mode_enable`

函数`adc_oversample_mode_enable`描述见下表:

表 3-37. 函数 `adc_oversample_mode_enable`

|                          |  |
|--------------------------|--|
| 函数名称                     | <code>adc_oversample_mode_enable</code>                            |
| 函数原形                     | <code>void adc_oversample_mode_enable(uint32_t adc_periph);</code> |
| 功能描述                     | 使能ADC过采样   |
| 先决条件                     | -  |
| 被调用函数                    | -  |
| 输入参数{in}                 |  |
| <code>adc_periph</code>  | ADC外设  |
| <code>ADCx(x=0,1)</code> | ADC外设选择  |
| 输出参数{out}                |  |
| -                        | -  |
| 返回值                      |  |
| -                        | -  |

Example:

```
/* enable ADC0 oversample mode */
adc_oversample_mode_enable (ADC0);
```

### 函数 `adc_oversample_mode_disable`

函数`adc_oversample_mode_disable`描述见下表:

表 3-38. 函数 `adc_oversample_mode_disable`

|                          |   |
|--------------------------|---|
| 函数名称                     | <code>adc_oversample_mode_disable</code>                            |
| 函数原形                     | <code>void adc_oversample_mode_disable(uint32_t adc_periph);</code> |
| 功能描述                     | 禁能ADC过采样  |
| 先决条件                     | -   |
| 被调用函数                    | -   |
| 输入参数{in}                 |   |
| <code>adc_periph</code>  | ADC外设   |
| <code>ADCx(x=0,1)</code> | ADC外设选择   |
| 输出参数{out}                |   |
| -                        | -   |
| 返回值                      |   |
| -                        | -   |

Example:

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

adc\_oversample\_mode\_disable (ADC0);

### 函数 adc\_flag\_get

函数adc\_flag\_get描述见下表:

表 3-39. 函数 adc\_flag\_get

|               |  |
|---------------|--|
| 函数名称          | adc_flag_get   |
| 函数原形          | FlagStatus adc_flag_get(uint32_t adc_periph, uint32_t flag); |
| 功能描述          | 获取ADC标志位   |
| 先决条件          | -  |
| 被调用函数         | -  |
| 输入参数{in}      |  |
| adc_periph    | ADC外设  |
| ADCx(x=0,1)   | ADC外设选择  |
| 输入参数{in}      |  |
| flag          | ADC标志位   |
| ADC_FLAG_WDE0 | 模拟看门狗0事件标志位  |
| ADC_FLAG_EOC  | 组转换结束标志位   |
| ADC_FLAG_EOIC | 注入通道组转换结束标志位   |
| ADC_FLAG_STIC | 注入通道组转换开始标志位   |
| ADC_FLAG_STRC | 规则通道组转换开始标志位   |
| ADC_FLAG_WDE1 | 模拟看门狗1事件标志位  |
| 输出参数{out}     |  |
| -             | -  |
| 返回值           |  |
| FlagStatus    | SET或RESET  |

例如:

```
/* get the ADC0 analog watchdog 0 flag bits */
```

```
FlagStatus flag_value;
```

```
flag_value = adc_flag_get(ADC0, ADC_FLAG_WDE0);
```

### 函数 adc\_flag\_clear

函数adc\_flag\_clear描述见下表:

表 3-40. 函数 adc\_flag\_clear

|       |  |
|-------|--|
| 函数名称  | adc_flag_clear   |
| 函数原形  | void adc_flag_clear(uint32_t adc_periph, uint32_t flag); |
| 功能描述  | 清除ADC标志位   |
| 先决条件  | -  |
| 被调用函数 | -  |

| 输入参数{in}             |              |
|----------------------|--------------|
| <b>adc_periph</b>    | ADC外设        |
| <i>ADCx(x=0,1)</i>   | ADC外设选择      |
| 输入参数{in}             |              |
| <b>flag</b>          | ADC标志位       |
| <i>ADC_FLAG_WDE0</i> | 模拟看门狗0事件标志位  |
| <i>ADC_FLAG_EOC</i>  | 组转换结束标志位     |
| <i>ADC_FLAG_EOIC</i> | 注入通道组转换结束标志位 |
| <i>ADC_FLAG_STIC</i> | 注入通道组转换开始标志位 |
| <i>ADC_FLAG_STRC</i> | 规则通道组转换开始标志位 |
| <i>ADC_FLAG_WDE1</i> | 模拟看门狗1事件标志位  |
| 输出参数{out}            |              |
| -                    | -            |
| 返回值                  |              |
| -                    | -            |

例如：

```
/* clear the ADC0 analog watchdog 0 flag bits */
```

```
adc_flag_clear(ADC0, ADC_FLAG_WDE0);
```

## 函数 adc\_interrupt\_enable

函数adc\_interrupt\_enable描述见下表：

表 3-41. 函数 adc\_interrupt\_enable

| <b>函数名称</b>         | adc_interrupt_enable  |
|---------------------|---|
| <b>函数原形</b>         | void adc_interrupt_enable(uint32_t adc_periph, uint32_t interrupt); |
| <b>功能描述</b>         | ADC中断使能   |
| <b>先决条件</b>         | -   |
| <b>被调用函数</b>        | -   |
| 输入参数{in}            |   |
| <b>adc_periph</b>   | ADC外设   |
| <i>ADCx(x=0,1)</i>  | ADC外设选择   |
| 输入参数{in}            |   |
| <b>interrupt</b>    | ADC中断标志位  |
| <i>ADC_INT_WDE0</i> | 模拟看门狗0中断标志位   |
| <i>ADC_INT_EOC</i>  | 组转换结束中断标志位  |
| <i>ADC_INT_EOIC</i> | 注入通道组转换结束中断标志位  |
| <i>ADC_INT_WDE1</i> | 模拟看门狗1中断标志位   |
| 输出参数{out}           |   |
| -                   | -   |
| 返回值                 |   |
| -                   | -   |

例如：

```
/* enable ADC0 analog watchdog 0 interrupt */
adc_interrupt_enable(ADC0, ADC_INT_WDE0);
```

### 函数 adc\_interrupt\_disable

函数adc\_interrupt\_disable描述见下表：

表 3-42. 函数 adc\_interrupt\_disable

|              |   |
|--------------|---|
| 函数名称         | adc_interrupt_disable   |
| 函数原形         | void adc_interrupt_enable(uint32_t adc_periph, uint32_t interrupt); |
| 功能描述         | ADC中断禁能   |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| adc_periph   | ADC外设   |
| ADCx(x=0,1)  | ADC外设选择   |
| 输入参数{in}     |   |
| interrupt    | ADC中断标志位  |
| ADC_INT_WDE0 | 模拟看门狗0中断标志位   |
| ADC_INT_EOC  | 组转换结束中断标志位  |
| ADC_INT_EOIC | 注入通道组转换结束中断标志位  |
| ADC_INT_WDE1 | 模拟看门狗1中断标志位   |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如：

```
/* disable ADC0 analog watchdog 0 interrupt */
adc_interrupt_disable(ADC0, ADC_INT_WDE0);
```

### 函数 adc\_interrupt\_flag\_get

函数adc\_interrupt\_flag\_get描述见下表：

表 3-43. 函数 adc\_interrupt\_flag\_get

|          |  |
|----------|--|
| 函数名称     | adc_interrupt_flag_get   |
| 函数原形     | FlagStatus adc_interrupt_flag_get(uint32_t adc_periph, uint32_t int_flag); |
| 功能描述     | 获取ADC中断标志位   |
| 先决条件     | -  |
| 被调用函数    | -  |
| 输入参数{in} |  |

|                          |                |
|--------------------------|----------------|
| <b>adc_periph</b>        | ADC外设          |
| <i>ADCx(x=0,1)</i>       | ADC外设选择        |
| <b>输入参数{in}</b>          |                |
| <b>int_flag</b>          | ADC中断标志位       |
| <i>ADC_INT_FLAG_WDE0</i> | 模拟看门狗0中断标志位    |
| <i>ADC_INT_FLAG_EOC</i>  | 组转换结束中断标志位     |
| <i>ADC_INT_FLAG_EOIC</i> | 注入通道组转换结束中断标志位 |
| <i>ADC_INT_FLAG_WDE1</i> | 模拟看门狗1中断标志位    |
| <b>输出参数{out}</b>         |                |
| -                        | -              |
| <b>返回值</b>               |                |
| <b>FlagStatus</b>        | SET或RESET      |

例如：

```
/* get the ADC0 analog watchdog 0 interrupt bits */
```

```
FlagStatus flag_value;
```

```
flag_value = adc_interrupt_flag_get(ADC0, ADC_INT_WDE0);
```

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

函数adc\_interrupt\_flag\_clear描述见下表：

**表 3-44. 函数 **adc\_interrupt\_flag\_clear****

|                          |  |
|--------------------------|--|
| <b>函数名称</b>              | adc_interrupt_flag_clear   |
| <b>函数原形</b>              | void adc_interrupt_flag_clear(uint32_t adc_periph, uint32_t int_flag); |
| <b>功能描述</b>              | 清除ADC中断标志位   |
| <b>先决条件</b>              | -  |
| <b>被调用函数</b>             | -  |
| <b>输入参数{in}</b>          |  |
| <b>adc_periph</b>        | ADC外设  |
| <i>ADCx(x=0,1)</i>       | ADC外设选择  |
| <b>输入参数{in}</b>          |  |
| <b>int_flag</b>          | ADC中断标志位   |
| <i>ADC_INT_FLAG_WDE0</i> | 模拟看门狗0中断标志位  |
| <i>ADC_INT_FLAG_EOC</i>  | 组转换结束中断标志位   |
| <i>ADC_INT_FLAG_EOIC</i> | 注入通道组转换结束中断标志位   |

|                   |             |
|-------------------|-------------|
| OIC               |             |
| ADC_INT_FLAG_WDE1 | 模拟看门狗1中断标志位 |
| 输出参数{out}         |             |
| -                 | -           |
| 返回值               |             |
| -                 | -           |

例如：

```
/* clear the ADC0 analog watchdog 0 interrupt bits */
```

```
adc_interrupt_flag_clear(ADC0, ADC_INT_WDE0);
```

### 3.3. BKP

位于备份域中的备份寄存器由VDD电源供电，备份寄存器有10个16位（20字节）寄存器可用来存储并保护用户应用数据，从待机模式唤醒或系统复位也不会对这些寄存器造成影响。章节[3.3.1](#)描述了BKP的寄存器列表，章节[3.3.2](#)对BKP库函数进行说明。

#### 3.3.1. 外设寄存器说明

BKP寄存器列表如下表所示：

表 3-45. BKP 寄存器

| 寄存器名称              | 寄存器描述        |
|--------------------|--------------|
| BKP_DATAx (x=0..9) | 备份数据寄存器      |
| BKP_OCTL           | RTC信号输出控制寄存器 |
| BKP_TPCTL          | 侵入引脚控制寄存器    |
| BKP_TPCS           | 侵入控制状态寄存器    |

#### 3.3.2. 外设库函数说明

BKP库函数列表如下表所示：

表 3-46. BKP 库函数

| 库函数名称                              | 库函数描述         |
|------------------------------------|---------------|
| bkp_deinit                         | 复位备份寄存器       |
| bkp_data_write                     | 写备份数据寄存器      |
| bkp_data_read                      | 读备份数据寄存器      |
| bkp_rtc_calibration_output_enable  | RTC时钟校准输出使能   |
| bkp_rtc_calibration_output_disable | RTC时钟校准输出失能   |
| bkp_rtc_signal_output_enable       | RTC闹钟或秒信号输出使能 |
| bkp_rtc_signal_output_disable      | RTC闹钟或秒信号输出失能 |



| 库函数名称                               | 库函数描述                     |
|-------------------------------------|---------------------------|
| bkp_rtc_output_select               | RTC输出选择，RTC输出可选择为闹钟脉冲或秒脉冲 |
| bkp_rtc_clock_output_select         | RTC时钟输出选择                 |
| bkp_rtc_clock_calibration_direction | RTC时钟校准方向选择               |
| bkp_rtc_calibration_value_set       | 设置RTC时钟校准值                |
| bkp_osc32in_pin_select              | OSC32IN引脚选择               |
| bkp_tamper_detection_enable         | TAMPER引脚使能                |
| bkp_tamper_detection_disable        | TAMPER引脚失能                |
| bkp_tamper_active_level_set         | TAMPER引脚有效电平设置            |
| bkp_interrupt_enable                | TAMPER中断使能                |
| bkp_interrupt_disable               | TAMPER中断失能                |
| bkp_flag_get                        | 获取标志位                     |
| bkp_flag_clear                      | 清除标志位                     |
| bkp_interrupt_flag_get              | 获取中断标志位                   |
| bkp_interrupt_flag_clear            | 清除中断标志位                   |

### 枚举类型 `bkp_data_register_enum`

表 3-47. 枚举类型 `bkp_data_register_enum`

| 成员名称       | 功能描述      |
|------------|-----------|
| BKP_DATA_0 | BKP数据寄存器0 |
| BKP_DATA_1 | BKP数据寄存器1 |
| BKP_DATA_2 | BKP数据寄存器2 |
| BKP_DATA_3 | BKP数据寄存器3 |
| BKP_DATA_4 | BKP数据寄存器4 |
| BKP_DATA_5 | BKP数据寄存器5 |
| BKP_DATA_6 | BKP数据寄存器6 |
| BKP_DATA_7 | BKP数据寄存器7 |
| BKP_DATA_8 | BKP数据寄存器8 |
| BKP_DATA_9 | BKP数据寄存器9 |

### 函数 `bkp_deinit`

函数**bkp\_deinit**描述见下表：

表 3-48. 函数 `bkp_deinit`

|           |  |
|-----------|--|
| 函数名称      | <code>bkp_deinit</code>  |
| 函数原型      | <code>void bkp_deinit(void);</code>                                    |
| 功能描述      | 复位备份数据寄存器  |
| 先决条件      | -  |
| 被调用函数     | <code>rcu_bkp_reset_enable</code> / <code>rcu_bkp_reset_disable</code> |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |

|     |   |
|-----|---|
| -   | - |
| 返回值 |   |
| -   | - |

例如:

```
/* reset BKP registers */
```

```
bkp_deinit();
```

### 函数 bkp\_data\_write

函数bkp\_data\_write描述见下表:

表 3-49. 函数 bkp\_data\_write

|                      |   |
|----------------------|---|
| 函数名称                 | bkp_data_write  |
| 函数原型                 | void bkp_data_write(bkp_data_register_enum register_number, uint16_t data); |
| 功能描述                 | 写备份数据寄存器  |
| 先决条件                 | -   |
| 被调用函数                | -   |
| 输入参数{in}             |   |
| register_number      | 参考枚举 <a href="#">表3-47. 枚举类型bkp_data_register_enum</a>                      |
| BKP_DATA_x(x = 0..9) | BKP数据寄存器x   |
| 输入参数{in}             |   |
| Data                 | 待写入BKP数据寄存器的数据  |
| 0-0xffff             | 数值  |
| 输出参数{out}            |   |
| -                    | -   |
| 返回值                  |   |
| -                    | -   |

例如:

```
/* write BKP data register */
```

```
bkp_data_write (BKP_DATA_0, 0x1226);
```

### 函数 bkp\_data\_read

函数bkp\_data\_read描述见下表:

表 3-50. 函数 bkp\_data\_read

|      |   |
|------|---|
| 函数名称 | bkp_data_read   |
| 函数原型 | uint16_t bkp_data_read(bkp_data_register_enum register_number); |
| 功能描述 | 读备份数据寄存器  |
| 先决条件 | -   |

|                      |   |
|----------------------|---|
| 被调用函数                | -   |
| 输入参数{in}             |   |
| register_number      | 参考枚举 <a href="#">表3-47. 枚举类型 bkp_data_register_enum</a> |
| BKP_DATA_x(x = 0..9) | BKP数据寄存器x   |
| 输出参数{out}            |   |
| -                    | -   |
| 返回值                  |   |
| uint16_t             | 0-0xffff  |

例如：

```
/* read BKP data register */
```

```
uint16_t data;
```

```
data = bkp_data_read (BKP_DATA_0);
```

### 函数 bkp\_rtc\_calibration\_output\_enable

函数 bkp\_rtc\_calibration\_output\_enable 描述见下表：

表 3-51. 函数 bkp\_rtc\_calibration\_output\_enable

|           |   |
|-----------|---|
| 函数名称      | bkp_rtc_calibration_output_enable             |
| 函数原型      | void bkp_rtc_calibration_output_enable(void); |
| 功能描述      | RTC时钟校准输出使能                                   |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| -         | -   |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

```
/* enable RTC clock calibration output */
```

```
bkp_rtc_calibration_output_enable();
```

### 函数 bkp\_rtc\_calibration\_output\_disable

函数 bkp\_rtc\_calibration\_output\_disable 描述见下表：

表 3-52. 函数 bkp\_rtc\_calibration\_output\_disable

|      |  |
|------|--|
| 函数名称 | bkp_rtc_calibration_output_disable             |
| 函数原型 | void bkp_rtc_calibration_output_disable(void); |

|           |             |
|-----------|-------------|
| 功能描述      | RTC时钟校准输出失能 |
| 先决条件      | -           |
| 被调用函数     | -           |
| 输入参数{in}  |             |
| -         | -           |
| 输出参数{out} |             |
| -         | -           |
| 返回值       |             |
| -         | -           |

例如：

```
/* disable RTC clock calibration output */
```

```
bkp_rtc_calibration_output_disable();
```

### 函数 bkp\_rtc\_signal\_output\_enable

函数bkp\_rtc\_signal\_output\_enable描述见下表：

表 3-53. 函数 bkp\_rtc\_signal\_output\_enable

|           |   |
|-----------|---|
| 函数名称      | bkp_rtc_signal_output_enable              |
| 函数原型      | void bkp_rtc_signal_output_enable (void); |
| 功能描述      | RTC闹钟或秒信号输出使能                             |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| -         | -   |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

```
/* enable RTC alarm or second signal output */
```

```
bkp_rtc_signal_output_enable();
```

### 函数 bkp\_rtc\_signal\_output\_disable

函数bkp\_rtc\_signal\_output\_disable描述见下表：

表 3-54. 函数 bkp\_rtc\_signal\_output\_disable

|      |  |
|------|--|
| 函数名称 | bkp_rtc_signal_output_disable              |
| 函数原型 | void bkp_rtc_signal_output_disable (void); |
| 功能描述 | RTC闹钟或秒信号输出失能                              |

|           |   |
|-----------|---|
| 先决条件      | - |
| 被调用函数     | - |
| 输入参数{in}  |   |
| -         | - |
| 输出参数{out} |   |
| -         | - |
| 返回值       |   |
| -         | - |

例如：

```
/* disable RTC alarm or second signal output */
```

```
bkp_rtc_signal_output_disable();
```

### 函数 bkp\_rtc\_output\_select

函数bkp\_rtc\_output\_select描述见下表：

表 3-55. 函数 bkp\_rtc\_output\_select

|                         |  |
|-------------------------|--|
| 函数名称                    | bkp_rtc_output_select                            |
| 函数原型                    | void bkp_rtc_output_select (uint16_t outputsel); |
| 功能描述                    | RTC输出选择，RTC输出可选择为闹钟脉冲或秒脉冲                        |
| 先决条件                    | -  |
| 被调用函数                   | -  |
| 输入参数{in}                |  |
| outputsel               | RTC输出选择  |
| RTC_OUTPUT_ALARM_PULSE  | RTC闹钟脉冲被选择为RTC输出                                 |
| RTC_OUTPUT_SECOND_PULSE | RTC秒脉冲被选择为RTC输出                                  |
| 输出参数{out}               |  |
| -                       | -  |
| 返回值                     |  |
| -                       | -  |

例如：

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

```
bkp_rtc_output_select (RTC_OUTPUT_ALARM_PULSE);
```

### 函数 bkp\_rtc\_clock\_output\_select

函数bkp\_rtc\_clock\_output\_select描述见下表：

表 3-56. 函数 bkp\_rtc\_clock\_output\_select

|                  |  |
|------------------|--|
| 函数名称             | bkp_rtc_clock_output_select                          |
| 函数原型             | void bkp_rtc_clock_output_select(uint16_t clocksel); |
| 功能描述             | RTC时钟输出选择，RTC时钟输出可选择为不分频或64分频                        |
| 先决条件             | -  |
| 被调用函数            | -  |
| 输入参数{in}         |  |
| clocksel         | RTC时钟输出选择  |
| RTC_CLOCK_DIV_64 | RTC时钟输出被选择为64分频                                      |
| RTC_CLOCK_DIV_1  | RTC时钟输出被选择为不分频                                       |
| 输出参数{out}        |  |
| -                | -  |
| 返回值              |  |
| -                | -  |

例如：

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

```
bkp_rtc_clock_output_select (RTC_CLOCK_DIV_64);
```

### 函数 bkp\_rtc\_clock\_calibration\_direction

函数bkp\_rtc\_clock\_calibration\_direction描述见下表：

表 3-57. 函数 bkp\_rtc\_clock\_calibration\_direction

|                     |  |
|---------------------|--|
| 函数名称                | bkp_rtc_clock_calibration_direction                            |
| 函数原型                | void bkp_rtc_clock_calibration_direction (uint16_t direction); |
| 功能描述                | RTC时钟校准方向选择，RTC时钟校准方向可选择为变快或变慢                                 |
| 先决条件                | -  |
| 被调用函数               | -  |
| 输入参数{in}            |  |
| direction           | RTC时钟校准方向  |
| RTC_CLOCK_SLOW_DOWN | RTC时钟变慢  |
| RTC_CLOCK_SPEED_UP  | RTC时钟变快  |
| 输出参数{out}           |  |
| -                   | -  |
| 返回值                 |  |
| -                   | -  |

例如：

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

```
bkp_rtc_clock_calibration_direction (RTC_CLOCK_SLOWED_DOWN);
```

### 函数 bkp\_rtc\_calibration\_value\_set

函数bkp\_rtc\_calibration\_value\_set描述见下表：

**表 3-58. 函数 bkp\_rtc\_calibration\_value\_set**

|             |  |
|-------------|--|
| 函数名称        | bkp_rtc_calibration_value_set                      |
| 函数原型        | void bkp_rtc_calibration_value_set(uint8_t value); |
| 功能描述        | RTC时钟校准值   |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| value       | RTC时钟校准值   |
| 0x00 - 0x7F | 校准值  |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如：

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

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

### 函数 bkp\_osc32in\_pin\_select

函数bkp\_osc32in\_pin\_select描述见下表：

**表 3-59. 函数 bkp\_osc32in\_pin\_select**

|              |   |
|--------------|---|
| 函数名称         | bkp_osc32in_pin_select                          |
| 函数原型         | void bkp_osc32in_pin_select(uint16_t inputpin); |
| 功能描述         | OSC32IN引脚选择                                     |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| inputpin     | OSC32IN引脚选择                                     |
| OSC32IN_PC13 | 选择PC13作为OSC32IN引脚                               |
| OSC32IN_PC14 | 选择PC14作为OSC32IN引脚                               |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如：

```
/* select OSC32IN pin */
bkp_osc32in_pin_select (OSC32IN_PC14);
```

### 函数 bkp\_tamper\_detection\_enable

函数bkp\_tamper\_detection\_enable描述见下表：

**表 3-60. 函数 bkp\_tamper\_detection\_enable**

|           |  |
|-----------|--|
| 函数名称      | bkp_tamper_detection_enable              |
| 函数原型      | void bkp_tamper_detection_enable (void); |
| 功能描述      | TAMPER引脚使能                               |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如：

```
/* enable tamper pin detection */
bkp_tamper_detection_enable();
```

### 函数 bkp\_tamper\_detection\_disable

函数bkp\_tamper\_detection\_disable描述见下表：

**表 3-61. 函数 bkp\_tamper\_detection\_disable**

|           |   |
|-----------|---|
| 函数名称      | bkp_tamper_detection_disable              |
| 函数原型      | void bkp_tamper_detection_disable (void); |
| 功能描述      | TAMPER引脚失能                                |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| -         | -   |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：



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

```
bkp_tamper_detection_disable ();
```

### 函数 bkp\_tamper\_active\_level\_set

函数bkp\_tamper\_active\_level\_set描述见下表：

**表 3-62. 函数 bkp\_tamper\_active\_level\_set**

|                        |  |
|------------------------|--|
| 函数名称                   | bkp_tamper_active_level_set                        |
| 函数原型                   | void bkp_tamper_active_level_set (uint16_t level); |
| 功能描述                   | TAMPER引脚有效电平设置                                     |
| 先决条件                   | -  |
| 被调用函数                  | -  |
| 输入参数{in}               |  |
| level                  | TAMPER引脚有效电平                                       |
| TAMPER_PIN_ACTIVE_HIGH | TAMPER引脚高电平有效                                      |
| TAMPER_PIN_ACTIVE_LOW  | TAMPER引脚低电平有效                                      |
| 输出参数{out}              |  |
| -                      | -  |
| 返回值                    |  |
| -                      | -  |

例如：

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

```
bkp_tamper_active_level_set (TAMPER_PIN_ACTIVE_HIGH);
```

### 函数 bkp\_tamper\_interrupt\_enable

函数bkp\_tamper\_interrupt\_enable描述见下表：

**表 3-63. 函数 bkp\_tamper\_interrupt\_enable**

|           |  |
|-----------|--|
| 函数名称      | bkp_tamper_interrupt_enable              |
| 函数原型      | void bkp_tamper_interrupt_enable (void); |
| 功能描述      | TAMPER中断使能                               |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |

例如：

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

```
bkp_tamper_interrupt_enable ();
```

### 函数 bkp\_tamper\_interrupt\_disable

函数bkp\_tamper\_interrupt\_disable描述见下表：

表 3-64. 函数 bkp\_tamper\_interrupt\_disable

|           |   |
|-----------|---|
| 函数名称      | bkp_tamper_interrupt_disable              |
| 函数原型      | void bkp_tamper_interrupt_disable (void); |
| 功能描述      | TAMPER中断失能                                |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| -         | -   |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

```
/* disable tamper pin interrupt */
```

```
bkp_tamper_interrupt_disable ();
```

### 函数 bkp\_flag\_get

函数bkp\_flag\_get描述见下表：

表 3-65. 函数 bkp\_flag\_get

|                 |   |
|-----------------|---|
| 函数名称            | bkp_flag_get                            |
| 函数原型            | FlagStatus bkp_flag_get(uint16_t flag); |
| 功能描述            | 获取标志位                                   |
| 先决条件            | -                                       |
| 被调用函数           | -                                       |
| 输入参数{in}        |   |
| BKP_FLAG_TAMPER | 侵入事件标志                                  |
| 输出参数{out}       |   |
| -               | -                                       |
| 返回值             |   |

|            |           |
|------------|-----------|
| FlagStatus | SET或RESET |
|------------|-----------|

例如:

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

```
FlagStatus status;
```

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

### 函数 bkp\_flag\_clear

函数bkp\_flag\_clear描述见下表:

表 3-66. 函数 bkp\_flag\_clear

|                 |                                     |
|-----------------|-------------------------------------|
| 函数名称            | bkp_flag_clear                      |
| 函数原型            | void bkp_flag_clear(uint16_t flag); |
| 功能描述            | 清除标志位                               |
| 先决条件            | -                                   |
| 被调用函数           | -                                   |
| 输入参数{in}        |                                     |
| BKP_FLAG_TAMPER | 侵入事件标志                              |
| 输出参数{out}       |                                     |
| -               | -                                   |
| 返回值             |                                     |
| -               | -                                   |

例如:

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

```
bkp_flag_clear (BKP_FLAG_TAMPER);
```

### 函数 bkp\_interrupt\_flag\_get

函数bkp\_interrupt\_flag\_get描述见下表:

表 3-67. 函数 bkp\_interrupt\_flag\_get

|                     |   |
|---------------------|---|
| 函数名称                | bkp_interrupt_flag_get                            |
| 函数原型                | FlagStatus bkp_interrupt_flag_get(uint16_t flag); |
| 功能描述                | 获取中断标志位   |
| 先决条件                | -   |
| 被调用函数               | -   |
| 输入参数{in}            |   |
| BKP_INT_FLAG_TAMPER | 侵入事件中中断标志   |
| 输出参数{out}           |   |

|            |           |
|------------|-----------|
| -          | -         |
| 返回值        |           |
| FlagStatus | SET或RESET |

例如：

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

```
bkp_interrupt_flag_get (BKP_FLAG_TAMPER);
```

### 函数 bkp\_interrupt\_flag\_clear

函数bkp\_interrupt\_flag\_clear描述见下表：

表 3-68. 函数 bkp\_interrupt\_flag\_clear

|                     |   |
|---------------------|---|
| 函数名称                | bkp_interrupt_flag_clear                      |
| 函数原型                | void bkp_interrupt_flag_clear(uint16_t flag); |
| 功能描述                | 清除中断标志位                                       |
| 先决条件                | -   |
| 被调用函数               | -   |
| 输入参数{in}            |   |
| BKP_INT_FLAG_TAMPER | 侵入事件中断标志                                      |
| 输出参数{out}           |   |
| -                   | -   |
| 返回值                 |   |
| -                   | -   |

例如：

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

```
bkp_interrupt_flag_clear (BKP_INT_FLAG_TAMPER);
```

## 3.4. CAN

CAN（Controller Area Network）总线是一种可以在无主机情况下实现微处理器和设备之间相互通信的总线标准。CAN网络接口支持CAN总线协议2.0A/B、ISO11898-1:2015规范和BOSCH CAN-FD规范。章节[3.4.1](#)描述了CAN的寄存器列表，章节[3.4.2](#)对CAN库函数进行说明。

### 3.4.1. 外设寄存器说明

CAN寄存器列表如下表所示：

表 3-69. CAN 寄存器

|          |         |
|----------|---------|
| 寄存器名称    | 寄存器描述   |
| CAN_CTL0 | 控制寄存器 0 |

| 寄存器名称                        | 寄存器描述                         |
|------------------------------|-------------------------------|
| CAN_CTL1                     | 控制寄存器 1                       |
| CAN_TIMER                    | 计数器寄存器                        |
| CAN_RMPUBF                   | 接收邮箱公有过滤寄存器                   |
| CAN_ERR0                     | 错误寄存器 0                       |
| CAN_ERR1                     | 错误寄存器 1                       |
| CAN_INTEN                    | 中断使能寄存器                       |
| CAN_STAT                     | 状态寄存器                         |
| CAN_CTL2                     | 控制寄存器 2                       |
| CAN_CRCC                     | 常规帧 CRC 寄存器                   |
| CAN_RFIFOPUBF                | 接收 FIFO 共有过滤寄存器               |
| CAN_RFIFOIFMN                | 接收 FIFO 标识符过滤元素匹配序号寄存器        |
| CAN_BT                       | 位时间寄存器                        |
| CAN_RFIFOMPFX<br>(x = 0..31) | 接收 FIFO/邮箱私有过滤 x 寄存器          |
| CAN_PN_CTL0                  | 虚拟联网模式控制寄存器 0                 |
| CAN_PN_TO                    | 虚拟联网模式超时寄存器                   |
| CAN_PN_STAT                  | 虚拟联网模式状态寄存器                   |
| CAN_PN_EID0                  | 虚拟联网模式期望标识符 0 寄存器             |
| CAN_PN_EDLC                  | 虚拟联网模式期望 DLC 寄存器              |
| CAN_PN_EDL0                  | 虚拟联网模式期望数据低字 0 寄存器            |
| CAN_PN_EDL1                  | 虚拟联网模式期望数据低字 1 寄存器            |
| CAN_PN_IFEID1                | 虚拟联网模式标识符过滤器 / 期望标识符 1 寄存器    |
| CAN_PN_DF0EDH<br>0           | 虚拟联网模式数据 0 过滤器 / 期望数据高字 0 寄存器 |
| CAN_PN_DF1EDH<br>1           | 虚拟联网模式数据 1 过滤器 / 期望数据高字 1 寄存器 |
| CAN_PN_RWMxCS<br>(x = 0..3)  | 虚拟联网模式接收唤醒邮箱 x 控制状态信息寄存器      |
| CAN_PN_RWMxI<br>(x = 0..3)   | 虚拟联网模式接收唤醒邮箱 x 标识符寄存器         |
| CAN_PN_RWMxD0<br>(x = 0..3)  | 虚拟联网模式接收唤醒邮箱 x 数据 0 寄存器       |
| CAN_PN_RWMxD1<br>(x = 0..3)  | 虚拟联网模式接收唤醒邮箱 x 数据 1 寄存器       |
| CAN_FDCTL                    | FD 控制寄存器                      |
| CAN_FDBT                     | FD 位时间寄存器                     |
| CAN_CRCCFD                   | 常规帧和 FD 帧 CRC 寄存器             |

### 3.4.2. 外设库函数说明

CAN库函数列表如下表所示：

**表 3-70. CAN 库函数**

| 库函数名称                                  | 库函数描述                 |
|--|-----------------------|
| can_deinit                             | 复位 CAN                |
| can_software_reset                     | 复位 CAN 内部状态机和 CAN 寄存器 |
| can_init                               | CAN 模块初始化             |
| can_struct_para_init                   | 将 CAN 结构体初始化为默认值      |
| can_private_filter_config              | 配置接收 FIFO/邮箱私有过滤器     |
| can_operation_mode_enter               | 进入对应的模式               |
| can_operation_mode_get                 | 获取操作模式                |
| can_inactive_mode_exit                 | 退出暂停模式                |
| can_pn_mode_exit                       | 退出虚拟联网模式              |
| can_fd_config                          | CAN FD 初始化            |
| can_bitrate_switch_enable              | 使能波特率切换功能             |
| can_bitrate_switch_disable             | 禁能波特率切换功能             |
| can_tdc_get                            | 获取传输延迟补偿值             |
| can_tdc_enable                         | 使能传输延迟补偿              |
| can_tdc_disable                        | 禁能传输延迟补偿              |
| can_rx_fifo_config                     | 配置接收 FIFO             |
| can_rx_fifo_filter_table_config        | 配置接收 FIFO 标识符过滤器表     |
| can_rx_fifo_read                       | 读取接收 FIFO 数据          |
| can_rx_fifo_filter_matching_number_get | 获取接收 FIFO 标识符过滤元素匹配序号 |
| can_rx_fifo_clear                      | 清接收 FIFO              |
| can_ram_address_get                    | 获取邮箱 RAM 地址           |
| can_mailbox_config                     | 配置邮箱                  |
| can_mailbox_transmit_abort             | 中止邮箱发送                |
| can_mailbox_transmit_inactive          | 失活发送邮箱                |
| can_mailbox_receive_data_read          | 读取接收邮箱数据              |
| can_mailbox_receive_lock               | 锁定接收邮箱                |
| can_mailbox_receive_unlock             | 解锁接收邮箱                |
| can_mailbox_receive_inactive           | 失活接收邮箱                |
| can_mailbox_code_get                   | 获取邮箱 CODE 值           |
| can_error_counter_config               | 配置错误计数                |
| can_error_counter_get                  | 获取错误计数                |
| can_error_state_get                    | 获取错误状态指示              |
| can_crc_get                            | 获取邮箱 CRC 值            |
| can_pn_mode_config                     | 配置虚拟联网模式参数            |
| can_pn_mode_filter_config              | 配置虚拟联网模式过滤器参数         |

| 库函数名称                             | 库函数描述            |
|-----------------------------------|------------------|
| can_pn_mode_num_of_match_get      | 获取虚拟联网模式下匹配的消息计数 |
| can_pn_mode_data_read             | 获取匹配的消息          |
| can_self_reception_enable         | 使能自接收            |
| can_self_reception_disable        | 禁能自接收            |
| can_transmit_abort_enable         | 使能发送中止功能         |
| can_transmit_abort_disable        | 禁能发送中止功能         |
| can_auto_busoff_recovery_enable   | 使能离线自动恢复模式       |
| can_auto_busoff_recovery_disable  | 禁能离线自动恢复模式       |
| can_time_sync_enable              | 使能时间同步模式         |
| can_time_sync_disable             | 禁能时间同步模式         |
| can_edge_filter_mode_enable       | 使能边沿过滤模式         |
| can_edge_filter_mode_disable      | 禁能边沿过滤模式         |
| can_ped_mode_enable               | 使能协议异常检测模式       |
| can_ped_mode_disable              | 禁能协议异常检测模式       |
| can_arbitration_delay_bits_config | 配置仲裁启动延迟位        |
| can_bsp_mode_config               | 配置位采样模式          |
| can_flag_get                      | 获取 CAN 标志状态      |
| can_flag_clear                    | 清除 CAN 标志状态      |
| can_interrupt_enable              | 使能 CAN 中断        |
| can_interrupt_disable             | 禁能 CAN 中断        |
| can_interrupt_flag_get            | 获取 CAN 中断标志状态    |
| can_interrupt_flag_clear          | 清除 CAN 中断标志状态    |

### 结构体 can\_error\_counter\_struct

表 3-71. 结构体 can\_error\_counter\_struct

| 成员名称                    | 功能描述                        |
|-------------------------|-----------------------------|
| fd_data_phase_rx_errcnt | FD 帧 BRS 位为隐性位时数据阶段的接收错误计数器 |
| fd_data_phase_tx_errcnt | FD 帧 BRS 位为隐性位时数据阶段的发送错误计数器 |
| rx_errcnt               | CAN 协议定义的接收错误计数器            |
| tx_errcnt               | CAN 协议定义的发送错误计数器            |

### 结构体 can\_parameter\_struct

表 3-72. 结构体 can\_parameter\_struct

| 成员名称                    | 功能描述                   |
|-------------------------|------------------------|
| internal_counter_source | 内部计数器时钟源               |
| mb_tx_order             | 邮箱发送顺序                 |
| mb_rx_ide_rtr_type      | 邮箱接收时 IDE 和 RTR 域的过滤类型 |

| 成员名称                           | 功能描述              |
|--------------------------------|-------------------|
| mb_remote_frame                | 远程请求帧存储           |
| self_reception                 | 使能或禁能自接收          |
| mb_tx_abort_enable             | 使能或禁能发送中止         |
| local_priority_enable          | 使能或禁能本地优先级        |
| rx_private_filter_queue_enable | 使能接收私有过滤使能&接收邮箱队列 |
| edge_filter_enable             | 使能边沿过滤            |
| protocol_exception_enable      | 使能协议异常检测          |
| rx_filter_order                | 接收过滤顺序            |
| memory_size                    | 内存大小              |
| mb_public_filter               | 邮箱公有过滤器           |
| prescaler                      | 波特率分频系数           |
| resync_jump_width              | 再同步补偿宽度           |
| prop_time_segment              | 传播时间段             |
| time_segment_1                 | 相位缓冲段 1           |
| time_segment_2                 | 相位缓冲段 2           |

### 结构体 can\_mailbox\_descriptor\_struct

表 3-73. 结构体 can\_mailbox\_descriptor\_struct

| 成员名称       | 功能描述           |
|------------|----------------|
| timestamp  | 来自内部计数器值产生的时间戳 |
| dlc        | 数据字节长度代码       |
| rtr        | 远程传输请求         |
| ide        | 标识符扩展位         |
| srr        | 替代远程请求         |
| code       | 邮箱代码           |
| esi        | 错误状态指示         |
| brs        | 位速率切换          |
| fdf        | FD 格式指示        |
| id         | 标识符            |
| prio       | 本地优先级          |
| data       | 数据             |
| data_bytes | 数据字节           |
| padding    | FD 模式填充值       |

### 结构体 can\_rx\_fifo\_struct

表 3-74. 结构体 can\_rx\_fifo\_struct

| 成员名称      | 功能描述           |
|-----------|----------------|
| timestamp | 来自内部计数器值产生的时间戳 |



| 成员名称    | 功能描述        |
|---------|-------------|
| dlc     | 数据字节长度代码    |
| rtr     | 远程传输请求      |
| ide     | 标识符扩展位      |
| srr     | 替代远程请求      |
| idhit   | 标识符过滤元素匹配序号 |
| id      | 标识符         |
| data[2] | FIFO 数据     |

### 结构体 can\_fd\_parameter\_struct

表 3-75. 结构体 can\_fd\_parameter\_struct

| 成员名称                  | 功能描述             |
|-----------------------|------------------|
| iso_can_fd_enable     | 使能 ISO CAN FD 协议 |
| bitrate_switch_enable | 使能数据阶段波特率切换      |
| mailbox_data_size     | 邮箱数据大小           |
| tdc_enable            | 传输延迟补偿使能         |
| tdc_offset            | 传输延迟补偿偏置         |
| prescaler             | 波特率分频系数          |
| resync_jump_width     | 再同步补偿宽度          |
| prop_time_segment     | 传播时间段            |
| time_segment_1        | 相位缓冲段 1          |
| time_segment_2        | 相位缓冲段 2          |

### 结构体 can\_rx\_fifo\_id\_filter\_struct

表 3-76. 结构体 can\_rx\_fifo\_id\_filter\_struct

| 成员名称           | 功能描述               |
|----------------|--------------------|
| remote_frame   | 期望是否接收匹配的远程帧到 FIFO |
| extended_frame | 期望是否接收匹配的扩展帧到 FIFO |
| id             | 期望的标识符             |

### 结构体 can\_fifo\_parameter\_struct

表 3-77. 结构体 can\_fifo\_parameter\_struct

| 成员名称                     | 功能描述              |
|--------------------------|-------------------|
| dma_enable               | 使能 DMA            |
| filter_format_and_number | FIFO 标识符过滤元素格式和数量 |
| fifo_public_filter       | FIFO 公有过滤器        |

### 结构体 `can_pn_mode_filter_struct`

表 3-78. 结构体 `can_pn_mode_filter_struct`

| 成员名称                            | 功能描述       |
|---------------------------------|------------|
| <code>remote_frame</code>       | 期望 RTR     |
| <code>extended_frame</code>     | 期望 IDE     |
| <code>id</code>                 | 期望 ID      |
| <code>dlc_high_threshold</code> | 期望 DLC 上限值 |
| <code>dlc_low_threshold</code>  | 期望 DLC 下限值 |
| <code>payload[2]</code>         | 期望数据       |

### 结构体 `can_pn_mode_config_struct`

表 3-79. 结构体 `can_pn_mode_config_struct`

| 成员名称                       | 功能描述           |
|----------------------------|----------------|
| <code>timeout_int</code>   | 使能或禁能超时唤醒中断    |
| <code>match_int</code>     | 使能或禁能匹配唤醒中断    |
| <code>num_matches</code>   | 设置消息匹配次数       |
| <code>match_timeout</code> | 设置超时唤醒时间值      |
| <code>frame_filter</code>  | 设置帧的过滤类型       |
| <code>id_filter</code>     | 设置 ID 域的过滤类型   |
| <code>data_filter</code>   | 设置 DATA 域的过滤类型 |

### 结构体 `can_crc_struct`

表 3-80. 结构体 `can_crc_struct`

| 成员名称  | 功能描述                                    |
|---|---|
| <code>classical_frm_mb_number</code>          | 发送了 CRC 值为 CRCTC[14:0]的邮箱的编号            |
| <code>classical_frm_transmitted_crc</code>    | 最新发送的常规帧的 CRC 计算值                       |
| <code>classical_fd_frm_mb_number</code>       | 发送常规帧或者 FD 帧时, CRC 值为 CRCTC[20:0]的邮箱的编号 |
| <code>classical_fd_frm_transmitted_crc</code> | 发送的常规帧 / FD 帧的 CRC 计算值                  |

### 枚举类型 `can_interrupt_enum`

表 3-81. 枚举类型 `can_interrupt_enum`

| 成员名称                            | 功能描述      |
|---------------------------------|-----------|
| <code>CAN_INT_RX_WARNING</code> | Rx 错误警告中断 |
| <code>CAN_INT_TX_WARNING</code> | Tx 错误警告中断 |

| 成员名称                    | 功能描述             |
|-------------------------|------------------|
| CAN_INT_ERR_SUMMARY     | 错误汇总中断           |
| CAN_INT_BUSOFF          | 离线中断             |
| CAN_INT_BUSOFF_RECOVERY | 离线恢复中断           |
| CAN_INT_ERR_SUMMARY_FD  | FD 帧数据位时间的错误汇总中断 |
| CAN_INT_MB0             | 邮箱 0 成功发送或接收帧中断  |
| CAN_INT_MB1             | 邮箱 1 成功发送或接收帧中断  |
| CAN_INT_MB2             | 邮箱 2 成功发送或接收帧中断  |
| CAN_INT_MB3             | 邮箱 3 成功发送或接收帧中断  |
| CAN_INT_MB4             | 邮箱 4 成功发送或接收帧中断  |
| CAN_INT_MB5             | 邮箱 5 成功发送或接收帧中断  |
| CAN_INT_MB6             | 邮箱 6 成功发送或接收帧中断  |
| CAN_INT_MB7             | 邮箱 7 成功发送或接收帧中断  |
| CAN_INT_MB8             | 邮箱 8 成功发送或接收帧中断  |
| CAN_INT_MB9             | 邮箱 9 成功发送或接收帧中断  |
| CAN_INT_MB10            | 邮箱 10 成功发送或接收帧中断 |
| CAN_INT_MB11            | 邮箱 11 成功发送或接收帧中断 |
| CAN_INT_MB12            | 邮箱 12 成功发送或接收帧中断 |
| CAN_INT_MB13            | 邮箱 13 成功发送或接收帧中断 |
| CAN_INT_MB14            | 邮箱 14 成功发送或接收帧中断 |
| CAN_INT_MB15            | 邮箱 15 成功发送或接收帧中断 |
| CAN_INT_MB16            | 邮箱 16 成功发送或接收帧中断 |
| CAN_INT_MB17            | 邮箱 17 成功发送或接收帧中断 |
| CAN_INT_MB18            | 邮箱 18 成功发送或接收帧中断 |
| CAN_INT_MB19            | 邮箱 19 成功发送或接收帧中断 |
| CAN_INT_MB20            | 邮箱 20 成功发送或接收帧中断 |
| CAN_INT_MB21            | 邮箱 21 成功发送或接收帧中断 |
| CAN_INT_MB22            | 邮箱 22 成功发送或接收帧中断 |
| CAN_INT_MB23            | 邮箱 23 成功发送或接收帧中断 |
| CAN_INT_MB24            | 邮箱 24 成功发送或接收帧中断 |
| CAN_INT_MB25            | 邮箱 25 成功发送或接收帧中断 |
| CAN_INT_MB26            | 邮箱 26 成功发送或接收帧中断 |
| CAN_INT_MB27            | 邮箱 27 成功发送或接收帧中断 |
| CAN_INT_MB28            | 邮箱 28 成功发送或接收帧中断 |
| CAN_INT_MB29            | 邮箱 29 成功发送或接收帧中断 |
| CAN_INT_MB30            | 邮箱 30 成功发送或接收帧中断 |
| CAN_INT_MB31            | 邮箱 31 成功发送或接收帧中断 |
| CAN_INT_FIFO_AVAILABLE  | Rx FIFO 非空中断     |

| 成员名称                   | 功能描述         |
|------------------------|--------------|
| CAN_INT_FIFO_WARNING   | Rx FIFO 警告中断 |
| CAN_INT_FIFO_OVERFLOW  | Rx FIFO 溢出中断 |
| CAN_INT_WAKEUP_MATCH   | PN 模式匹配唤醒中断  |
| CAN_INT_WAKEUP_TIMEOUT | PN 模式超时唤醒中断  |

### 枚举类型 `can_flag_enum`

表 3-82. 枚举类型 `can_flag_enum`

| 成员名称                       | 功能描述     |
|----------------------------|----------|
| CAN_FLAG_CAN_PN            | 虚拟联网状态   |
| CAN_FLAG_SOFT_RST          | 软件复位状态   |
| CAN_FLAG_ERR_SUMMARY       | 错误汇总     |
| CAN_FLAG_BUSOFF            | 离线状态     |
| CAN_FLAG_RECEIVING         | 接收状态     |
| CAN_FLAG_TRANSMITTING      | 发送状态     |
| CAN_FLAG_IDLE              | 空闲标志     |
| CAN_FLAG_RX_WARNING        | 接收错误警告标志 |
| CAN_FLAG_TX_WARNING        | 发送错误警告标志 |
| CAN_FLAG_STUFF_ERR         | 填充错误状态   |
| CAN_FLAG_FORMAT_ERR        | 格式错误状态   |
| CAN_FLAG_CRC_ERR           | CRC 错误状态 |
| CAN_FLAG_ACK_ERR           | ACK 错误状态 |
| CAN_FLAG_BIT_DOMINANT_ERR  | 位显性错误状态  |
| CAN_FLAG_BIT_RECESSIVE_ERR | 位隐性错误状态  |

| 成员名称                          | 功能描述                          |
|-------------------------------|-------------------------------|
| CAN_FLAG_SYNC_ERR             | 同步标志                          |
| CAN_FLAG_BUSOFF_RECOVERY      | 离线恢复标志                        |
| CAN_FLAG_ERR_SUMMARY_FD       | FD 帧 BRS 位为隐性位时数据阶段的错误汇总标志    |
| CAN_FLAG_ERR_OVERRUN          | 错误溢出状态                        |
| CAN_FLAG_STUFF_ERR_FD         | D 帧 BRS 位为隐性位时数据阶段的填充错误状态     |
| CAN_FLAG_FORMAT_ERR_FD        | FD 帧 BRS 位为隐性位时数据阶段的格式错误状态    |
| CAN_FLAG_CRC_ERR_FD           | FD 帧 BRS 位为隐性位时数据阶段的 CRC 错误状态 |
| CAN_FLAG_BIT_DOMINANT_ERR_FD  | FD 帧 BRS 位为隐性位时数据阶段的位显性错误状态   |
| CAN_FLAG_BIT_RECESSIVE_ERR_FD | FD 帧 BRS 位为隐性位时数据阶段的位隐性错误状态   |
| CAN_FLAG_MB0                  | 邮箱 0 成功发送或接收帧标志               |
| CAN_FLAG_MB1                  | 邮箱 1 成功发送或接收帧标志               |
| CAN_FLAG_MB2                  | 邮箱 2 成功发送或接收帧标志               |
| CAN_FLAG_MB3                  | 邮箱 3 成功发送或接收帧标志               |
| CAN_FLAG_MB4                  | 邮箱 4 成功发送或接收帧标志               |
| CAN_FLAG_MB5                  | 邮箱 5 成功发送或接收帧标志               |
| CAN_FLAG_MB6                  | 邮箱 6 成功发送或接收帧标志               |
| CAN_FLAG_MB7                  | 邮箱 7 成功发送或接收帧标志               |
| CAN_FLAG_MB8                  | 邮箱 8 成功发送或接收帧标志               |
| CAN_FLAG_MB9                  | 邮箱 9 成功发送或接收帧标志               |
| CAN_FLAG_MB10                 | 邮箱 10 成功发送或接收帧标志              |
| CAN_FLAG_MB11                 | 邮箱 11 成功发送或接收帧标志              |
| CAN_FLAG_MB12                 | 邮箱 12 成功发送或接收帧标志              |
| CAN_FLAG_MB13                 | 邮箱 13 成功发送或接收帧标志              |
| CAN_FLAG_MB14                 | 邮箱 14 成功发送或接收帧标志              |
| CAN_FLAG_MB15                 | 邮箱 15 成功发送或接收帧标志              |
| CAN_FLAG_MB16                 | 邮箱 16 成功发送或接收帧标志              |
| CAN_FLAG_MB17                 | 邮箱 17 成功发送或接收帧标志              |
| CAN_FLAG_MB18                 | 邮箱 18 成功发送或接收帧标志              |
| CAN_FLAG_MB19                 | 邮箱 19 成功发送或接收帧标志              |
| CAN_FLAG_MB20                 | 邮箱 20 成功发送或接收帧标志              |
| CAN_FLAG_MB21                 | 邮箱 21 成功发送或接收帧标志              |

| 成员名称                      | 功能描述             |
|---------------------------|------------------|
| CAN_FLAG_MB22             | 邮箱 22 成功发送或接收帧标志 |
| CAN_FLAG_MB23             | 邮箱 23 成功发送或接收帧标志 |
| CAN_FLAG_MB24             | 邮箱 24 成功发送或接收帧标志 |
| CAN_FLAG_MB25             | 邮箱 25 成功发送或接收帧标志 |
| CAN_FLAG_MB26             | 邮箱 26 成功发送或接收帧标志 |
| CAN_FLAG_MB27             | 邮箱 27 成功发送或接收帧标志 |
| CAN_FLAG_MB28             | 邮箱 28 成功发送或接收帧标志 |
| CAN_FLAG_MB29             | 邮箱 29 成功发送或接收帧标志 |
| CAN_FLAG_MB30             | 邮箱 30 成功发送或接收帧标志 |
| CAN_FLAG_MB31             | 邮箱 31 成功发送或接收帧标志 |
| CAN_FLAG_FIFO_AVAILABLE   | Rx FIFO 非空标志     |
| CAN_FLAG_FIFO_WARNING     | Rx FIFO 警告标志     |
| CAN_FLAG_FIFO_OVERFLOW    | Rx FIFO 溢出标志     |
| CAN_FLAG_WAKEUP_MATCH     | PN 模式匹配唤醒标志      |
| CAN_FLAG_WAKEUP_TIMEOUT   | PN 模式超时唤醒标志      |
| CAN_FLAG_TDC_OUT_OF_RANGE | 传输延迟超出补偿范围标志     |

### 枚举类型 can\_interrupt\_flag\_enum

表 3-83. 枚举类型 can\_interrupt\_flag\_enum

| 成员名称                         | 功能描述               |
|------------------------------|--------------------|
| CAN_INT_FLAG_ERR_SUMMARY     | 错误汇总中断标志           |
| CAN_INT_FLAG_BUSOFF          | 离线中断标志             |
| CAN_INT_FLAG_RX_WARNING      | Rx 错误警告中断标志        |
| CAN_INT_FLAG_TX_WARNING      | Tx 错误警告中断标志        |
| CAN_INT_FLAG_BUSOFF_RECOVERY | 离线恢复中断标志           |
| CAN_INT_FLAG_ERR_SUMMARY_FD  | FD 帧数据位时间的错误汇总中断标志 |
| CAN_INT_FLAG_MBO             | 邮箱 0 成功发送或接收帧中断标志  |

| 成员名称                  | 功能描述               |
|-----------------------|--------------------|
| CAN_INT_FLAG_M<br>B1  | 邮箱 1 成功发送或接收帧中断标志  |
| CAN_INT_FLAG_M<br>B2  | 邮箱 2 成功发送或接收帧中断标志  |
| CAN_INT_FLAG_M<br>B3  | 邮箱 3 成功发送或接收帧中断标志  |
| CAN_INT_FLAG_M<br>B4  | 邮箱 4 成功发送或接收帧中断标志  |
| CAN_INT_FLAG_M<br>B5  | 邮箱 5 成功发送或接收帧中断标志  |
| CAN_INT_FLAG_M<br>B6  | 邮箱 6 成功发送或接收帧中断标志  |
| CAN_INT_FLAG_M<br>B7  | 邮箱 7 成功发送或接收帧中断标志  |
| CAN_INT_FLAG_M<br>B8  | 邮箱 8 成功发送或接收帧中断标志  |
| CAN_INT_FLAG_M<br>B9  | 邮箱 9 成功发送或接收帧中断标志  |
| CAN_INT_FLAG_M<br>B10 | 邮箱 10 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B11 | 邮箱 11 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B12 | 邮箱 12 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B13 | 邮箱 13 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B14 | 邮箱 14 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B15 | 邮箱 15 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B16 | 邮箱 16 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B17 | 邮箱 17 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B18 | 邮箱 18 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B19 | 邮箱 19 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B20 | 邮箱 20 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B21 | 邮箱 21 成功发送或接收帧中断标志 |

| 成员名称                            | 功能描述               |
|---------------------------------|--------------------|
| CAN_INT_FLAG_M<br>B22           | 邮箱 22 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B23           | 邮箱 23 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B24           | 邮箱 24 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B25           | 邮箱 25 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B26           | 邮箱 26 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B27           | 邮箱 27 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B28           | 邮箱 28 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B29           | 邮箱 29 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B30           | 邮箱 30 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_M<br>B31           | 邮箱 31 成功发送或接收帧中断标志 |
| CAN_INT_FLAG_FI<br>FO_AVAILABLE | Rx FIFO 非空中断标志     |
| CAN_INT_FLAG_FI<br>FO_WARNING   | Rx FIFO 警告中断标志     |
| CAN_INT_FLAG_FI<br>FO_OVERFLOW  | Rx FIFO 溢出中断标志     |
| CAN_INT_FLAG_W<br>AKEUP_MATCH   | PN 模式匹配唤醒中断标志      |
| CAN_INT_FLAG_W<br>AKEUP_TIMEOUT | PN 模式超时唤醒中断标志      |

### 枚举类型 `can_operation_modes_enum`

表 3-84. 枚举类型 `can_operation_modes_enum`

| 成员名称                         | 功能描述   |
|------------------------------|--------|
| CAN_NORMAL_MO<br>DE          | 正常模式   |
| CAN_MONITOR_M<br>ODE         | 监听模式   |
| CAN_LOOPBACK_<br>SILENT_MODE | 回环静默模式 |
| CAN_INACTIVE_M               | 暂停模式   |



| 成员名称             | 功能描述   |
|------------------|--------|
| ODE              |        |
| CAN_DISABLE_MODE | 禁能模式   |
| CAN_PN_MODE      | 虚拟联网模式 |

### 枚举类型 `can_struct_type_enum`

表 3-85. 枚举类型 `can_struct_type_enum`

| 成员名称                      | 功能描述           |
|---------------------------|----------------|
| CAN_INIT_STRUCT           | CAN 初始化参数结构体   |
| CAN_FD_INIT_STRUCT        | CAN FD 参数结构体   |
| CAN_FIFO_INIT_STRUCT      | CAN FIFO 参数结构体 |
| CAN_PN_MODE_INIT_STRUCT   | 虚拟联网模式参数结构体    |
| CAN_PN_MODE_FILTER_STRUCT | 虚拟联网模式过滤器参数结构体 |

### 枚举类型 `can_error_state_enum`

表 3-86. 枚举类型 `can_error_state_enum`

| 成员名称                    | 功能描述         |
|-------------------------|--------------|
| CAN_ERROR_STATE_ACTIVE  | CAN 处于主动错误状态 |
| CAN_ERROR_STATE_PASSIVE | CAN 处于被动错误状态 |
| CAN_ERROR_STATE_BUS_OFF | CAN 处于离线状态   |

### 函数 `can_deinit`

函数 `can_deinit` 描述见下表：

表 3-87. 函数 `can_deinit`

| 函数名称                     | <code>can_deinit</code>  |
|--------------------------|--|
| 函数原型                     | <code>void can_deinit(uint32_t can_periph);</code>                           |
| 功能描述                     | 复位 CAN   |
| 先决条件                     | -  |
| 被调用函数                    | <code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code> |
| 输入参数 {in}                |  |
| <code>can_periph</code>  | CAN 外设   |
| <code>CANx(x=0,1)</code> | CAN 外设选择   |

| 输出参数{out} |   |
|-----------|---|
| -         | - |
| 返回值       |   |
| -         | - |

例如:

```
/* deinitialize CAN0*/
```

```
can_deinit(CAN0);
```

### 函数 can\_software\_reset

函数can\_software\_reset描述见下表:

表 3-88. 函数 can\_software\_reset

| 函数名称        | can_software_reset                                 |
|-------------|--|
| 函数原型        | ErrStatus can_software_reset(uint32_t can_periph); |
| 功能描述        | 复位 CAN 内部状态机和 CAN 寄存器                              |
| 先决条件        | -  |
| 被调用函数       |  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择   |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| ErrStatus   | ERROR 或 SUCCESS                                    |

例如:

```
ErrStatus err;
```

```
/* reset CAN0 */
```

```
err = can_software_reset(CAN0);
```

### 函数 can\_init

函数can\_init描述见下表:

表 3-89. 函数 can\_init

| 函数名称  | can_init   |
|-------|--|
| 函数原型  | ErrStatus can_init(uint32_t can_periph, can_parameter_struct* can_parameter_init); |
| 功能描述  | CAN 模块初始化  |
| 先决条件  | -  |
| 被调用函数 | -  |

| 输入参数{in}                  |   |
|---------------------------|---|
| <b>can_periph</b>         | CAN 外设  |
| <i>CANx(x=0,1)</i>        | CAN 外设选择  |
| 输入参数{in}                  |   |
| <b>can_parameter_init</b> | 参见 <a href="#">表 3-72. 结构体 can_parameter_struct</a> |
| 输出参数{out}                 |   |
| -                         | -   |
| 返回值                       |   |
| <b>ErrStatus</b>          | ERROR 或 SUCCESS                                     |

例如:

```
can_parameter_struct can_parameter;

ErrStatus err;

.....

/* initialize CAN */

err = can_init(CAN0, &can_parameter);
```

### 函数 can\_struct\_para\_init

函数can\_struct\_para\_init描述见下表:

**表 3-90. 函数 can\_struct\_para\_init**

| 函数名称            | can_struct_para_init  |
|-----------------|---|
| 函数原型            | void can_struct_para_init(can_struct_type_enum type, void* p_struct); |
| 功能描述            | 将 CAN 结构体初始化为默认值  |
| 先决条件            | -   |
| 被调用函数           | -   |
| 输入参数{in}        |   |
| <b>type</b>     | 参见枚举类型 <a href="#">表 3-85. 枚举类型 can_struct_type_enum</a>              |
| 输入参数{in}        |   |
| <b>p_struct</b> | 指向特定的结构体  |
| 输出参数{out}       |   |
| -               | -   |
| 返回值             |   |
| -               | -   |

例如:

```
can_parameter_struct can_parameter;

/* initialize CAN */

can_struct_para_init(CAN_INIT_STRUCT, &can_parameter);
```

函数 `can_private_filter_config`

函数 `can_private_filter_config` 描述见下表:

表 3-91. 函数 `can_private_filter_config`

|                            |   |
|----------------------------|---|
| 函数名称                       | <code>can_private_filter_config</code>  |
| 函数原型                       | <code>void can_private_filter_config(uint32_t can_periph, uint32_t index, uint32_t filter_data);</code> |
| 功能描述                       | 配置接收 FIFO/邮箱私有过滤器   |
| 先决条件                       | -   |
| 被调用函数                      | -   |
| 输入参数{in}                   |   |
| <code>can_periph</code>    | CAN 外设  |
| <code>CANx(x=0,1)</code>   | CAN 外设选择  |
| 输入参数{in}                   |   |
| <code>index</code>         | 邮箱索引  |
| <code>0-31</code>          | 邮箱索引选择  |
| 输入参数{in}                   |   |
| <code>filter_data</code>   | 配置的过滤器数据  |
| <code>0..0xFFFFFFFF</code> | 过滤器数据   |
| 输出参数{out}                  |   |
| -                          | -   |
| 返回值                        |   |
| -                          | -   |

例如:

```
/* configure CAN0 mailbox 0 private filter */
can_private_filter_config(CAN0, 0, 0xFFFFFFFF);
```

函数 `can_operation_mode_enter`

函数 `can_operation_mode_enter` 描述见下表:

表 3-92. 函数 `can_operation_mode_enter`

|                          |  |
|--------------------------|--|
| 函数名称                     | <code>can_operation_mode_enter</code>  |
| 函数原型                     | <code>ErrStatus can_operation_mode_enter(uint32_t can_periph, can_operation_modes_enum mode);</code> |
| 功能描述                     | 进入对应的模式  |
| 先决条件                     | -  |
| 被调用函数                    | -  |
| 输入参数{in}                 |  |
| <code>can_periph</code>  | CAN 外设   |
| <code>CANx(x=0,1)</code> | CAN 外设选择   |
| 输入参数{in}                 |  |

|                  |  |
|------------------|--|
| <b>mode</b>      | 参见枚举类型 <a href="#">表 3-84. 枚举类型 can_operation_modes_enum</a> |
| <b>输出参数{out}</b> |  |
| -                | -  |
| <b>返回值</b>       |  |
| <b>ErrStatus</b> | ERROR 或 SUCCESS  |

例如:

```
ErrStatus err;
```

```
/* CAN0 enter normal mode */
```

```
err = can_operation_mode_enter(CAN0, CAN_NORMAL_MODE);
```

### 函数 can\_operation\_mode\_get

函数can\_operation\_mode\_get描述见下表:

**表 3-93. 函数 can\_operation\_mode\_get**

|                                 |   |
|---------------------------------|---|
| <b>函数名称</b>                     | can_operation_mode_get  |
| <b>函数原型</b>                     | can_operation_modes_enum can_operation_mode_get(uint32_t can_periph); |
| <b>功能描述</b>                     | 获取操作模式  |
| <b>先决条件</b>                     | -   |
| <b>被调用函数</b>                    | -   |
| <b>输入参数{in}</b>                 |   |
| <b>can_periph</b>               | CAN 外设  |
| CANx(x=0,1)                     | CAN 外设选择  |
| <b>输出参数{out}</b>                |   |
| -                               | -   |
| <b>返回值</b>                      |   |
| <b>can_operation_modes_enum</b> | 参见枚举类型 <a href="#">表 3-84. 枚举类型 can_operation_modes_enum</a>          |

例如:

```
can_operation_modes_enum mode;
```

```
/* get CAN0 mode*/
```

```
mode = can_operation_mode_get(CAN0);
```

### 函数 can\_inactive\_mode\_exit

函数can\_inactive\_mode\_exit描述见下表:

**表 3-94. 函数 can\_inactive\_mode\_exit**

|             |  |
|-------------|--|
| <b>函数名称</b> | can_inactive_mode_exit                                 |
| <b>函数原型</b> | ErrStatus can_inactive_mode_exit(uint32_t can_periph); |

|             |                 |
|-------------|-----------------|
| 功能描述        | 退出暂停模式          |
| 先决条件        | -               |
| 被调用函数       | -               |
| 输入参数{in}    |                 |
| can_periph  | CAN 外设          |
| CANx(x=0,1) | CAN 外设选择        |
| 输出参数{out}   |                 |
| -           | -               |
| 返回值         |                 |
| ErrStatus   | ERROR 或 SUCCESS |

例如:

```
ErrStatus err;
```

```
/* CAN0 exit INACTIVE mode */
```

```
err = can_inactive_mode_exit(CAN0);
```

### 函数 can\_pn\_mode\_exit

函数can\_pn\_mode\_exit描述见下表:

表 3-95. 函数 can\_pn\_mode\_exit

|             |  |
|-------------|--|
| 函数名称        | can_pn_mode_exit                                 |
| 函数原型        | ErrStatus can_pn_mode_exit(uint32_t can_periph); |
| 功能描述        | 退出虚拟联网模式   |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择   |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| ErrStatus   | ERROR 或 SUCCESS                                  |

例如:

```
ErrStatus err;
```

```
/* CAN0 exit PN mode */
```

```
err = can_pn_mode_exit(CAN0);
```

### 函数 can\_fd\_config

函数can\_fd\_config描述见下表:

表 3-96. 函数 can\_fd\_config

|                  |   |
|------------------|---|
| 函数名称             | can_fd_config   |
| 函数原型             | void can_fd_config(uint32_t can_periph, can_fd_parameter_struct *can_fd_para_init); |
| 功能描述             | CAN FD 初始化  |
| 先决条件             | -   |
| 被调用函数            | -   |
| 输入参数{in}         |   |
| can_periph       | CAN 外设  |
| CANx(x=0,1)      | CAN 外设选择  |
| 输入参数{in}         |   |
| can_fd_para_init | 参见结构体 <a href="#">表 3-75. 结构体 can_fd_parameter_struct</a>                           |
| 输出参数{out}        |   |
| -                | -   |
| 返回值              |   |
| -                | -   |

例如:

```
can_fd_parameter_struct fd_parameter;
```

```
/* FD parameter configurations */
```

```
.....
```

```
can_fd_config(CAN0, &fd_parameter);
```

### 函数 can\_bitrate\_switch\_enable

函数can\_bitrate\_switch\_enable描述见下表:

表 3-97. 函数 can\_bitrate\_switch\_enable

|             |  |
|-------------|--|
| 函数名称        | can_bitrate_switch_enable                            |
| 函数原型        | void can_bitrate_switch_enable(uint32_t can_periph); |
| 功能描述        | 使能波特率切换功能  |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择   |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如:

```
/* enable CAN0 bit rate switching */
```

```
can_bitrate_switch_enable(CAN0);
```

### 函数 can\_bitrate\_switch\_disable

函数can\_bitrate\_switch\_disable描述见下表：

**表 3-98. 函数 can\_bitrate\_switch\_disable**

|             |   |
|-------------|---|
| 函数名称        | can_bitrate_switch_disable                            |
| 函数原型        | void can_bitrate_switch_disable(uint32_t can_periph); |
| 功能描述        | 禁能波特率切换功能   |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| can_periph  | CAN 外设  |
| CANx(x=0,1) | CAN 外设选择  |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如：

```
/* disable CAN0 bit rate switching */
```

```
can_bitrate_switch_disable(CAN0);
```

### 函数 can\_tdc\_get

函数can\_tdc\_get描述见下表：

**表 3-99. 函数 can\_tdc\_get**

|             |  |
|-------------|--|
| 函数名称        | can_tdc_get                                |
| 函数原型        | uint32_t can_tdc_get(uint32_t can_periph); |
| 功能描述        | 获取传输延迟补偿值                                  |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设                                     |
| CANx(x=0,1) | CAN 外设选择                                   |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| uint32_t    | 0 - 0x3F                                   |

例如：



```
uint32_t tdc;

/* get transmitter delay compensation value */

tdc = can_tdc_get(CAN0);
```

### 函数 can\_tdc\_enable

函数can\_tdc\_enable描述见下表：

表 3-100. 函数 can\_tdc\_enable

|             |   |
|-------------|---|
| 函数名称        | can_tdc_enable                            |
| 函数原型        | void can_tdc_enable(uint32_t can_periph); |
| 功能描述        | 使能传输延迟补偿                                  |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| can_periph  | CAN 外设                                    |
| CANx(x=0,1) | CAN 外设选择                                  |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如：

```
/* enable transmitter delay compensation */

can_tdc_enable(CAN0);
```

### 函数 can\_tdc\_disable

函数can\_tdc\_disable描述见下表：

表 3-101. 函数 can\_tdc\_disable

|             |  |
|-------------|--|
| 函数名称        | can_tdc_disable                            |
| 函数原型        | void can_tdc_disable(uint32_t can_periph); |
| 功能描述        | 禁能传输延迟补偿                                   |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设                                     |
| CANx(x=0,1) | CAN 外设选择                                   |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如:

```
/* disable transmitter delay compensation */
```

```
can_tdc_disable(CAN0);
```

### 函数 can\_rx\_fifo\_config

函数can\_rx\_fifo\_config描述见下表:

表 3-102. 函数 can\_rx\_fifo\_config

|                    |  |
|--------------------|--|
| 函数名称               | can_rx_fifo_config   |
| 函数原型               | void can_rx_fifo_config(uint32_t can_periph, can_fifo_parameter_struct *can_fifo_para_init); |
| 功能描述               | 配置接收 FIFO  |
| 先决条件               | -  |
| 被调用函数              | -  |
| 输入参数{in}           |  |
| can_periph         | CAN 外设   |
| CANx(x=0,1)        | CAN 外设选择   |
| 输入参数{in}           |  |
| can_fifo_para_init | 参见结构体 <a href="#">表 3-77. 结构体 can_fifo_parameter_struct</a>                                  |
| 输出参数{out}          |  |
| -                  | -  |
| 返回值                |  |
| -                  | -  |

例如:

```
can_fifo_parameter_struct fifo_struct;
```

```
/* configure rx FIFO */
```

```
.....
```

```
can_rx_fifo_config(CAN0, &fifo_struct);
```

### 函数 can\_rx\_fifo\_filter\_table\_config

函数can\_rx\_fifo\_filter\_table\_config描述见下表:

表 3-103. 函数 can\_rx\_fifo\_filter\_table\_config

|       |  |
|-------|--|
| 函数名称  | can_rx_fifo_filter_table_config  |
| 函数原型  | void can_rx_fifo_filter_table_config(uint32_t can_periph, can_rx_fifo_id_filter_struct id_filter_table[]); |
| 功能描述  | 配置接收 FIFO 标识符过滤器表  |
| 先决条件  | -  |
| 被调用函数 | -  |

| 输入参数{in}        |  |
|-----------------|--|
| can_periph      | CAN 外设   |
| CANx(x=0,1)     | CAN 外设选择   |
| 输入参数{in}        |  |
| id_filter_table | 参见结构体 <a href="#">表 3-76. 结构体 can_rx_fifo_id_filter_struct</a> |
| 输出参数{out}       |  |
| -               | -  |
| 返回值             |  |
| -               | -  |

例如：

```
can_rx_fifo_id_filter_struct id_filter_table[104];

/* configure rx FIFO filter table */

.....

can_rx_fifo_filter_table_config(CAN0, id_filter_table);
```

### 函数 can\_rx\_fifo\_read

函数can\_rx\_fifo\_read描述见下表：

**表 3-104. 函数 can\_rx\_fifo\_read**

| 函数名称        | can_rx_fifo_read   |
|-------------|--|
| 函数原型        | void can_rx_fifo_read(uint32_t can_periph, can_rx_fifo_struct *rx_fifo); |
| 功能描述        | 读取接收 FIFO 数据   |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择   |
| 输出参数{out}   |  |
| rx_fifo     | 参见结构体 <a href="#">表 3-74. 结构体 can_rx_fifo_struct</a>                     |
| 返回值         |  |
| -           | -  |

例如：

```
can_rx_fifo_struct rx_fifo;

/* read rx FIFO data */

can_rx_fifo_read(CAN0, &rx_fifo);
```

### 函数 can\_rx\_fifo\_filter\_matching\_number\_get

函数can\_rx\_fifo\_filter\_matching\_number\_get描述见下表：

表 3-105. 函数 `can_rx_fifo_filter_matching_number_get`

|                          |  |
|--------------------------|--|
| 函数名称                     | <code>can_rx_fifo_filter_matching_number_get</code>                                |
| 函数原型                     | <code>uint32_t can_rx_fifo_filter_matching_number_get(uint32_t can_periph);</code> |
| 功能描述                     | 获取接收 FIFO 标识符过滤元素匹配序号  |
| 先决条件                     | -  |
| 被调用函数                    | -  |
| 输入参数{in}                 |  |
| <code>can_periph</code>  | CAN 外设   |
| <code>CANx(x=0,1)</code> | CAN 外设选择   |
| 输出参数{out}                |  |
| -                        | -  |
| 返回值                      |  |
| <code>uint32_t</code>    | 0-416  |

例如:

```
uint32_t number;
```

```
/* get rx FIFO filter matching number */
```

```
number = can_rx_fifo_filter_matching_number_get(CAN0);
```

### 函数 `can_rx_fifo_clear`

函数 `can_rx_fifo_clear` 描述见下表:

表 3-106. 函数 `can_rx_fifo_clear`

|                          |   |
|--------------------------|---|
| 函数名称                     | <code>can_rx_fifo_clear</code>                            |
| 函数原型                     | <code>void can_rx_fifo_clear(uint32_t can_periph);</code> |
| 功能描述                     | 清接收 FIFO  |
| 先决条件                     | -   |
| 被调用函数                    | -   |
| 输入参数{in}                 |   |
| <code>can_periph</code>  | CAN 外设  |
| <code>CANx(x=0,1)</code> | CAN 外设选择  |
| 输出参数{out}                |   |
| -                        | -   |
| 返回值                      |   |
| -                        | -   |

例如:

```
/* clear rx FIFO */
```

```
can_rx_fifo_clear(CAN0);
```

**函数 can\_ram\_address\_get**

函数can\_ram\_address\_get描述见下表:

**表 3-107. 函数 can\_ram\_address\_get**

|             |   |
|-------------|---|
| 函数名称        | can_ram_address_get   |
| 函数原型        | uint32_t* can_ram_address_get(uint32_t can_periph, uint32_t index); |
| 功能描述        | 获取邮箱 RAM 地址   |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| can_periph  | CAN 外设  |
| CANx(x=0,1) | CAN 外设选择  |
| 输入参数{in}    |   |
| index       | 邮箱索引  |
| 0-31        | 邮箱索引选择  |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| uint32_t    | 0-0xFFFFFFFF  |

例如:

```
uint32_t address;
```

```
/* get CAN0 mailbox 0 RAM address* /
```

```
address = can_ram_address_get(CAN0, 0);
```

**函数 can\_mailbox\_config**

函数can\_mailbox\_config描述见下表:

**表 3-108. 函数 can\_mailbox\_config**

|             |  |
|-------------|--|
| 函数名称        | can_mailbox_config   |
| 函数原型        | void can_mailbox_config(uint32_t can_periph, uint32_t index, can_mailbox_descriptor_struct *mdpara); |
| 功能描述        | 配置邮箱   |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择   |
| 输入参数{in}    |  |
| index       | 邮箱索引   |
| 0-31        | 邮箱索引选择   |

| 输入参数{in}  |   |
|-----------|---|
| mdpara    | 参见结构体 <a href="#">表 3-73. 结构体 can_mailbox_descriptor_struct</a> |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

```
can_mailbox_descriptor_struct transmit_message;
```

```
.....
```

```
/* transmit message */
```

```
can_mailbox_config(CAN0, 0, &transmit_message);
```

### 函数 can\_mailbox\_transmit\_abort

函数can\_mailbox\_transmit\_abort描述见下表：

**表 3-109. 函数 can\_mailbox\_transmit\_abort**

| 函数名称        | can_mailbox_transmit_abort  |
|-------------|---|
| 函数原型        | void can_mailbox_transmit_abort(uint32_t can_periph, uint32_t index); |
| 功能描述        | 中止邮箱发送  |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| can_periph  | CAN 外设  |
| CANx(x=0,1) | CAN 外设选择  |
| 输入参数{in}    |   |
| index       | 邮箱索引  |
| 0-31        | 邮箱索引选择  |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如：

```
/* abort mailbox transmit */
```

```
can_mailbox_transmit_abort(CAN0, 0);
```

### 函数 can\_mailbox\_transmit\_inactive

函数can\_mailbox\_transmit\_inactive描述见下表：

表 3-110. 函数 can\_mailbox\_transmit\_inactive

|             |  |
|-------------|--|
| 函数名称        | can_mailbox_transmit_inactive  |
| 函数原型        | void can_mailbox_transmit_inactive(uint32_t can_periph, uint32_t index); |
| 功能描述        | 失活发送邮箱   |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择   |
| 输入参数{in}    |  |
| index       | 邮箱索引   |
| 0-31        | 邮箱索引选择   |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如:

```
/* inactive transmit mailbox */
```

```
can_mailbox_transmit_inactive(CAN0, 0);
```

### 函数 can\_mailbox\_receive\_data\_read

函数can\_mailbox\_receive\_data\_read描述见下表:

表 3-111. 函数 can\_mailbox\_receive\_data\_read

|             |  |
|-------------|--|
| 函数名称        | can_mailbox_receive_data_read  |
| 函数原型        | ErrStatus can_mailbox_receive_data_read(uint32_t can_periph, uint32_t index, can_mailbox_descriptor_struct *mdpara); |
| 功能描述        | 读取接收邮箱数据   |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择   |
| 输入参数{in}    |  |
| index       | 邮箱索引   |
| 0-31        | 邮箱索引选择   |
| 输入参数{in}    |  |
| mdpara      | 参见结构体 <a href="#">表 3-73. 结构体 can_mailbox_descriptor_struct</a>  |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |

|           |                 |
|-----------|-----------------|
| ErrStatus | ERROR 或 SUCCESS |
|-----------|-----------------|

例如:

```
ErrStatus err;
```

```
can_mailbox_descriptor_struct receive_message;
```

```
.....
```

```
/* check the receive message */
```

```
err = can_mailbox_receive_data_read(CAN0, 0, &receive_message);
```

### 函数 can\_mailbox\_receive\_lock

函数can\_mailbox\_receive\_lock描述见下表:

表 3-112. 函数 can\_mailbox\_receive\_lock

|             |   |
|-------------|---|
| 函数名称        | can_mailbox_receive_lock  |
| 函数原型        | void can_mailbox_receive_lock(uint32_t can_periph, uint32_t index); |
| 功能描述        | 锁定接收邮箱  |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| can_periph  | CAN 外设  |
| CANx(x=0,1) | CAN 外设选择  |
| 输入参数{in}    |   |
| index       | 邮箱索引  |
| 0-31        | 邮箱索引选择  |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如:

```
/* lock the receive mailbox 0 */
```

```
can_mailbox_receive_lock(CAN0, 0);
```

### 函数 can\_mailbox\_receive\_unlock

函数can\_mailbox\_receive\_unlock描述见下表:

表 3-113. 函数 can\_mailbox\_receive\_unlock

|      |   |
|------|---|
| 函数名称 | can_mailbox_receive_unlock                            |
| 函数原型 | void can_mailbox_receive_unlock(uint32_t can_periph); |
| 功能描述 | 解锁接收邮箱  |



|             |          |
|-------------|----------|
| 先决条件        | -        |
| 被调用函数       | -        |
| 输入参数{in}    |          |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择 |
| 输出参数{out}   |          |
| -           | -        |
| 返回值         |          |
| -           | -        |

例如：

```
/* unlock the receive mailbox */
```

```
can_mailbox_receive_unlock(CAN0);
```

### 函数 can\_mailbox\_receive\_inactive

函数can\_mailbox\_receive\_inactive描述见下表：

表 3-114. 函数 can\_mailbox\_receive\_inactive

|             |   |
|-------------|---|
| 函数名称        | can_mailbox_receive_inactive  |
| 函数原型        | void can_mailbox_receive_inactive(uint32_t can_periph, uint32_t index); |
| 功能描述        | 失活接收邮箱  |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| can_periph  | CAN 外设  |
| CANx(x=0,1) | CAN 外设选择  |
| 输入参数{in}    |   |
| index       | 邮箱索引  |
| 0-31        | 邮箱索引选择  |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如：

```
/* inactive the receive mailbox */
```

```
can_mailbox_receive_inactive(CAN0, 0);
```

### 函数 can\_mailbox\_code\_get

函数can\_mailbox\_code\_get描述见下表：

表 3-115. 函数 can\_mailbox\_code\_get

|             |   |
|-------------|---|
| 函数名称        | can_mailbox_code_get  |
| 函数原型        | uint32_t can_mailbox_code_get(uint32_t can_periph, uint32_t index); |
| 功能描述        | 获取邮箱 CODE 值   |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| can_periph  | CAN 外设  |
| CANx(x=0,1) | CAN 外设选择  |
| 输入参数{in}    |   |
| index       | 邮箱索引  |
| 0-31        | 邮箱索引选择  |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| uint32_t    | 0-0xF   |

例如:

```
uint32_t code;
```

```
/* get mailbox code value */
```

```
code = can_mailbox_code_get(CAN0, 0);
```

### 函数 can\_error\_counter\_config

函数 can\_error\_counter\_config 描述见下表:

表 3-116. 函数 can\_error\_counter\_config

|               |  |
|---------------|--|
| 函数名称          | can_error_counter_config   |
| 函数原型          | void can_error_counter_config(uint32_t can_periph, can_error_counter_struct *errcnt_struct); |
| 功能描述          | 配置错误计数   |
| 先决条件          | -  |
| 被调用函数         | -  |
| 输入参数{in}      |  |
| can_periph    | CAN 外设   |
| CANx(x=0,1)   | CAN 外设选择   |
| 输入参数{in}      |  |
| errcnt_struct | 参见结构体 <a href="#">表 3-71. 结构体 can_error_counter_struct</a>                                   |
| 输出参数{out}     |  |
| -             | -  |
| 返回值           |  |
| -             | -  |

例如:

```
can_error_counter_struct err_struct;

.....

/* configure error counter */

can_error_counter_config(CAN0, &err_struct);
```

### 函数 can\_error\_counter\_get

函数can\_error\_counter\_get描述见下表:

表 3-117. 函数 can\_error\_counter\_get

| 函数名称          | can_error_counter_get   |
|---------------|---|
| 函数原型          | void can_error_counter_get(uint32_t can_periph, can_error_counter_struct *errcnt_struct); |
| 功能描述          | 获取错误计数  |
| 先决条件          | -   |
| 被调用函数         | -   |
| 输入参数{in}      |   |
| can_periph    | CAN 外设  |
| CANx(x=0,1)   | CAN 外设选择  |
| 输入参数{in}      |   |
| errcnt_struct | 参见结构体 <a href="#">表 3-71. 结构体 can_error_counter_struct</a>                                |
| 输出参数{out}     |   |
| -             | -   |
| 返回值           |   |
| -             | -   |

例如:

```
can_error_counter_struct err_struct;

/* get error count */

can_error_counter_get(CAN0, &err_struct);
```

### 函数 can\_error\_state\_get

函数can\_error\_state\_get描述见下表:

表 3-118. 函数 can\_error\_state\_get

| 函数名称  | can_error_state_get  |
|-------|--|
| 函数原型  | can_error_state_enum can_error_state_get(uint32_t can_periph); |
| 功能描述  | 获取错误状态指示   |
| 先决条件  | -  |
| 被调用函数 | -  |

| 输入参数{in}             |  |
|----------------------|--|
| can_periph           | CAN 外设   |
| CANx(x=0,1)          | CAN 外设选择   |
| 输出参数{out}            |  |
| -                    | -  |
| 返回值                  |  |
| can_error_state_enum | 参见枚举类型 <a href="#">表 3-86. 枚举类型 can_error_state_enum</a> |

例如:

```
can_error_state_enum error_state;
```

```
/* get error state indicator */
```

```
error_state = can_error_state_get(CAN0);
```

### 函数 can\_crc\_get

函数can\_crc\_get描述见下表:

表 3-119. 函数 can\_crc\_get

| 函数名称        | can_crc_get  |
|-------------|--|
| 函数原型        | void can_crc_get(uint32_t can_periph, can_crc_struct *crc_struct); |
| 功能描述        | 获取邮箱 CRC 值   |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择   |
| 返回值         |  |
| crc_struct  | 参见结构体 <a href="#">表 3-80. 结构体 can_crc_struct</a>                   |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如:

```
can_crc_struct crc_struct;
```

```
/* get mailbox CRC value */
```

```
can_crc_get(CAN0, &crc_struct);
```

### 函数 can\_pn\_mode\_config

函数can\_pn\_mode\_config描述见下表:

表 3-120. 函数 can\_pn\_mode\_config

|              |   |
|--------------|---|
| 函数名称         | can_pn_mode_config  |
| 函数原型         | void can_pn_mode_config(uint32_t can_periph,<br>can_pn_mode_config_struct *pnmod_config); |
| 功能描述         | 配置虚拟联网模式参数  |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| can_periph   | CAN 外设  |
| CANx(x=0,1)  | CAN 外设选择  |
| 返回值          |   |
| pnmod_config | 参见结构体 <a href="#">表 3-79. 结构体 can_pn_mode_config_struct</a>                               |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如:

```
can_pn_mode_config_struct pn_struct;

.....

/* configure Pretended Networking mode parameter */

can_pn_mode_config(CAN0, &pn_struct);
```

### 函数 can\_pn\_mode\_filter\_config

函数can\_pn\_mode\_filter\_config描述见下表:

表 3-121. 函数 can\_pn\_mode\_filter\_config

|             |   |
|-------------|---|
| 函数名称        | can_pn_mode_filter_config   |
| 函数原型        | void can_pn_mode_filter_config(uint32_t can_periph,<br>can_pn_mode_filter_struct *expect, can_pn_mode_filter_struct *filter); |
| 功能描述        | 配置虚拟联网模式过滤器参数   |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| can_periph  | CAN 外设  |
| CANx(x=0,1) | CAN 外设选择  |
| 返回值         |   |
| expect      | 参见结构体 <a href="#">表 3-78. 结构体 can_pn_mode_filter_struct</a>   |
| 返回值         |   |
| filter      | 参见结构体 <a href="#">表 3-78. 结构体 can_pn_mode_filter_struct</a>   |
| 输出参数{out}   |   |

|     |   |
|-----|---|
| -   | - |
| 返回值 |   |
| -   | - |

例如:

```
can_pn_mode_filter_struct pn_filter[2];
.....
/* configure pn mode filter */
can_pn_mode_filter_config(CAN0, &pn_filter[0], &pn_filter[1]);
```

### 函数 can\_pn\_mode\_num\_of\_match\_get

函数can\_pn\_mode\_num\_of\_match\_get描述见下表:

表 3-122. 函数 can\_pn\_mode\_num\_of\_match\_get

|             |  |
|-------------|--|
| 函数名称        | can_pn_mode_num_of_match_get                               |
| 函数原型        | int32_t can_pn_mode_num_of_match_get(uint32_t can_periph); |
| 功能描述        | 获取虚拟联网模式下匹配的消息计数   |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择   |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| int32_t     | 0-255 或-1  |

例如:

```
int32_t counter;
/* get matching message counter of Pretended Networking mode */
counter = can_pn_mode_num_of_match_get(CAN0);
```

### 函数 can\_pn\_mode\_data\_read

函数can\_pn\_mode\_data\_read描述见下表:

表 3-123. 函数 can\_pn\_mode\_data\_read

|      |   |
|------|---|
| 函数名称 | can_pn_mode_data_read   |
| 函数原型 | void can_pn_mode_data_read(uint32_t can_periph, uint32_t index, can_mailbox_descriptor_struct *mdpara); |
| 功能描述 | 获取匹配的消息   |

|             |   |
|-------------|---|
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| can_periph  | CAN 外设  |
| CANx(x=0,1) | CAN 外设选择  |
| 输入参数{in}    |   |
| index       | 邮箱索引  |
| 0-31        | 邮箱索引选择  |
| 输入参数{in}    |   |
| mdpara      | 参见结构体 <a href="#">表 3-73. 结构体 can_mailbox_descriptor_struct</a> |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如：

```
can_mailbox_descriptor_struct mb_para;
```

```
/* get matching message */
```

```
can_pn_mode_data_read(CAN0, 0, &mb_para);
```

### 函数 can\_self\_reception\_enable

函数can\_self\_reception\_enable描述见下表：

表 3-124. 函数 can\_self\_reception\_enable

|             |  |
|-------------|--|
| 函数名称        | can_self_reception_enable                            |
| 函数原型        | void can_self_reception_enable(uint32_t can_periph); |
| 功能描述        | 使能自接收  |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择   |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如：

```
/* enable self reception */
```

```
can_self_reception_enable(CAN0);
```

**函数 can\_self\_reception\_disable**

函数can\_self\_reception\_disable描述见下表：

**表 3-125. 函数 can\_self\_reception\_disable**

|             |   |
|-------------|---|
| 函数名称        | can_self_reception_disable                            |
| 函数原型        | void can_self_reception_disable(uint32_t can_periph); |
| 功能描述        | 禁能自接收   |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| can_periph  | CAN 外设  |
| CANx(x=0,1) | CAN 外设选择  |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如：

```
/* disable self reception */
```

```
can_self_reception_disable(CAN0);
```

**函数 can\_transmit\_abort\_enable**

函数can\_transmit\_abort\_enable描述见下表：

**表 3-126. 函数 can\_transmit\_abort\_enable**

|             |  |
|-------------|--|
| 函数名称        | can_transmit_abort_enable                            |
| 函数原型        | void can_transmit_abort_enable(uint32_t can_periph); |
| 功能描述        | 使能发送中止功能   |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择   |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如：

```
/* enable transmit abort */
```

```
can_transmit_abort_enable(CAN0);
```



**函数 can\_transmit\_abort\_disable**

函数can\_transmit\_abort\_disable描述见下表：

**表 3-127. 函数 can\_transmit\_abort\_disable**

|             |   |
|-------------|---|
| 函数名称        | can_transmit_abort_disable                            |
| 函数原型        | void can_transmit_abort_disable(uint32_t can_periph); |
| 功能描述        | 禁能发送中止功能  |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| can_periph  | CAN 外设  |
| CANx(x=0,1) | CAN 外设选择  |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如：

```
/* disable transmit abort */
```

```
can_transmit_abort_disable(CAN0);
```

**函数 can\_auto\_busoff\_recovery\_enable**

函数can\_auto\_busoff\_recovery\_enable描述见下表：

**表 3-128. 函数 can\_auto\_busoff\_recovery\_enable**

|             |  |
|-------------|--|
| 函数名称        | can_auto_busoff_recovery_enable                            |
| 函数原型        | void can_auto_busoff_recovery_enable(uint32_t can_periph); |
| 功能描述        | 使能离线自动恢复模式   |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择   |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如：

```
/* enable auto bus off recovery mode */
```

```
can_auto_busoff_recovery_enable(CAN0);
```

**函数 can\_auto\_busoff\_recovery\_disable**

函数can\_auto\_busoff\_recovery\_disable描述见下表：

**表 3-129. 函数 can\_auto\_busoff\_recovery\_disable**

|             |   |
|-------------|---|
| 函数名称        | can_auto_busoff_recovery_disable                            |
| 函数原型        | void can_auto_busoff_recovery_disable(uint32_t can_periph); |
| 功能描述        | 禁能离线自动恢复模式  |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| can_periph  | CAN 外设  |
| CANx(x=0,1) | CAN 外设选择  |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如：

```
/* disable auto bus off recovery mode */
can_auto_busoff_recovery_disable(CAN0);
```

**函数 can\_time\_sync\_enable**

函数can\_time\_sync\_enable描述见下表：

**表 3-130. 函数 can\_time\_sync\_enable**

|             |   |
|-------------|---|
| 函数名称        | can_time_sync_enable                            |
| 函数原型        | void can_time_sync_enable(uint32_t can_periph); |
| 功能描述        | 使能时间同步模式  |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| can_periph  | CAN 外设  |
| CANx(x=0,1) | CAN 外设选择  |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如：

```
/* enable time sync mode */
can_time_sync_enable(CAN0);
```

**函数 can\_time\_sync\_disable**

函数can\_time\_sync\_disable描述见下表:

**表 3-131. 函数 can\_time\_sync\_disable**

|             |  |
|-------------|--|
| 函数名称        | can_time_sync_disable                            |
| 函数原型        | void can_time_sync_disable(uint32_t can_periph); |
| 功能描述        | 禁能时间同步模式   |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择   |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如:

```
/* disable time sync mode */
can_time_sync_disable(CAN0);
```

**函数 can\_edge\_filter\_mode\_enable**

函数can\_edge\_filter\_mode\_enable描述见下表:

**表 3-132. 函数 can\_edge\_filter\_mode\_enable**

|             |  |
|-------------|--|
| 函数名称        | can_edge_filter_mode_enable                            |
| 函数原型        | void can_edge_filter_mode_enable(uint32_t can_periph); |
| 功能描述        | 使能边沿过滤模式   |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择   |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如:

```
/* enable edge filter mode */
can_edge_filter_mode_enable(CAN0);
```

**函数 can\_edge\_filter\_mode\_disable**

函数can\_edge\_filter\_mode\_disable描述见下表：

**表 3-133. 函数 can\_edge\_filter\_mode\_disable**

|             |   |
|-------------|---|
| 函数名称        | can_edge_filter_mode_disable                            |
| 函数原型        | void can_edge_filter_mode_disable(uint32_t can_periph); |
| 功能描述        | 禁能边沿过滤模式  |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| can_periph  | CAN 外设  |
| CANx(x=0,1) | CAN 外设选择  |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如：

```
/* disable edge filter mode */
can_edge_filter_mode_disable(CAN0);
```

**函数 can\_ped\_mode\_enable**

函数can\_ped\_mode\_enable描述见下表：

**表 3-134. 函数 can\_ped\_mode\_enable**

|             |  |
|-------------|--|
| 函数名称        | can_ped_mode_enable                            |
| 函数原型        | void can_ped_mode_enable(uint32_t can_periph); |
| 功能描述        | 使能协议异常检测模式                                     |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择                                       |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如：

```
/* enable protocol exception detection mode */
can_ped_mode_enable(CAN0);
```

**函数 can\_ped\_mode\_disable**

函数can\_ped\_mode\_disable描述见下表:

**表 3-135. 函数 can\_ped\_mode\_disable**

|             |   |
|-------------|---|
| 函数名称        | can_ped_mode_disable                            |
| 函数原型        | void can_ped_mode_disable(uint32_t can_periph); |
| 功能描述        | 禁能协议异常检测模式                                      |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| can_periph  | CAN 外设  |
| CANx(x=0,1) | CAN 外设选择  |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如:

```
/* disable protocol exception detection mode */
```

```
can_ped_mode_disable(CAN0);
```

**函数 can\_arbitration\_delay\_bits\_config**

函数can\_arbitration\_delay\_bits\_config描述见下表:

**表 3-136. 函数 can\_arbitration\_delay\_bits\_config**

|             |   |
|-------------|---|
| 函数名称        | can_arbitration_delay_bits_config   |
| 函数原型        | void can_arbitration_delay_bits_config(uint32_t can_periph, uint32_t delay_bits); |
| 功能描述        | 配置仲裁启动延迟位   |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| can_periph  | CAN 外设  |
| CANx(x=0,1) | CAN 外设选择  |
| 输入参数{in}    |   |
| delay_bits  | 延迟位   |
| 0-31        | 延迟位选择   |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如：

```
/* configure arbitration delay bits */
can_arbitration_delay_bits_config(CAN0, 2);
```

### 函数 can\_bsp\_mode\_config

函数can\_bsp\_mode\_config描述见下表：

表 3-137. 函数 can\_bsp\_mode\_config

|                                |  |
|--------------------------------|--|
| 函数名称                           | can_bsp_mode_config  |
| 函数原型                           | void can_bsp_mode_config(uint32_t can_periph, uint32_t sampling_mode); |
| 功能描述                           | 配置位采样模式  |
| 先决条件                           | -  |
| 被调用函数                          | -  |
| 输入参数{in}                       |  |
| can_periph                     | CAN 外设   |
| CANx(x=0,1)                    | CAN 外设选择   |
| 输入参数{in}                       |  |
| sampling_mode                  | 位采样模式  |
| CAN_BSP_MODE_ON<br>E_SAMPLE    | 对接收的位只采样一次   |
| CAN_BSP_MODE_TR<br>HEE_SAMPLES | 对接收的位采样 3 次  |
| 输出参数{out}                      |  |
| -                              | -  |
| 返回值                            |  |
| -                              | -  |

例如：

```
/* configure bit sampling mode */
can_bsp_mode_config(CAN0, CAN_BSP_MODE_ONE_SAMPLE);
```

### 函数 can\_flag\_get

函数can\_flag\_get描述见下表：

表 3-138. 函数 can\_flag\_get

|          |   |
|----------|---|
| 函数名称     | can_flag_get  |
| 函数原型     | FlagStatus can_flag_get(uint32_t can_periph, can_flag_enum flag); |
| 功能描述     | 获取 CAN 标志状态   |
| 先决条件     | -   |
| 被调用函数    | -   |
| 输入参数{in} |   |

|                   |   |
|-------------------|---|
| <b>can_periph</b> | CAN 外设  |
| $CANx(x=0,1)$     | CAN 外设选择  |
| 输入参数{in}          |   |
| <b>flag</b>       | 参见枚举类型 <a href="#">表 3-82. 枚举类型 can_flag_enum</a> |
| 输出参数{out}         |   |
| -                 | -   |
| 返回值               |   |
| <b>FlagStatus</b> | SET 或 RESET                                       |

例如:

FlagStatus flag;

/\* get CAN fifo available flag \*/

flag = can\_flag\_get(CAN0, CAN\_FLAG\_FIFO\_AVAILABLE);

### 函数 can\_flag\_clear

函数can\_flag\_clear描述见下表:

**表 3-139. 函数 can\_flag\_clear**

|                   |   |
|-------------------|---|
| 函数名称              | can_flag_clear  |
| 函数原型              | void can_flag_clear(uint32_t can_periph, can_flag_enum flag); |
| 功能描述              | 清除 CAN 标志状态   |
| 先决条件              | -   |
| 被调用函数             | -   |
| 输入参数{in}          |   |
| <b>can_periph</b> | CAN 外设  |
| $CANx(x=0,1)$     | CAN 外设选择  |
| 输入参数{in}          |   |
| <b>flag</b>       | 参见枚举类型 <a href="#">表 3-82. 枚举类型 can_flag_enum</a>             |
| 输出参数{out}         |   |
| -                 | -   |
| 返回值               |   |
| -                 | -   |

例如:

/\* clear CAN fifo available flag \*/

can\_flag\_clear(CAN0, CAN\_FLAG\_FIFO\_AVAILABLE);

### 函数 can\_interrupt\_enable

函数can\_interrupt\_enable描述见下表:

表 3-140. 函数 can\_interrupt\_enable

|             |   |
|-------------|---|
| 函数名称        | can_interrupt_enable  |
| 函数原型        | void can_interrupt_enable(uint32_t can_periph, can_interrupt_enum interrupt); |
| 功能描述        | 使能 CAN 中断   |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| can_periph  | CAN 外设  |
| CANx(x=0,1) | CAN 外设选择  |
| 输入参数{in}    |   |
| interrupt   | 参见枚举类型 <a href="#">表 3-81. 枚举类型 can_interrupt_enum</a>                        |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如:

```
/* enable CAN bus off interrupt */
can_interrupt_enable(CAN0, CAN_INT_BUSOFF);
```

### 函数 can\_interrupt\_disable

函数 can\_interrupt\_disable 描述见下表:

表 3-141. 函数 can\_interrupt\_disable

|             |  |
|-------------|--|
| 函数名称        | can_interrupt_disable  |
| 函数原型        | void can_interrupt_disable(uint32_t can_periph, can_interrupt_enum interrupt); |
| 功能描述        | 禁能 CAN 中断  |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择   |
| 输入参数{in}    |  |
| interrupt   | 参见枚举类型 <a href="#">表 3-81. 枚举类型 can_interrupt_enum</a>                         |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如:



```
/* disable CAN bus off interrupt */
```

```
can_interrupt_disable(CAN0, CAN_INT_BUSOFF);
```

### 函数 can\_interrupt\_flag\_get

函数can\_interrupt\_flag\_get描述见下表:

表 3-142. 函数 can\_interrupt\_flag\_get

|             |  |
|-------------|--|
| 函数名称        | can_interrupt_flag_get   |
| 函数原型        | FlagStatus can_interrupt_flag_get(uint32_t can_periph,<br>can_interrupt_flag_enum int_flag); |
| 功能描述        | 获取 CAN 中断标志状态  |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择   |
| 输入参数{in}    |  |
| int_flag    | 参见枚举类型 <a href="#">表 3-83. 枚举类型 can_interrupt_flag_enum</a>                                  |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| FlagStatus  | SET 或 RESET  |

例如:

```
FlagStatus int_flag;
```

```
/* get CAN fifo available interrupt flag */
```

```
int_flag = can_interrupt_flag_get(CAN0, CAN_INT_FLAG_FIFO_AVAILABLE);
```

### 函数 can\_interrupt\_flag\_clear

函数can\_interrupt\_flag\_clear描述见下表:

表 3-143. 函数 can\_interrupt\_flag\_clear

|             |  |
|-------------|--|
| 函数名称        | can_interrupt_flag_clear   |
| 函数原型        | void can_interrupt_flag_clear(uint32_t can_periph, can_interrupt_flag_enum<br>int_flag); |
| 功能描述        | 清除 CAN 中断标志状态  |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| can_periph  | CAN 外设   |
| CANx(x=0,1) | CAN 外设选择   |

| 输入参数{in}  |   |
|-----------|---|
| int_flag  | 参见枚举类型 <a href="#">表 3-83. 枚举类型 can_interrupt_flag_enum</a> |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

```
/* clear CAN fifo available interrupt flag */
can_interrupt_flag_clear(CAN0, CAN_INT_FLAG_FIFO_AVAILABLE);
```

## 3.5. CMP

通用比较器可独立工作，其输出端可用于I/O口，也可和定时器结合使用。比较器可通过模拟信号将MCU从低功耗模式中唤醒，在一定的条件下，可将模拟信号作为TIMER的触发源，结合DAC和TIMER的PWM输出，可以实现电流控制。章节[3.5.1](#)描述了CMP的寄存器列表，章节[3.5.2](#)对CMP库函数进行说明。

### 3.5.1. 外设寄存器说明

CMP寄存器列表如下表所示：

表 3-144. CMP 寄存器

| 寄存器名称   | 寄存器描述       |
|---------|-------------|
| CMPX_CS | CMP0控制状态寄存器 |

### 3.5.2. 外设库函数说明

CMP库函数列表如下表所示：

表 3-145. CMP 库函数

| 库函数名称                         | 库函数描述      |
|-------------------------------|------------|
| cmp_deinit                    | 复位CMP      |
| cmp_mode_init                 | CMP工作模式初始化 |
| cmp_noninverting_input_select | CMP正相输入选择  |
| cmp_output_init               | CMP输出初始化   |
| cmp_blanking_init             | CMP消隐功能初始化 |
| cmp_enable                    | 使能CMP      |
| cmp_disable                   | 禁能CMP      |
| cmp_lock_enable               | 锁定CMP      |
| cmp_voltage_scaler_enable     | 使能带隙标量     |
| cmp_voltage_scaler_disable    | 禁能带隙标量     |
| cmp_scaler_bridge_enable      | 使能标量桥接     |

| 库函数名称                     | 库函数描述     |
|---------------------------|-----------|
| cmp_scaler_bridge_disable | 禁能标量桥接    |
| cmp_output_level_get      | 获取CMP输出状态 |

### 枚举类型 **cmp\_enum**

表 3-146. 枚举类型 **cmp\_enum**

| 成员名称 | 功能描述 |
|------|------|
| CMP0 | 比较器0 |

### 函数 **cmp\_deinit**

函数cmp\_deinit描述见下表：

表 3-147. 函数 **cmp\_deinit**

| 函数名称       | cmp_deinit   |
|------------|--|
| 函数原型       | void cmp_deinit(cmp_enum cmp_periph);              |
| 功能描述       | 复位CMP  |
| 先决条件       | -  |
| 被调用函数      | rcu_periph_reset_enable / rcu_periph_reset_disable |
| 输入参数{in}   |  |
| cmp_periph | 参考枚举 <a href="#">表 3-146. 枚举类型cmp_enum</a>         |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如：

```
/* deinitialize CMP0 */
```

```
cmp_deinit(CMP0);
```

### 函数 **cmp\_mode\_init**

函数cmp\_mode\_init描述见下表：

表 3-148. 函数 **cmp\_mode\_init**

| 函数名称       | cmp_mode_init   |
|------------|---|
| 函数原型       | void cmp_mode_init(cmp_enum cmp_periph, uint32_t operating_mode, uint32_t inverting_input, uint32_t output_hysteresis); |
| 功能描述       | CMP工作模式初始化  |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| cmp_periph | 参考枚举 <a href="#">表 3-146. 枚举类型cmp_enum</a>  |

| 输入参数{in}  |                       |
|---|-----------------------|
| <b>operating_mode</b>                           | 速度和功耗运行模式             |
| <i>CMP_MODE_HIGH<br/>SPEED</i>                  | 高速/全功耗                |
| <i>CMP_MODE_MIDD<br/>LESPEED</i>                | 中速/中功耗                |
| <i>CMP_MODE_LOW<br/>SPEED</i>                   | 低速/低功耗                |
| 输入参数{in}  |                       |
| <b>inverting_input</b>                          | 反相输入源选择               |
| <i>CMP_INVERTING_I<br/>NPUT_1_4VREFIN<br/>T</i> | $VREFINT * 1/4$ 作为输入源 |
| <i>CMP_INVERTING_I<br/>NPUT_1_2VREFIN<br/>T</i> | $VREFINT * 1/2$ 作为输入源 |
| <i>CMP_INVERTING_I<br/>NPUT_3_4VREFIN<br/>T</i> | $VREFINT * 3/4$ 作为输入源 |
| <i>CMP_INVERTING_I<br/>NPUT_VREFINT</i>         | VREFINT作为输入源          |
| <i>CMP_INVERTING_I<br/>NPUT_DAC0_OUT0</i>       | PA4（DAC）作为输入源         |
| <i>CMP_INVERTING_I<br/>NPUT_PC11</i>            | PC11作为输入源             |
| <i>CMP_INVERTING_I<br/>NPUT_PC10</i>            | PC10作为输入源             |
| <i>CMP_INVERTING_I<br/>NPUT_PB8</i>             | PB8作为输入源              |
| <i>CMP_INVERTING_I<br/>NPUT_PA0</i>             | PA0作为输入源              |
| <i>CMP_INVERTING_I<br/>NPUT_PA3</i>             | PA3作为输入源              |
| <i>CMP_INVERTING_I<br/>NPUT_PA4</i>             | PA4作为输入源              |
| <i>CMP_INVERTING_I<br/>NPUT_PA5</i>             | PA5作为输入源              |
| <i>CMP_INVERTING_I<br/>NPUT_PA6</i>             | PA6作为输入源              |
| 输入参数{in}  |                       |
| <b>output_hysteresis</b>                        | 迟滞水平                  |
| <i>CMP_HYSTERESIS<br/>_NO</i>                   | 无迟滞                   |

|                                    |     |
|------------------------------------|-----|
| <code>CMP_HYSTERESIS_LOW</code>    | 低迟滞 |
| <code>CMP_HYSTERESIS_MIDDLE</code> | 中迟滞 |
| <code>CMP_HYSTERESIS_HIGH</code>   | 高迟滞 |
| 输出参数{out}                          |     |
| -                                  | -   |
| 返回值                                |     |
| -                                  | -   |

例如：

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

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

### 函数 `cmp_noninverting_input_select`

函数 `cmp_noninverting_input_select` 描述见下表：

表 3-149. 函数 `cmp_noninverting_input_select`

|                                       |  |
|---------------------------------------|--|
| 函数名称                                  | <code>cmp_noninverting_input_select</code>   |
| 函数原型                                  | <code>void cmp_noninverting_input_select(cmp_enum cmp_periph, uint32_t noninverting_input);</code> |
| 功能描述                                  | CMP正相输入选择  |
| 先决条件                                  | -  |
| 被调用函数                                 | -  |
| 输入参数{in}                              |  |
| <code>cmp_periph</code>               | 参考枚举 <a href="#">表 3-146. 枚举类型 <code>cmp_enum</code></a>   |
| 输入参数{in}                              |  |
| <code>noninverting_input</code>       | 正相输入源选择  |
| <code>CMP_INVERTING_INPUT_PC11</code> | PC11作为输入源  |
| <code>CMP_INVERTING_INPUT_PC10</code> | PC10作为输入源  |
| <code>CMP_INVERTING_INPUT_PB8</code>  | PB8作为输入源   |
| <code>CMP_INVERTING_INPUT_PA0</code>  | PA0作为输入源   |
| <code>CMP_INVERTING_INPUT_PA3</code>  | PA3作为输入源   |
| <code>CMP_INVERTING_INPUT_PA4</code>  | PA4作为输入源   |

|                                      |          |
|--------------------------------------|----------|
| <code>CMP_INVERTING_INPUT_PA5</code> | PA5作为输入源 |
| <code>CMP_INVERTING_INPUT_PA6</code> | PA6作为输入源 |
| 输出参数{out}                            |          |
| -                                    | -        |
| 返回值                                  |          |
| -                                    | -        |

例如：

```
/* selecte the noninverting input for CMP0 */
```

```
cmp_noninverting_input_select (CMP0, CMP_INVERTING_INPUT_PC11);
```

### 函数 `cmp_output_init`

函数`cmp_output_init`描述见下表：

表 3-150. 函数 `cmp_output_init`

|  |   |
|--|---|
| 函数名称   | <code>cmp_output_init</code>  |
| 函数原型   | <code>void cmp_output_init(cmp_enum cmp_periph , uint32_t output_selection, uint32_t output_polarity);</code> |
| 功能描述   | CMP输出初始化  |
| 先决条件   | -   |
| 被调用函数  | -   |
| 输入参数{in}                                     |   |
| <code>cmp_periph</code>                      | 参考枚举 <a href="#">表 3-146. 枚举类型 <code>cmp_enum</code></a>  |
| 输入参数{in}                                     |   |
| <code>output_selection</code>                | CMP输出选择   |
| <code>CMP_OUTPUT_NONE</code>                 | CMP输出无选择  |
| <code>CMP_OUTPUT_TIMER0_IC0</code>           | CMP输出选择TIMER0_CH0输入捕获   |
| <code>CMP_OUTPUT_TIMER7_IC0</code>           | CMP输出选择TIMER7_CH0输入捕获   |
| 输入参数{in}                                     |   |
| <code>output_polarity</code>                 | CMP输出极性   |
| <code>CMP_OUTPUT_POLARITY_INVERTED</code>    | 输出反相  |
| <code>CMP_OUTPUT_POLARITY_NONINVERTED</code> | 输出正相  |
| 输出参数{out}                                    |   |
| -  | -   |

| 返回值 |   |
|-----|---|
| -   | - |

例如:

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

```
cmp_output_init(CMP0, CMP_OUTPUT_POLARITY_NOINVERTED);
```

### 函数 cmp\_blanking\_init

函数cmp\_blanking\_init描述见下表:

表 3-151. 函数 cmp\_blanking\_init

|                           |  |
|---------------------------|--|
| 函数名称                      | cmp_blanking_init  |
| 函数原型                      | void cmp_blanking_init(cmp_enum cmp_periph, uint32_t blanking_source_selection); |
| 功能描述                      | CMP消隐功能初始化   |
| 先决条件                      | -  |
| 被调用函数                     | -  |
| 输入参数{in}                  |  |
| cmp_periph                | 参考枚举 <a href="#">表 3-146. 枚举类型cmp_enum</a>                                       |
| 输入参数{in}                  |  |
| blanking_source_selection | 消隐源选择  |
| CMP_BLANKING_NONE         | 无消隐  |
| CMP_BLANKING_TIMER0_OC1   | TIMER0_CH1输出比较信号为消隐源   |
| CMP_BLANKING_TIMER1_OC1   | TIMER1_CH1输出比较信号为消隐源   |
| CMP_BLANKING_TIMER7_OC1   | TIMER7_CH1输出比较信号为消隐源   |
| 输出参数{out}                 |  |
| -                         | -  |
| 返回值                       |  |
| -                         | -  |

例如:

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

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

### 函数 cmp\_enable

函数cmp\_enable描述见下表:

表 3-152. 函数 cmp\_enable

|            |  |
|------------|--|
| 函数名称       | cmp_enable                                 |
| 函数原型       | void cmp_enable(cmp_enum cmp_periph);      |
| 功能描述       | 使能CMP                                      |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| cmp_periph | 参考枚举 <a href="#">表 3-146. 枚举类型cmp_enum</a> |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如：

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

### 函数 cmp\_disable

函数cmp\_disable描述见下表：

表 3-153. 函数 cmp\_disable

|            |  |
|------------|--|
| 函数名称       | cmp_disable                                |
| 函数原型       | void cmp_disable(cmp_enum cmp_periph);     |
| 功能描述       | 禁能CMP                                      |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| cmp_periph | 参考枚举 <a href="#">表 3-146. 枚举类型cmp_enum</a> |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如：

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

### 函数 cmp\_lock\_enable

函数cmp\_lock\_enable描述见下表：



表 3-154. 函数 cmp\_lock\_enable

|            |  |
|------------|--|
| 函数名称       | cmp_lock_enable                            |
| 函数原型       | void cmp_lock_enable(cmp_enum cmp_periph); |
| 功能描述       | 锁定CMP                                      |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| cmp_periph | 参考枚举 <a href="#">表 3-146. 枚举类型cmp_enum</a> |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如：

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

### 函数 cmp\_voltage\_scaler\_enable

函数cmp\_voltage\_scaler\_enable描述见下表：

表 3-155. 函数 cmp\_voltage\_scaler\_enable

|            |  |
|------------|--|
| 函数名称       | cmp_voltage_scaler_enable                            |
| 函数原型       | void cmp_voltage_scaler_enable(cmp_enum cmp_periph); |
| 功能描述       | 使能CMP带隙标量  |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| cmp_periph | 参考枚举 <a href="#">表 3-146. 枚举类型cmp_enum</a>           |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如：

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

### 函数 cmp\_voltage\_scaler\_disable

函数cmp\_voltage\_scaler\_disable描述见下表：

表 3-156. 函数 cmp\_voltage\_scaler\_disable

|            |   |
|------------|---|
| 函数名称       | cmp_voltage_scaler_disable                            |
| 函数原型       | void cmp_voltage_scaler_disable(cmp_enum cmp_periph); |
| 功能描述       | 禁能CMP带隙标量   |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| cmp_periph | 参考枚举 <a href="#">表 3-146. 枚举类型cmp_enum</a>            |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如：

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

### 函数 cmp\_scaler\_bridge\_enable

函数cmp\_scaler\_bridge\_enable描述见下表：

表 3-157. 函数 cmp\_scaler\_bridge\_enable

|            |   |
|------------|---|
| 函数名称       | cmp_scaler_bridge_enable                            |
| 函数原型       | void cmp_scaler_bridge_enable(cmp_enum cmp_periph); |
| 功能描述       | 使能CMP标量桥接   |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| cmp_periph | 参考枚举 <a href="#">表 3-146. 枚举类型cmp_enum</a>          |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如：

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

### 函数 cmp\_scaler\_bridge\_disable

函数cmp\_scaler\_bridge\_disable描述见下表：

表 3-158. 函数 cmp\_scaler\_bridge\_disable

|            |  |
|------------|--|
| 函数名称       | cmp_scaler_bridge_disable                            |
| 函数原型       | void cmp_scaler_bridge_disable(cmp_enum cmp_periph); |
| 功能描述       | 禁能CMP标量桥接  |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| cmp_periph | 参考枚举 <a href="#">表 3-146. 枚举类型cmp_enum</a>           |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如:

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

### 函数 cmp\_output\_level\_get

函数cmp\_output\_level\_get描述见下表:

表 3-159. 函数 cmp\_output\_level\_get

|                          |   |
|--------------------------|---|
| 函数名称                     | cmp_output_level_get                                |
| 函数原型                     | uint32_t cmp_output_level_get(uint32_t cmp_periph); |
| 功能描述                     | 获取CMP输出状态   |
| 先决条件                     | -   |
| 被调用函数                    | -   |
| 输入参数{in}                 |   |
| cmp_periph               | 参考枚举 <a href="#">表 3-146. 枚举类型cmp_enum</a>          |
| 输出参数{out}                |   |
| -                        | -   |
| 返回值                      |   |
| uint32_t                 | 输出电平  |
| CMP_OUTPUTLEV<br>EL_HIGH | 比较器输出高电平  |
| CMP_OUTPUTLEV<br>EL_LOW  | 比较器输出低电平  |

例如:

```
uint32_t level;

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

## 3.6. CRC

循环冗余校验码是一种用在数字网络和存储设备上的差错校验码，可以校验原始数据的偶然误差。章节[3.6.1](#)描述了CRC的寄存器列表，章节[3.6.2](#)对CRC库函数进行说明。

### 3.6.1. 外设寄存器说明

CRC寄存器列表如下表所示：

表 3-160. CRC 寄存器

| 寄存器名称     | 寄存器描述      |
|-----------|------------|
| CRC_DATA  | CRC数据寄存器   |
| CRC_FDATA | CRC独立数据寄存器 |
| CRC_CTL   | CRC控制寄存器   |
| CRC_IDATA | CRC初值寄存器   |
| CRC_POLY  | CRC多项式寄存器  |

### 3.6.2. 外设库函数说明

CRC库函数列表如下表所示：

表 3-161. CRC 库函数

| 库函数名称                           | 库函数描述              |
|---------------------------------|--------------------|
| crc_deinit                      | 复位CRC计算单元          |
| crc_reverse_output_data_enable  | 使能输出数据翻转功能         |
| crc_reverse_output_data_disable | 失能输出数据翻转功能         |
| crc_data_register_reset         | 根据数据寄存器的复位值复位数据寄存器 |
| crc_data_register_read          | 读数据寄存器             |
| crc_free_data_register_read     | 读独立数据寄存器           |
| crc_free_data_register_write    | 写独立数据寄存器           |
| crc_init_data_register_write    | 写初值寄存器             |
| crc_input_data_reverse_config   | 配置输入数据翻转功能         |
| crc_polynomial_size_set         | 配置多项式长度            |
| crc_polynomial_set              | 设置多项式寄存器数据         |
| crc_single_data_calculate       | CRC计算单个数据          |
| crc_block_data_calculate        | CRC计算数组            |

#### 函数 crc\_deinit

函数crc\_deinit描述见下表：

表 3-162. 函数 crc\_deinit

|      |                        |
|------|------------------------|
| 函数名称 | crc_deinit             |
| 函数原形 | void crc_deinit(void); |

|           |           |
|-----------|-----------|
| 功能描述      | 复位CRC计算单元 |
| 先决条件      | -         |
| 被调用函数     | -         |
| 输入参数{in}  |           |
| -         | -         |
| 输出参数{out} |           |
| -         | -         |
| 返回值       |           |
| -         | -         |

例如：

```
/* reset crc */
```

```
crc_deinit();
```

### 函数 `crc_reverse_output_data_enable`

函数 `crc_reverse_output_data_enable` 描述见下表：

**表 3-163. 函数 `crc_reverse_output_data_enable`**

|           |   |
|-----------|---|
| 函数名称      | <code>crc_reverse_output_data_enable</code>             |
| 函数原形      | <code>void crc_reverse_output_data_enable(void);</code> |
| 功能描述      | 使能输出数据翻转功能  |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| -         | -   |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

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

```
crc_reverse_output_data_enable();
```

### 函数 `crc_reverse_output_data_disable`

函数 `crc_reverse_output_data_disable` 描述见下表：

**表 3-164. 函数 `crc_reverse_output_data_disable`**

|      |  |
|------|--|
| 函数名称 | <code>crc_reverse_output_data_disable</code>             |
| 函数原形 | <code>void crc_reverse_output_data_disable(void);</code> |
| 功能描述 | 失能输出数据翻转功能   |

|           |   |
|-----------|---|
| 先决条件      | - |
| 被调用函数     | - |
| 输入参数{in}  |   |
| -         | - |
| 输出参数{out} |   |
| -         | - |
| 返回值       |   |
| -         | - |

例如：

```
/* disable crc reverse operation of output data */
```

```
crc_reverse_output_data_disable();
```

### 函数 `crc_data_register_reset`

函数 `crc_data_register_reset` 描述见下表：

**表 3-165. 函数 `crc_data_register_reset`**

|           |  |
|-----------|--|
| 函数名称      | <code>crc_data_register_reset</code>             |
| 函数原形      | <code>void crc_data_register_reset(void);</code> |
| 功能描述      | 根据数据寄存器的复位值（0xFFFFFFFF）复位数据寄存器                   |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如：

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

```
crc_data_register_reset();
```

### 函数 `crc_data_register_read`

函数 `crc_data_register_read` 描述见下表：

**表 3-166. 函数 `crc_data_register_read`**

|      |   |
|------|---|
| 函数名称 | <code>crc_data_register_read</code>                 |
| 函数原形 | <code>uint32_t crc_data_register_read(void);</code> |
| 功能描述 | 读数据寄存器  |
| 先决条件 | -   |

|           |                              |
|-----------|------------------------------|
| 被调用函数     | -                            |
| 输入参数{in}  |                              |
| -         | -                            |
| 输出参数{out} |                              |
| -         | -                            |
| 返回值       |                              |
| uint32_t  | 从数据寄存器读取的32位数据（0-0xFFFFFFFF） |

例如：

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

```
uint32_t crc_value = 0;
```

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

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

函数crc\_free\_data\_register\_read描述见下表：

表 3-167. 函数 crc\_free\_data\_register\_read

|           |  |
|-----------|--|
| 函数名称      | crc_free_data_register_read                |
| 函数原形      | uint8_t crc_free_data_register_read(void); |
| 功能描述      | 读独立数据寄存器                                   |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| uint8_t   | 从独立数据寄存器读取的8位数据（0-0xFF）                    |

例如：

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

```
uint8_t crc_value = 0;
```

```
crc_value = crc_free_data_register_read();
```

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

函数crc\_free\_data\_register\_write描述见下表：

表 3-168. 函数 crc\_free\_data\_register\_write

|      |   |
|------|---|
| 函数名称 | crc_free_data_register_write                          |
| 函数原形 | void crc_free_data_register_write(uint8_t free_data); |

|           |          |
|-----------|----------|
| 功能描述      | 写独立数据寄存器 |
| 先决条件      | -        |
| 被调用函数     | -        |
| 输入参数{in}  |          |
| free_data | 设定的8位数据  |
| 输出参数{out} |          |
| -         | -        |
| 返回值       |          |
| -         | -        |

例如：

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

### 函数 crc\_init\_data\_register\_write

函数crc\_init\_data\_register\_write描述见下表：

表 3-169. 函数 crc\_init\_data\_register\_write

|           |   |
|-----------|---|
| 函数名称      | crc_init_data_register_write                          |
| 函数原形      | void crc_init_data_register_write(uint32_t init_data) |
| 功能描述      | 写初值寄存器  |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| init_data | 设定的32位数据  |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

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

### 函数 crc\_input\_data\_reverse\_config

函数crc\_input\_data\_reverse\_config描述见下表：

表 3-170. 函数 crc\_input\_data\_reverse\_config

|      |   |
|------|---|
| 函数名称 | crc_input_data_reverse_config                             |
| 函数原形 | void crc_input_data_reverse_config(uint32_t data_reverse) |
| 功能描述 | 配置输入数据翻转功能  |



|                                |             |
|--------------------------------|-------------|
| 先决条件                           | -           |
| 被调用函数                          | -           |
| 输入参数{in}                       |             |
| <b>data_reverse</b>            | 设定的输入数据翻转功能 |
| <i>CRC_INPUT_DATA_NOT</i>      | 输入数据不翻转     |
| <i>CRC_INPUT_DATA_BYTE</i>     | 输入数据按字节翻转   |
| <i>CRC_INPUT_DATA_HALFWORD</i> | 输入数据按半字翻转   |
| <i>CRC_INPUT_DATA_WORD</i>     | 输入数据按字翻转    |
| 输出参数{out}                      |             |
| -                              | -           |
| 返回值                            |             |
| -                              | -           |

例如：

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

```
crc_input_data_reverse_config(CRC_INPUT_DATA_WORD);
```

### 函数 **crc\_polynomial\_size\_set**

函数 **crc\_polynomial\_size\_set** 描述见下表：

**表 3-171. 函数 **crc\_polynomial\_size\_set****

|                      |  |
|----------------------|--|
| 函数名称                 | crc_polynomial_size_set                          |
| 函数原形                 | void crc_polynomial_size_set(uint32_t poly_size) |
| 功能描述                 | 配置多项式长度  |
| 先决条件                 | -  |
| 被调用函数                | -  |
| 输入参数{in}             |  |
| <b>poly_size</b>     | 多项式的长度   |
| <i>CRC_CTL_PS_32</i> | 32位多项式值用于CRC计算                                   |
| <i>CRC_CTL_PS_16</i> | 16位多项式值用于CRC计算                                   |
| <i>CRC_CTL_PS_8</i>  | 8位多项式值用于CRC计算                                    |
| <i>CRC_CTL_PS_7</i>  | 7位多项式值用于CRC计算                                    |
| 输出参数{out}            |  |
| -                    | -  |
| 返回值                  |  |
| -                    | -  |

例如：

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

```
crc_polynomial_size_set(CRC_CTL_PS_7);
```

### 函数 `crc_polynomial_set`

函数 `crc_polynomial_set` 描述见下表：

**表 3-172. 函数 `crc_polynomial_set`**

|             |   |
|-------------|---|
| 函数名称        | <code>crc_polynomial_set</code>                     |
| 函数原形        | <code>void crc_polynomial_set(uint32_t poly)</code> |
| 功能描述        | 设置多项式寄存器值   |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| <b>poly</b> | 设置多项式长度寄存器值   |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如：

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

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

### 函数 `crc_single_data_calculate`

函数 `crc_single_data_calculate` 描述见下表：

**表 3-173. 函数 `crc_single_data_calculate`**

|                                    |   |
|------------------------------------|---|
| 函数名称                               | <code>crc_single_data_calculate</code>  |
| 函数原形                               | <code>uint32_t crc_single_data_calculate(uint32_t sdata, uint8_t data_format);</code> |
| 功能描述                               | CRC 计算单个数据  |
| 先决条件                               | -   |
| 被调用函数                              | -   |
| 输入参数{in}                           |   |
| <b>sdata</b>                       | 特定的输入数据   |
| 输入参数{in}                           |   |
| <b>data_format</b>                 | 数据格式  |
| <code>INPUT_FORMAT_WORD</code>     | 输入数据格式为字  |
| <code>INPUT_FORMAT_HALFWORD</code> | 输入数据格式为半字   |
| <code>INPUT_FORMAT_BYTE</code>     | 输入数据格式为字节   |

|           |                       |
|-----------|-----------------------|
| YTE       |                       |
| 输出参数{out} |                       |
| -         | -                     |
| 返回值       |                       |
| uint32_t  | CRC计算结果（0-0xFFFFFFFF） |

例如：

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

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

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

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

### 函数 crc\_block\_data\_calculate

函数crc\_block\_data\_calculate描述见下表：

表 3-174. 函数 crc\_block\_data\_calculate

|                       |   |
|-----------------------|---|
| 函数名称                  | crc_block_data_calculate  |
| 函数原形                  | uint32_t crc_block_data_calculate(void *array, uint32_t size, uint8_t data_format); |
| 功能描述                  | CRC计算数组   |
| 先决条件                  | -   |
| 被调用函数                 | -   |
| 输入参数{in}              |   |
| array                 | 输入数组指针  |
| 输入参数{in}              |   |
| size                  | 数组大小  |
| 输入参数{in}              |   |
| data_format           | 数据格式  |
| INPUT_FORMAT_WORD     | 输入数据格式为字  |
| INPUT_FORMAT_HALFWORD | 输入数据格式为半字   |
| INPUT_FORMAT_BYTE     | 输入数据格式为字节   |
| 输出参数{out}             |   |
| -                     | -   |
| 返回值                   |   |
| uint32_t              | CRC计算结果（0-0xFFFFFFFF）   |

例如：

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

```
#define BUFFER_SIZE    6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {

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

valcrc = crc_block_data_calculate(data_buffer, BUFFER_SIZE, INPUT_FORMAT_WORD);
```

### 3.7. DBG

调试系统帮助调试者在低功耗模式下调试或者进行一些外设调试。章节[3.7.1](#)描述了DBG的寄存器列表，章节[3.7.2](#)对DBG库函数进行说明。

#### 3.7.1. 外设寄存器说明

DBG寄存器列表如下表所示：

表 3-175. DBG 寄存器

| 寄存器名称   | 寄存器描述     |
|---------|-----------|
| DBG_ID  | DBG ID寄存器 |
| DBG_CTL | DBG控制寄存器  |

#### 3.7.2. 外设库函数说明

DBG库函数列表如下表所示：

表 3-176. DBG 库函数

| 库函数名称                 | 库函数描述             |
|-----------------------|-------------------|
| dbg_deinit            | 复位DBG寄存器          |
| dbg_id_get            | 读DBG_ID寄存器        |
| dbg_low_power_enable  | 使能低功耗模式的MCU调试保持功能 |
| dbg_low_power_disable | 禁能低功耗模式的MCU调试保持功能 |
| dbg_periph_enable     | 使能外设的MCU调试保持功能    |
| dbg_periph_disable    | 禁能外设的MCU调试保持功能    |

#### 枚举类型 dbg\_periph\_enum

表 3-177. 枚举类型 dbg\_periph\_enum

| 成员名称            | 功能描述                    |
|-----------------|-------------------------|
| DBG_FWDGT_HOLD  | 当内核停止时，保持FWDGT计数器时钟     |
| DBG_WWDGT_HOLD  | 当内核停止时，保持WWDGT计数器时钟     |
| DBG_TIMER0_HOLD | 当内核停止时，保持TIMER0计数器计数值不变 |
| DBG_TIMER1_HOLD | 当内核停止时，保持TIMER1计数器计数值不变 |

| 成员名称             | 功能描述                         |
|------------------|------------------------------|
| DBG_I2C0_HOLD    | 当内核停止时，保持I2C0的SMBUS状态不变，用于调试 |
| DBG_I2C1_HOLD    | 当内核停止时，保持I2C1的SMBUS状态不变，用于调试 |
| DBG_TIMER7_HOLD  | 当内核停止时，保持TIMER7计数器计数值不变      |
| DBG_TIMER5_HOLD  | 当内核停止时，保持TIMER5计数器计数值不变      |
| DBG_TIMER6_HOLD  | 当内核停止时，保持TIMER6计数器计数值不变      |
| DBG_MFCOM_HOLD   | 当内核停止时，保持MFCOM计数器计数值不变       |
| DBG_CAN0_HOLD    | 当内核停止时，保持CAN0计数器计数值不变        |
| DBG_CAN1_HOLD    | 当内核停止时，保持CAN1计数器计数值不变        |
| DBG_TIMER20_HOLD | 当内核停止时，保持TIMER20计数器计数值不变     |
| DBG_TIMER19_HOLD | 当内核停止时，保持TIMER19计数器计数值不变     |

## 函数 dbg\_deinit

函数dbg\_deinit描述见下表：

表 3-178. 函数 dbg\_deinit

| 函数名称      | dbg_deinit             |
|-----------|------------------------|
| 函数原形      | void dbg_deinit(void); |
| 功能描述      | 复位DBG寄存器               |
| 先决条件      | -                      |
| 被调用函数     | -                      |
| 输入参数{in}  |                        |
| -         | -                      |
| 输出参数{out} |                        |
| -         | -                      |
| 返回值       |                        |
| -         | -                      |

例如：

```
/* reset DBG register */
```

```
dbg_deinit();
```

## 函数 dbg\_id\_get

函数dbg\_id\_get描述见下表：

表 3-179. 函数 dbg\_id\_get

| 函数名称 | dbg_id_get                 |
|------|----------------------------|
| 函数原形 | uint32_t dbg_id_get(void); |
| 功能描述 | 读DBG_ID寄存器                 |
| 先决条件 | -                          |

|           |                       |
|-----------|-----------------------|
| 被调用函数     | -                     |
| 输入参数{in}  |                       |
| -         | -                     |
| 输出参数{out} |                       |
| -         | -                     |
| 返回值       |                       |
| uint32_t  | DBG ID (0-0xFFFFFFFF) |

例如:

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

```
uint32_t id_value = 0;
```

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

### 函数 dbg\_low\_power\_enable

函数dbg\_low\_power\_enable描述见下表:

表 3-180. 函数 dbg\_low\_power\_enable

|                         |  |
|-------------------------|--|
| 函数名称                    | dbg_low_power_enable                               |
| 函数原形                    | void dbg_low_power_enable(uint32_t dbg_low_power); |
| 功能描述                    | 使能低功耗模式的MCU调试保持功能                                  |
| 先决条件                    | -  |
| 被调用函数                   | -  |
| 输入参数{in}                |  |
| dbg_low_power           | 低功耗模式调试保持  |
| DBG_LOW_POWER_SLEEP     | 在睡眠模式下, 保持调试器连接, 可进行调试                             |
| DBG_LOW_POWER_DEEPSLEEP | 在深度睡眠模式下, 保持调试器连接, 可进行调试                           |
| DBG_LOW_POWER_STANDBY   | 在待机模式下, 保持调试器连接, 可进行调试                             |
| 输出参数{out}               |  |
| -                       | -  |
| 返回值                     |  |
| -                       | -  |

例如:

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

```
dbg_low_power_enable(DBG_LOW_POWER_SLEEP);
```

**函数 dbg\_low\_power\_disable**

函数dbg\_low\_power\_disable描述见下表：

**表 3-181. 函数 dbg\_low\_power\_disable**

|                         |   |
|-------------------------|---|
| 函数名称                    | dbg_low_power_disable                               |
| 函数原形                    | void dbg_low_power_disable(uint32_t dbg_low_power); |
| 功能描述                    | 禁能低功耗模式的MCU调试保持功能                                   |
| 先决条件                    | -   |
| 被调用函数                   | -   |
| <b>输入参数{in}</b>         |   |
| dbg_low_power           | 低功耗模式调试保持禁止   |
| DBG_LOW_POWER_SLEEP     | 在睡眠模式下，不保持调试器连接，无法进行调试                              |
| DBG_LOW_POWER_DEEPSLEEP | 在深度睡眠模式下，不保持调试器连接，无法进行调试                            |
| DBG_LOW_POWER_STANDBY   | 在待机模式下，不保持调试器连接，无法进行调试                              |
| <b>输出参数{out}</b>        |   |
| -                       | -   |
| <b>返回值</b>              |   |
| -                       | -   |

例如：

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

**函数 dbg\_periph\_enable**

函数dbg\_periph\_enable描述见下表：

**表 3-182. 函数 dbg\_periph\_enable**

|                 |   |
|-----------------|---|
| 函数名称            | dbg_periph_enable                                   |
| 函数原形            | void dbg_periph_enable(dbg_periph_enum dbg_periph); |
| 功能描述            | 使能外设的MCU调试保持功能                                      |
| 先决条件            | -   |
| 被调用函数           | -   |
| <b>输入参数{in}</b> |   |
| dbg_periph      | 参考枚举变量 <a href="#">表3-177. 枚举类型dbg_periph_enum</a>  |
| DBG_FWDGT_HOLD  | 当内核停止时，保持FWDGT计数器时钟                                 |
| DBG_WWDGT_HOLD  | 当内核停止时，保持WWDGT计数器时钟                                 |
| DBG_TIMERx_HOLD | 当内核停止时，保持TIMERx计数器计数值不变（x=1, 5, 6, 7, 19, 20）       |

|                       |                                     |
|-----------------------|-------------------------------------|
| <i>DBG_I2Cx_HOLD</i>  | 当内核停止时，保持I2Cx（x=0，1）的SMBUS状态不变，用于调试 |
| <i>DBG_MFCOM_HOLD</i> | 当内核停止时，保持MFCOM计数器，用于调试              |
| <i>DBG_CANx_HOLD</i>  | 当内核停止时，保持CANx（x=0，1）的计数器数值不变，用于调试   |
| 输出参数{out}             |                                     |
| -                     | -                                   |
| 返回值                   |                                     |
| -                     | -                                   |

例如：

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

```
dbg_periph_enable(DBG_TIMER1_HOLD);
```

### 函数 dbg\_periph\_disable

函数dbg\_periph\_disable描述见下表：

表 3-183. 函数 dbg\_periph\_disable

|                        |  |
|------------------------|--|
| 函数名称                   | dbg_periph_disable                                   |
| 函数原形                   | void dbg_periph_disable(dbg_periph_enum dbg_periph); |
| 功能描述                   | 禁能外设的MCU调试保持功能                                       |
| 先决条件                   | -  |
| 被调用函数                  | -  |
| 输入参数{in}               |  |
| dbg_periph             | 参考枚举变量 <a href="#">表3-177. 枚举类型dbg_periph_enum</a>   |
| <i>DBG_FWDGT_HOLD</i>  | 当内核停止时，保持FWDGT计数器时钟                                  |
| <i>DBG_WWDGT_HOLD</i>  | 当内核停止时，保持WWDGT计数器时钟                                  |
| <i>DBG_TIMERx_HOLD</i> | 当内核停止时，保持TIMERx计数器计数值不变（x=1，5，6，7，19，20）             |
| <i>DBG_I2Cx_HOLD</i>   | 当内核停止时，保持I2Cx（x=0，1）的SMBUS状态不变，用于调试                  |
| <i>DBG_MFCOM_HOLD</i>  | 当内核停止时，保持MFCOM计数器，用于调试                               |
| <i>DBG_CANx_HOLD</i>   | 当内核停止时，保持CANx（x=0，1）的计数器数值不变                         |
| 输出参数{out}              |  |
| -                      | -  |
| 返回值                    |  |
| -                      | -  |

例如：

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

```
dbg_periph_disable(DBG_TIMER1_HOLD);
```

## 3.8. DAC

数字/模拟转换器可以将12位的数字数据转换为外部引脚上的电压输出，章节[3.8.1](#)描述了DAC



的寄存器列表，章节[3.8.2](#)对DAC库函数进行说明。

### 3.8.1. 外设寄存器说明

DAC寄存器列表如下表所示：

**表 3-184. DAC 寄存器**

| 寄存器名称          | 寄存器描述                   |
|----------------|-------------------------|
| DAC_CTL0       | DACx控制寄存器0              |
| DAC_SWT        | DACx软件触发寄存器             |
| DAC_OUT0_R12DH | DACx_OUT0 12位右对齐数据保持寄存器 |
| DAC_OUT0_L12DH | DACx_OUT0 12位左对齐数据保持寄存器 |
| DAC_OUT0_R8DH  | DACx_OUT0 8位右对齐数据保持寄存器  |
| DAC_OUT0_DO    | DACx_OUT0数据输出寄存器        |
| DAC_STAT0      | DACx状态寄存器0              |

### 3.8.2. 外设库函数说明

DAC库函数列表如下表所示：

**表 3-185. DAC 库函数**

| 库函数名称                       | 库函数描述          |
|-----------------------------|----------------|
| dac_deinit                  | DAC外设复位        |
| dac_enable                  | DAC使能          |
| dac_disable                 | DAC禁能          |
| dac_dma_enable              | DAC的DMA功能使能    |
| dac_dma_disable             | DAC的DMA功能禁能    |
| dac_gpio_connect_config     | DAC的GPIO连接模式配置 |
| dac_output_buffer_enable    | DAC输出缓冲区使能     |
| dac_output_buffer_disable   | DAC输出缓冲区禁能     |
| dac_output_value_get        | DAC输出数据获取      |
| dac_data_set                | DAC输出数据设置      |
| dac_trigger_enable          | DAC触发使能        |
| dac_trigger_disable         | DAC触发禁能        |
| dac_trigger_source_config   | DAC触发源配置       |
| dac_software_trigger_enable | DAC软件触发使能      |
| dac_wave_mode_config        | DAC噪声波模式配置     |
| dac_lfsr_noise_config       | DAC LFSR模式配置   |
| dac_triangle_noise_config   | DAC三角波模式配置     |
| dac_flag_get                | DAC标志位获取       |
| dac_flag_clear              | DAC标志位清除       |
| dac_interrupt_enable        | DAC中断使能        |
| dac_interrupt_disable       | DAC中断禁能        |

| 库函数名称                                 | 库函数描述      |
|---------------------------------------|------------|
| <code>dac_interrupt_flag_get</code>   | DAC中断标志位获取 |
| <code>dac_interrupt_flag_clear</code> | DAC中断标志位清除 |

### 函数 `dac_deinit`

函数`dac_deinit`描述见下表：

**表 3-186. 函数 `dac_deinit`**

|                         |  |
|-------------------------|--|
| 函数名称                    | <code>dac_deinit</code>  |
| 函数原型                    | <code>void dac_deinit(uint32_t dac_periph);</code>                           |
| 功能描述                    | DAC外设复位  |
| 先决条件                    | -  |
| 被调用函数                   | <code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code> |
| 输入参数{in}                |  |
| <code>dac_periph</code> | DAC外设  |
| <code>DACx</code>       | DAC外设选择 (x = 0)  |
| 输出参数{out}               |  |
| -                       | -  |
| 返回值                     |  |
| -                       | -  |

例如：

```
/* deinitialize DAC0 */
```

```
dac_deinit(DAC0);
```

### 函数 `dac_enable`

函数`dac_enable`描述见下表：

**表 3-187. 函数 `dac_enable`**

|                         |   |
|-------------------------|---|
| 函数名称                    | <code>dac_enable</code>   |
| 函数原型                    | <code>void dac_enable(uint32_t dac_periph, uint8_t dac_out);</code> |
| 功能描述                    | DAC使能   |
| 先决条件                    | -   |
| 被调用函数                   | -   |
| 输入参数{in}                |   |
| <code>dac_periph</code> | DAC外设   |
| <code>DACx</code>       | DAC外设选择 (x = 0)   |
| 输入参数{in}                |   |
| <code>dac_out</code>    | DAC输出   |
| <code>DAC_OUTx</code>   | DAC输出通道选择 (x = 0)   |
| 输出参数{out}               |   |
| -                       | -   |

| 返回值 |   |
|-----|---|
| -   | - |

例如:

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

### 函数 **dac\_disable**

函数dac\_disable描述见下表:

**表 3-188. 函数 **dac\_disable****

| 函数名称              | dac_disable   |
|-------------------|---|
| 函数原型              | void dac_disable(uint32_t dac_periph, uint8_t dac_out); |
| 功能描述              | DAC禁能   |
| 先决条件              | -   |
| 被调用函数             | -   |
| 输入参数{in}          |   |
| <b>dac_periph</b> | DAC外设   |
| <i>DACx</i>       | DAC外设选择 (x = 0)   |
| 输入参数{in}          |   |
| <b>dac_out</b>    | DAC输出   |
| <i>DAC_OUTx</i>   | DAC输出通道选择 (x = 0)                                       |
| 输出参数{out}         |   |
| -                 | -   |
| 返回值               |   |
| -                 | -   |

例如:

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

### 函数 **dac\_dma\_enable**

函数dac\_dma\_enable描述见下表:

**表 3-189. 函数 **dac\_dma\_enable****

| 函数名称     | dac_dma_enable   |
|----------|--|
| 函数原型     | void dac_dma_enable(uint32_t dac_periph, uint8_t dac_out); |
| 功能描述     | DAC的DMA功能使能  |
| 先决条件     | -  |
| 被调用函数    | -  |
| 输入参数{in} |  |

|                   |                   |
|-------------------|-------------------|
| <b>dac_periph</b> | DAC外设             |
| <i>DACx</i>       | DAC外设选择 (x = 0)   |
| <b>输入参数{in}</b>   |                   |
| <b>dac_out</b>    | DAC输出             |
| <i>DAC_OUTx</i>   | DAC输出通道选择 (x = 0) |
| <b>输出参数{out}</b>  |                   |
| -                 | -                 |
| <b>返回值</b>        |                   |
| -                 | -                 |

例如:

```
/* enable DAC0_OUT0 DMA function */
```

```
dac_dma_enable(DAC0, DAC_OUT0);
```

### 函数 **dac\_dma\_disable**

函数dac\_dma\_disable描述见下表:

**表 3-190. 函数 **dac\_dma\_disable****

|                   |   |
|-------------------|---|
| <b>函数名称</b>       | dac_dma_disable   |
| <b>函数原型</b>       | void dac_dma_disable(uint32_t dac_periph, uint8_t dac_out); |
| <b>功能描述</b>       | DAC的DMA功能禁能   |
| <b>先决条件</b>       | -   |
| <b>被调用函数</b>      | -   |
| <b>输入参数{in}</b>   |   |
| <b>dac_periph</b> | DAC外设   |
| <i>DACx</i>       | DAC外设选择 (x = 0)   |
| <b>输入参数{in}</b>   |   |
| <b>dac_out</b>    | DAC输出   |
| <i>DAC_OUTx</i>   | DAC输出通道选择 (x = 0)   |
| <b>输出参数{out}</b>  |   |
| -                 | -   |
| <b>返回值</b>        |   |
| -                 | -   |

例如:

```
/* disable DAC0_OUT0 DMA function */
```

```
dac_dma_disable(DAC0, DAC_OUT0);
```

### 函数 **dac\_gpio\_connect\_config**

函数dac\_gpio\_connect\_config描述见下表:

表 3-191. 函数 `dac_gpio_connect_config`

|                                    |   |
|------------------------------------|---|
| 函数名称                               | <code>dac_gpio_connect_config</code>  |
| 函数原型                               | <code>void dac_gpio_connect_config(uint32_t dac_periph, uint8_t dac_out, uint32_t gpio_connect);</code> |
| 功能描述                               | DAC的GPIO连接配置  |
| 先决条件                               | -   |
| 被调用函数                              | -   |
| 输入参数{in}                           |   |
| <code>dac_periph</code>            | DAC外设   |
| <code>DACx</code>                  | DAC外设选择 ( $x = 0$ )   |
| 输入参数{in}                           |   |
| <code>dac_out</code>               | DAC输出   |
| <code>DAC_OUTx</code>              | DAC输出通道选择 ( $x = 0$ )   |
| 输入参数{in}                           |   |
| <code>gpio_connect</code>          | DAC_OUTx连接GPIO模式  |
| <code>PIN_PERIPHERAL</code>        | DAC_OUTx输出连接外部管脚以及片上CMP   |
| <code>PIN_PERIPHERAL_BUFFER</code> | 根据输出buffer的开关，决定DAC_OUTx连接外部管脚以及片上CMP的模式  |
| 输出参数{out}                          |   |
| -                                  | -   |
| 返回值                                |   |
| -                                  | -   |

例如:

```
/* configure DAC0_OUT0 GPIO connection working in PIN_PERIPHERAL*/
```

```
dac_gpio_connect_config (DAC0, DAC_OUT0, PIN_PERIPHERAL);
```

### 函数 `dac_output_buffer_enable`

函数`dac_output_buffer_enable`描述见下表:

表 3-192. 函数 `dac_output_buffer_enable`

|                         |   |
|-------------------------|---|
| 函数名称                    | <code>dac_output_buffer_enable</code>   |
| 函数原型                    | <code>void dac_output_buffer_enable(uint32_t dac_periph, uint8_t dac_out);</code> |
| 功能描述                    | DAC输出缓冲区使能  |
| 先决条件                    | -   |
| 被调用函数                   | -   |
| 输入参数{in}                |   |
| <code>dac_periph</code> | DAC外设   |
| <code>DACx</code>       | DAC外设选择 ( $x = 0$ )   |
| 输入参数{in}                |   |
| <code>dac_out</code>    | DAC输出   |
| <code>DAC_OUTx</code>   | DAC输出通道选择 ( $x = 0$ )   |

| 输出参数{out} |   |
|-----------|---|
| -         | - |
| 返回值       |   |
| -         | - |

例如:

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

### 函数 dac\_output\_buffer\_disable

函数dac\_output\_buffer\_disable描述见下表:

表 3-193. 函数 dac\_output\_buffer\_disable

| 函数名称       | dac_output_buffer_disable   |
|------------|---|
| 函数原型       | void dac_output_buffer_disable(uint32_t dac_periph, uint8_t dac_out); |
| 功能描述       | DAC输出缓冲区禁能  |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| dac_periph | DAC外设   |
| DACx       | DAC外设选择 (x = 0)   |
| 输入参数{in}   |   |
| dac_out    | DAC输出   |
| DAC_OUTx   | DAC输出通道选择 (x = 0)   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如:

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

### 函数 dac\_output\_value\_get

函数dac\_output\_value\_get描述见下表:

表 3-194. 函数 dac\_output\_value\_get

| 函数名称 | dac_output_value_get   |
|------|--|
| 函数原型 | uint16_t dac_output_value_get(uint32_t dac_periph, uint8_t dac_out); |
| 功能描述 | DAC输出数据获取  |
| 先决条件 | -  |

|                   |                         |
|-------------------|-------------------------|
| 被调用函数             | -                       |
| 输入参数{in}          |                         |
| <b>dac_periph</b> | DAC外设                   |
| <i>DACx</i>       | DAC外设选择 (x = 0)         |
| 输入参数{in}          |                         |
| <b>dac_out</b>    | DAC输出                   |
| <i>DAC_OUTx</i>   | DAC输出通道选择 (x = 0)       |
| 输出参数{out}         |                         |
| -                 | -                       |
| 返回值               |                         |
| <b>uint16_t</b>   | 外设DACx数据保持寄存器值 (0~4095) |

例如:

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

```
uint16_t data = 0;
```

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

### 函数 **dac\_data\_set**

函数dac\_data\_set描述见下表:

表 3-195. 函数 **dac\_data\_set**

|                        |   |
|------------------------|---|
| 函数名称                   | dac_data_set  |
| 函数原型                   | void dac_data_set(uint32_t dac_periph, uint8_t dac_out, uint32_t dac_align, uint16_t data); |
| 功能描述                   | DAC输出数据设置   |
| 先决条件                   | -   |
| 被调用函数                  | -   |
| 输入参数{in}               |   |
| <b>dac_periph</b>      | DAC外设   |
| <i>DACx</i>            | DAC外设选择 (x = 0)   |
| 输入参数{in}               |   |
| <b>dac_out</b>         | DAC输出   |
| <i>DAC_OUTx</i>        | DAC输出通道选择 (x = 0)   |
| 输入参数{in}               |   |
| <b>dac_align</b>       | DAC对齐模式   |
| <i>DAC_ALIGN_12B_R</i> | 12位数据右对齐  |
| <i>DAC_ALIGN_12B_L</i> | 12位数据左对齐  |
| <i>DAC_ALIGN_8B_R</i>  | 8位数据右对齐   |
| 输入参数{in}               |   |
| <b>data</b>            | 写入DAC_OUTx的数据 (0~4095)  |

| 输出参数{out} |   |
|-----------|---|
| -         | - |
| 返回值       |   |
| -         | - |

例如:

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

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

### 函数 dac\_trigger\_enable

函数dac\_trigger\_enable描述见下表:

表 3-196. 函数 dac\_trigger\_enable

| 函数名称       | dac_trigger_enable   |
|------------|--|
| 函数原型       | void dac_trigger_enable(uint32_t dac_periph, uint8_t dac_out); |
| 功能描述       | DAC触发使能  |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| dac_periph | DAC外设  |
| DACx       | DAC外设选择 (x = 0)  |
| 输入参数{in}   |  |
| dac_out    | DAC输出  |
| DAC_OUTx   | DAC输出通道选择 (x = 0)  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如:

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

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

### 函数 dac\_trigger\_disable

函数dac\_trigger\_disable描述见下表:

表 3-197. 函数 dac\_trigger\_disable

| 函数名称 | dac_trigger_disable   |
|------|---|
| 函数原型 | void dac_trigger_disable(uint32_t dac_periph, uint8_t dac_out); |
| 功能描述 | DAC触发禁能   |
| 先决条件 | -   |



|                   |                   |
|-------------------|-------------------|
| 被调用函数             | -                 |
| 输入参数{in}          |                   |
| <b>dac_periph</b> | DAC外设             |
| <i>DACx</i>       | DAC外设选择 (x = 0)   |
| 输入参数{in}          |                   |
| <b>dac_out</b>    | DAC输出             |
| <i>DAC_OUTx</i>   | DAC输出通道选择 (x = 0) |
| 输出参数{out}         |                   |
| -                 | -                 |
| 返回值               |                   |
| -                 | -                 |

例如:

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

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

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

函数dac\_trigger\_source\_config描述见下表:

**表 3-198. 函数 **dac\_trigger\_source\_config****

|                             |   |
|-----------------------------|---|
| 函数名称                        | dac_trigger_source_config   |
| 函数原型                        | void dac_trigger_source_config(uint32_t dac_periph, uint8_t dac_out, uint32_t triggersource); |
| 功能描述                        | DAC触发源配置  |
| 先决条件                        | -   |
| 被调用函数                       | -   |
| 输入参数{in}                    |   |
| <b>dac_periph</b>           | DAC外设   |
| <i>DACx</i>                 | DAC外设选择 (x = 0)   |
| 输入参数{in}                    |   |
| <b>dac_out</b>              | DAC输出   |
| <i>DAC_OUTx</i>             | DAC输出通道选择 (x = 0)   |
| 输入参数{in}                    |   |
| <b>triggersource</b>        | DAC触发源  |
| <i>DAC_TRIGGER_EXTERNAL</i> | TRIGSEL触发   |
| <i>DAC_TRIGGER_SOFTWARE</i> | 软件触发  |
| 输出参数{out}                   |   |
| -                           | -   |
| 返回值                         |   |
| -                           | -   |

例如:

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

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

### 函数 `dac_software_trigger_enable`

函数 `dac_software_trigger_enable` 描述见下表:

**表 3-199. 函数 `dac_software_trigger_enable`**

|                         |  |
|-------------------------|--|
| 函数名称                    | <code>dac_software_trigger_enable</code>   |
| 函数原型                    | <code>void dac_software_trigger_enable(uint32_t dac_periph, uint8_t dac_out);</code> |
| 功能描述                    | DAC软件触发使能  |
| 先决条件                    | -  |
| 被调用函数                   | -  |
| 输入参数{in}                |  |
| <code>dac_periph</code> | DAC外设  |
| <code>DACx</code>       | DAC外设选择 ( $x = 0$ )  |
| 输入参数{in}                |  |
| <code>dac_out</code>    | DAC输出  |
| <code>DAC_OUTx</code>   | DAC输出通道选择 ( $x = 0$ )  |
| 输出参数{out}               |  |
| -                       | -  |
| 返回值                     |  |
| -                       | -  |

例如:

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

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

### 函数 `dac_wave_mode_config`

函数 `dac_wave_mode_config` 描述见下表:

**表 3-200. 函数 `dac_wave_mode_config`**

|                         |   |
|-------------------------|---|
| 函数名称                    | <code>dac_wave_mode_config</code>   |
| 函数原型                    | <code>void dac_wave_mode_config(uint32_t dac_periph, uint8_t dac_out, uint32_t wave_mode);</code> |
| 功能描述                    | DAC噪声波模式配置  |
| 先决条件                    | -   |
| 被调用函数                   | -   |
| 输入参数{in}                |   |
| <code>dac_periph</code> | DAC外设   |
| <code>DACx</code>       | DAC外设选择 ( $x = 0$ )   |

| 输入参数{in}                      |                   |
|-------------------------------|-------------------|
| <b>dac_out</b>                | DAC输出             |
| <i>DAC_OUTx</i>               | DAC输出通道选择 (x = 0) |
| 输入参数{in}                      |                   |
| <b>wave_mode</b>              | 噪声波模式选择           |
| <i>DAC_WAVE_DISABLE</i>       | 噪声波模式禁能           |
| <i>DAC_WAVE_MODE_LFSR</i>     | LFSR噪声波模式         |
| <i>DAC_WAVE_MODE_TRIANGLE</i> | 三角波噪声波模式          |
| 输出参数{out}                     |                   |
| -                             | -                 |
| 返回值                           |                   |
| -                             | -                 |

例如:

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

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

### 函数 **dac\_lfsr\_noise\_config**

函数dac\_lfsr\_noise\_config描述见下表:

**表 3-201. 函数 **dac\_lfsr\_noise\_config****

| <b>函数名称</b>             | dac_lfsr_noise_config   |
|-------------------------|---|
| <b>函数原型</b>             | void dac_lfsr_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t unmask_bits); |
| <b>功能描述</b>             | DAC LFSR模式配置  |
| <b>先决条件</b>             | -   |
| <b>被调用函数</b>            | -   |
| 输入参数{in}                |   |
| <b>dac_periph</b>       | DAC外设   |
| <i>DACx</i>             | DAC外设选择 (x = 0)   |
| 输入参数{in}                |   |
| <b>dac_out</b>          | DAC输出   |
| <i>DAC_OUTx</i>         | DAC输出通道选择 (x = 0)   |
| 输入参数{in}                |   |
| <b>unmask_bits</b>      | 噪声波的非屏蔽位宽   |
| <i>DAC_LFSR_BIT0</i>    | LFSR模式位0非屏蔽   |
| <i>DAC_LFSR_BITSx_0</i> | LFSR模式位[x:0]非屏蔽 (x = 1,2,3..11)   |
| 输出参数{out}               |   |

|     |   |
|-----|---|
| -   | - |
| 返回值 |   |
| -   | - |

例如:

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

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

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

函数dac\_triangle\_noise\_config描述见下表:

**表 3-202. 函数 dac\_triangle\_noise\_config**

|                          |   |
|--------------------------|---|
| 函数名称                     | dac_triangle_noise_config   |
| 函数原型                     | void dac_triangle_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t amplitude); |
| 功能描述                     | DAC三角波模式配置  |
| 先决条件                     | -   |
| 被调用函数                    | -   |
| 输入参数{in}                 |   |
| dac_periph               | DAC外设   |
| DACx                     | DAC外设选择 (x = 0)   |
| 输入参数{in}                 |   |
| dac_out                  | DAC输出   |
| DAC_OUTx                 | DAC输出通道选择 (x = 0)   |
| 输入参数{in}                 |   |
| amplitude                | 三角波幅值   |
| DAC_TRIANGLE_AMPLITUDE_x | $x = 2^n - 1$ (n = 1..12)   |
| 输出参数{out}                |   |
| -                        | -   |
| 返回值                      |   |
| -                        | -   |

例如:

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

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

### 函数 **dac\_flag\_get**

函数dac\_flag\_get描述见下表:

表 3-203. 函数 `dac_flag_get`

|   |   |
|---|---|
| 函数名称  | <code>dac_flag_get</code>   |
| 函数原型  | <code>FlagStatus dac_flag_get(uint32_t dac_periph, uint32_t flag);</code> |
| 功能描述  | DAC标志位获取  |
| 先决条件  | -   |
| 被调用函数   | -   |
| 输入参数{in}                                      |   |
| <code>dac_periph</code>                       | DAC外设   |
| <code>DACx</code>                             | DAC外设选择 ( $x = 0$ )   |
| 输入参数{in}                                      |   |
| <code>flag</code>                             | DAC状态标志位  |
| <code>DAC_FLAG_DDUD</code><br><code>R0</code> | DACx_OUT0 DMA欠载标志位  |
| 输出参数{out}                                     |   |
| -   | -   |
| 返回值   |   |
| <code>FlagStatus</code>                       | DAC位状态 (SET或RESET)  |

例如:

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

```
FlagStatus flag;
```

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

### 函数 `dac_flag_clear`

函数`dac_flag_clear`描述见下表:

表 3-204. 函数 `dac_flag_clear`

|   |   |
|---|---|
| 函数名称  | <code>dac_flag_clear</code>   |
| 函数原型  | <code>void dac_flag_clear(uint32_t dac_periph, uint32_t flag);</code> |
| 功能描述  | DAC标志位清除  |
| 先决条件  | -   |
| 被调用函数   | -   |
| 输入参数{in}                                      |   |
| <code>dac_periph</code>                       | DAC外设   |
| <code>DACx</code>                             | DAC外设选择 ( $x = 0$ )   |
| 输入参数{in}                                      |   |
| <code>flag</code>                             | DAC状态标志位  |
| <code>DAC_FLAG_DDUD</code><br><code>R0</code> | DACx_OUT0 DMA欠载标志位  |
| 输出参数{out}                                     |   |
| -   | -   |

| 返回值 |   |
|-----|---|
| -   | - |

例如:

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

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

### 函数 dac\_interrupt\_enable

函数dac\_interrupt\_enable描述见下表:

表 3-205. 函数 dac\_interrupt\_enable

| 函数名称              | dac_interrupt_enable  |
|-------------------|---|
| 函数原型              | void dac_interrupt_enable(uint32_t dac_periph, uint32_t interrupt); |
| 功能描述              | DAC中断使能   |
| 先决条件              | -   |
| 被调用函数             | -   |
| 输入参数{in}          |   |
| <b>dac_periph</b> | DAC外设   |
| DACx              | DAC外设选择 (x = 0)   |
| 输入参数{in}          |   |
| <b>interrupt</b>  | DAC中断   |
| DAC_INT_DDUDR0    | DACx_OUT0 DMA欠载中断   |
| 输出参数{out}         |   |
| -                 | -   |
| 返回值               |   |
| -                 | -   |

例如:

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

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

### 函数 dac\_interrupt\_disable

函数dac\_interrupt\_disable描述见下表:

表 3-206. 函数 dac\_interrupt\_disable

| 函数名称     | dac_interrupt_disable  |
|----------|--|
| 函数原型     | void dac_interrupt_disable(uint32_t dac_periph, uint32_t interrupt); |
| 功能描述     | DAC中断禁能  |
| 先决条件     | -  |
| 被调用函数    | -  |
| 输入参数{in} |  |

|                       |                   |
|-----------------------|-------------------|
| <b>dac_periph</b>     | DAC外设             |
| <i>DACx</i>           | DAC外设选择 (x = 0)   |
| <b>输入参数{in}</b>       |                   |
| <b>interrupt</b>      | DAC中断             |
| <i>DAC_INT_DDUDR0</i> | DACx_OUT0 DMA欠载中断 |
| <b>输出参数{out}</b>      |                   |
| -                     | -                 |
| <b>返回值</b>            |                   |
| -                     | -                 |

例如:

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

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

### 函数 **dac\_interrupt\_flag\_get**

函数dac\_interrupt\_flag\_get描述见下表:

**表 3-207. 函数 **dac\_interrupt\_flag\_get****

|                                 |  |
|---------------------------------|--|
| <b>函数名称</b>                     | dac_interrupt_flag_get   |
| <b>函数原型</b>                     | FlagStatus dac_interrupt_flag_get(uint32_t dac_periph, uint32_t int_flag); |
| <b>功能描述</b>                     | DAC中断标志位获取   |
| <b>先决条件</b>                     | -  |
| <b>被调用函数</b>                    | -  |
| <b>输入参数{in}</b>                 |  |
| <b>dac_periph</b>               | DAC外设  |
| <i>DACx</i>                     | DAC外设选择 (x = 0)  |
| <b>输入参数{in}</b>                 |  |
| <b>int_flag</b>                 | DAC中断标志位   |
| <i>DAC_INT_FLAG_D<br/>DUDR0</i> | DACx_OUT0 DMA欠载中断标志位   |
| <b>输出参数{out}</b>                |  |
| -                               | -  |
| <b>返回值</b>                      |  |
| <b>FlagStatus</b>               | DAC中断状态 (SET或RESET)  |

例如:

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

```
FlagStatus flag;
```

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

**函数 dac\_interrupt\_flag\_clear**

函数dac\_interrupt\_flag\_clear描述见下表：

**表 3-208. 函数 dac\_interrupt\_flag\_clear**

|                         |  |
|-------------------------|--|
| 函数名称                    | dac_interrupt_flag_clear   |
| 函数原型                    | void dac_interrupt_flag_clear(uint32_t dac_periph, uint32_t int_flag); |
| 功能描述                    | DAC中断标志位清除   |
| 先决条件                    | -  |
| 被调用函数                   | -  |
| 输入参数{in}                |  |
| dac_periph              | DAC外设  |
| DACx                    | DAC外设选择 (x = 0)  |
| 输入参数{in}                |  |
| int_flag                | DAC中断标志位   |
| DAC_INT_FLAG_D<br>DUDR0 | DACx_OUT0 DMA欠载中断标志位   |
| 输出参数{out}               |  |
| -                       | -  |
| 返回值                     |  |
| -                       | -  |

例如：

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

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

## 3.9. DMA/DMAMUX

DMA控制器提供了一种硬件的方式在外设和存储器之间或者存储器和存储器之间传输数据，而无需CPU的介入，从而使CPU可以专注在处理其他系统功能上。章节[3.9.1](#)描述了DMA的寄存器列表，章节[3.9.2](#)对DMA库函数进行说明。

DMAMUX是DMA请求的传输调度器。可编程的DMA请求多路复用器DMAMUX，可在外设和DMA控制器之间路由DMA请求线路，或者DMAMUX也可以将可编程事件连入到输入触发信号上，作为一个DMAMUX请求发生器，再由DMAMUX请求路由器在DMAMUX请求生成器产生的DMA请求和DMA控制器之间路由DMA请求线路。章节[3.9.1](#)描述了DMAMUX的寄存器列表，章节[3.9.2](#)对DMAMUX库函数进行说明。

### 3.9.1. 外设寄存器说明

DMA寄存器列表如下表所示：



表 3-209. DMA 寄存器

| 寄存器名称                    | 寄存器描述        |
|--------------------------|--------------|
| DMA_INTF                 | 中断标志位寄存器     |
| DMA_INTC                 | 中断标志位清除寄存器   |
| DMA_CHxCTL<br>(x=0..6)   | 通道x控制寄存器     |
| DMA_CHxCNT<br>(x=0..6)   | 通道x计数寄存器     |
| DMA_CHxPADDR<br>(x=0..6) | 通道x外设基地址寄存器  |
| DMA_CHxMADDR<br>(x=0..6) | 通道x存储器基地址寄存器 |

DMAMUX寄存器列表如下表所示：

表 3-210. DMAMUX 寄存器

| 寄存器名称                          | 寄存器描述            |
|--------------------------------|------------------|
| DMAMUX_RM_CHx<br>CFG (x=0..11) | 请求路由通道x配置寄存器     |
| DMAMUX_RM_INT<br>F             | 请求路由通道中断标志位寄存器   |
| DMAMUX_RM_INT<br>C             | 请求路由通道中断标志位清除寄存器 |
| DMAMUX_RG_CHx<br>CFG (x=0..3)  | 请求生成通道x配置寄存器     |
| DMAMUX_RG_INT<br>F             | 请求生成通道中断标志位寄存器   |
| DMAMUX_RG_INT<br>C             | 请求生成通道中断标志位清除寄存器 |

### 3.9.2. 外设库函数说明

DMA库函数列表如下表所示：

表 3-211. DMA 库函数

| 库函数名称                        | 库函数描述               |
|------------------------------|---------------------|
| dma_deinit                   | 复位外设DMA通道x的所有寄存器    |
| dma_struct_para_init         | 将DMA结构体中所有参数初始化为默认值 |
| dma_init                     | 初始化外设DMA的通道x        |
| dma_circulation_enable       | 使能DMA循环模式           |
| dma_circulation_disable      | 禁能DMA循环模式           |
| dma_memory_to_memory_enable  | 使能存储器到存储器DMA传输      |
| dma_memory_to_memory_disable | 禁能存储器到存储器DMA传输      |
| dma_channel_enable           | 使能DMA通道x传输          |

| 库函数名称                                      | 库函数描述                    |
|--|--------------------------|
| <code>dma_channel_disable</code>           | 禁能DMA通道x传输               |
| <code>dma_periph_address_config</code>     | 配置DMA通道x传输的外设基地址         |
| <code>dma_memory_address_config</code>     | 配置DMA通道x传输的存储器基地址        |
| <code>dma_transfer_number_config</code>    | 配置DMA通道x还有多少数据要传输        |
| <code>dma_transfer_number_get</code>       | 获取DMA通道x还有多少数据要传输        |
| <code>dma_priority_config</code>           | 配置DMA通道x的传输软件优先级         |
| <code>dma_memory_width_config</code>       | 配置DMA通道x传输的存储器数据         |
| <code>dma_periph_width_config</code>       | 配置DMA通道x传输的外设数据宽度        |
| <code>dma_memory_increase_enable</code>    | 使能DMA通道x传输的存储器地址生成算法增量模式 |
| <code>dma_memory_increase_disable</code>   | 禁能DMA通道x传输的存储器地址生成算法增量模式 |
| <code>dma_periph_increase_enable</code>    | 使能DMA通道x传输的外设地址生成算法增量模式  |
| <code>dma_periph_increase_disable</code>   | 禁能DMA通道x传输的外设地址生成算法增量模式  |
| <code>dma_transfer_direction_config</code> | 配置DMA通道x的传输方向            |
| <code>dma_flag_get</code>                  | 获取DMA通道x标志位状态            |
| <code>dma_flag_clear</code>                | 清除DMA通道x标志位状态            |
| <code>dma_interrupt_enable</code>          | 使能DMA通道x中断               |
| <code>dma_interrupt_disable</code>         | 禁能DMA通道x中断               |
| <code>dma_interrupt_flag_get</code>        | 获取DMA通道x中断标志位状态          |
| <code>dma_interrupt_flag_clear</code>      | 清除DMA通道x中断标志位状态          |

DMAMUX库函数列表如下表所示：

**表 3-212. DMAMUX 库函数**

| 库函数名称   | 库函数描述                      |
|---|----------------------------|
| <code>dmamux_sync_struct_para_init</code>             | 将DMAMUX同步结构体中所有参数初始化为默认值   |
| <code>dmamux_synchronization_init</code>              | 初始化DMAMUX同步结构体通道x          |
| <code>dmamux_synchronization_enable</code>            | 使能DMAMUX同步模式               |
| <code>dmamux_synchronization_disable</code>           | 禁能DMAMUX同步模式               |
| <code>dmamux_event_generation_enable</code>           | 使能DMAMUX事件输出               |
| <code>dmamux_event_generation_disable</code>          | 禁能DMAMUX事件输出               |
| <code>dmamux_gen_struct_para_init</code>              | 将DMAMUX请求生成结构体中所有参数初始化为默认值 |
| <code>dmamux_request_generator_init</code>            | 初始化DMAMUX请求生成结构体通道x        |
| <code>dmamux_request_generator_channel_enable</code>  | 使能DMAMUX请求生成通道x            |
| <code>dmamux_request_generator_channel_disable</code> | 禁能DMAMUX请求生成通道x            |
| <code>dmamux_synchronization_polarity_config</code>   | 配置DMAMUX同步输入的有效边沿          |
| <code>dmamux_request_forward_number_config</code>     | 配置DMAMUX通道x要传输多少个DMA请求     |
| <code>dmamux_sync_id_config</code>                    | 配置DMAMUX同步输入标识             |
| <code>dmamux_request_id_config</code>                 | 配置DMAMUX请求路由通道输入标识         |

| 库函数名称                                 | 库函数描述                |
|---------------------------------------|----------------------|
| dmamux_trigger_polarity_config        | 配置DMAMUX触发输入的有效边沿    |
| dmamux_request_generate_number_config | 配置DMAMUX请求生成器生成请求的数量 |
| dmamux_trigger_id_config              | 配置DMAMUX触发输入标识       |
| dmamux_flag_get                       | 获取DMAMUX通道x标志位状态     |
| dmamux_flag_clear                     | 清除DMAMUX通道x标志位状态     |
| dmamux_interrupt_enable               | 使能DMAMUX通道x中断        |
| dmamux_interrupt_disable              | 禁能DMAMUX通道x中断        |
| dmamux_interrupt_flag_get             | 获取DMAMUX通道x中断标志位状态   |
| dmamux_interrupt_flag_clear           | 清除DMAMUX通道x中断标志位状态   |

### 结构体 dma\_parameter\_struct

表 3-213. 结构体 dma\_parameter\_struct

| 成员名称         | 功能描述         |
|--------------|--------------|
| periph_addr  | 外设基地址        |
| periph_width | 外设数据传输宽度     |
| memory_addr  | 存储器基地址       |
| memory_width | 存储器数据传输宽度    |
| number       | DMA通道数据传输数量  |
| priority     | DMA通道传输软件优先级 |
| periph_inc   | 外设地址生成算法模式   |
| memory_inc   | 存储器地址生成算法模式  |
| direction    | DMA通道数据传输方向  |
| request      | 请求路由通道输入标识   |

### 结构体 dmamux\_sync\_parameter\_struct

表 3-214. 结构体 dmamux\_sync\_parameter\_struct

| 成员名称           | 功能描述        |
|----------------|-------------|
| sync_id        | 同步输入标识      |
| sync_polarity  | 同步输入信号有效边沿  |
| request_number | 要传输的DMA请求数量 |

### 结构体 dmamux\_gen\_parameter\_struct

表 3-215. 结构体 dmamux\_gen\_parameter\_struct

| 成员名称             | 功能描述                  |
|------------------|-----------------------|
| trigger_id       | 触发输入标识                |
| trigger_polarity | DMAMUX请求生成器触发输入信号有效边沿 |
| request_number   | 要生成的DMA请求数量           |

## 枚举 dma\_channel\_enum

表 3-216. 枚举 dma\_channel\_enum

| 成员名称    | 功能描述   |
|---------|--------|
| DMA_CH0 | DMA通道0 |
| DMA_CH1 | DMA通道1 |
| DMA_CH2 | DMA通道2 |
| DMA_CH3 | DMA通道3 |
| DMA_CH4 | DMA通道4 |
| DMA_CH5 | DMA通道5 |
| DMA_CH6 | DMA通道6 |

## 枚举 dmamux\_multiplexer\_channel\_enum

表 3-217. 枚举 dmamux\_multiplexer\_channel\_enum

| 成员名称               | 功能描述           |
|--------------------|----------------|
| DMAMUX_MUXCH<br>0  | DMAMUX请求路由通道0  |
| DMAMUX_MUXCH<br>1  | DMAMUX请求路由通道1  |
| DMAMUX_MUXCH<br>2  | DMAMUX请求路由通道2  |
| DMAMUX_MUXCH<br>3  | DMAMUX请求路由通道3  |
| DMAMUX_MUXCH<br>4  | DMAMUX请求路由通道4  |
| DMAMUX_MUXCH<br>5  | DMAMUX请求路由通道5  |
| DMAMUX_MUXCH<br>6  | DMAMUX请求路由通道6  |
| DMAMUX_MUXCH<br>7  | DMAMUX请求路由通道7  |
| DMAMUX_MUXCH<br>8  | DMAMUX请求路由通道8  |
| DMAMUX_MUXCH<br>9  | DMAMUX请求路由通道9  |
| DMAMUX_MUXCH<br>10 | DMAMUX请求路由通道10 |
| DMAMUX_MUXCH<br>11 | DMAMUX请求路由通道11 |

## 枚举 dmamux\_generator\_channel\_enum

表 3-218. 枚举 dmamux\_generator\_channel\_enum

| 成员名称          | 功能描述          |
|---------------|---------------|
| DMAMUX_GENCH0 | DMAMUX请求生成通道0 |
| DMAMUX_GENCH1 | DMAMUX请求生成通道1 |
| DMAMUX_GENCH2 | DMAMUX请求生成通道2 |
| DMAMUX_GENCH3 | DMAMUX请求生成通道3 |

## 枚举 dmamux\_interrupt\_enum

表 3-219. 枚举 dmamux\_interrupt\_enum

| 成员名称                      | 功能描述                 |
|---------------------------|----------------------|
| DMAMUX_INT_MU<br>XCH0_SO  | DMAMUX请求路由通道0同步溢出中断  |
| DMAMUX_INT_MU<br>XCH1_SO  | DMAMUX请求路由通道1同步溢出中断  |
| DMAMUX_INT_MU<br>XCH2_SO  | DMAMUX请求路由通道2同步溢出中断  |
| DMAMUX_INT_MU<br>XCH3_SO  | DMAMUX请求路由通道3同步溢出中断  |
| DMAMUX_INT_MU<br>XCH4_SO  | DMAMUX请求路由通道4同步溢出中断  |
| DMAMUX_INT_MU<br>XCH5_SO  | DMAMUX请求路由通道5同步溢出中断  |
| DMAMUX_INT_MU<br>XCH6_SO  | DMAMUX请求路由通道6同步溢出中断  |
| DMAMUX_INT_MU<br>XCH7_SO  | DMAMUX请求路由通道7同步溢出中断  |
| DMAMUX_INT_MU<br>XCH8_SO  | DMAMUX请求路由通道8同步溢出中断  |
| DMAMUX_INT_MU<br>XCH9_SO  | DMAMUX请求路由通道9同步溢出中断  |
| DMAMUX_INT_MU<br>XCH10_SO | DMAMUX请求路由通道10同步溢出中断 |
| DMAMUX_INT_MU<br>XCH11_SO | DMAMUX请求路由通道11同步溢出中断 |
| DMAMUX_INT_GE<br>NCH0_TO  | DMAMUX请求生成通道0触发溢出中断  |
| DMAMUX_INT_GE<br>NCH1_TO  | DMAMUX请求生成通道1触发溢出中断  |
| DMAMUX_INT_GE<br>NCH2_TO  | DMAMUX请求生成通道2触发溢出中断  |

|                          |                     |
|--------------------------|---------------------|
| DMAMUX_INT_GE<br>NCH3_TO | DMAMUX请求生成通道3触发溢出中断 |
|--------------------------|---------------------|

### 枚举 dmamux\_flag\_enum

表 3-220. 枚举 dmamux\_flag\_enum

| 成员名称                       | 功能描述                 |
|----------------------------|----------------------|
| DMAMUX_FLAG_M<br>UXCH0_SO  | DMAMUX请求路由通道0同步溢出标志  |
| DMAMUX_FLAG_M<br>UXCH1_SO  | DMAMUX请求路由通道1同步溢出标志  |
| DMAMUX_FLAG_M<br>UXCH2_SO  | DMAMUX请求路由通道2同步溢出标志  |
| DMAMUX_FLAG_M<br>UXCH3_SO  | DMAMUX请求路由通道3同步溢出标志  |
| DMAMUX_FLAG_M<br>UXCH4_SO  | DMAMUX请求路由通道4同步溢出标志  |
| DMAMUX_FLAG_M<br>UXCH5_SO  | DMAMUX请求路由通道5同步溢出标志  |
| DMAMUX_FLAG_M<br>UXCH6_SO  | DMAMUX请求路由通道6同步溢出标志  |
| DMAMUX_FLAG_M<br>UXCH7_SO  | DMAMUX请求路由通道7同步溢出标志  |
| DMAMUX_FLAG_M<br>UXCH8_SO  | DMAMUX请求路由通道8同步溢出标志  |
| DMAMUX_FLAG_M<br>UXCH9_SO  | DMAMUX请求路由通道9同步溢出标志  |
| DMAMUX_FLAG_M<br>UXCH10_SO | DMAMUX请求路由通道10同步溢出标志 |
| DMAMUX_FLAG_M<br>UXCH11_SO | DMAMUX请求路由通道11同步溢出标志 |
| DMAMUX_FLAG_G<br>ENCH0_TO  | DMAMUX请求生成通道0触发溢出标志  |
| DMAMUX_FLAG_G<br>ENCH1_TO  | DMAMUX请求生成通道1触发溢出标志  |
| DMAMUX_FLAG_G<br>ENCH2_TO  | DMAMUX请求生成通道2触发溢出标志  |
| DMAMUX_FLAG_G<br>ENCH3_TO  | DMAMUX请求生成通道3触发溢出标志  |

### 枚举 dmamux\_interrupt\_flag\_enum

表 3-221. 枚举 dmamux\_interrupt\_flag\_enum

| 成员名称 | 功能描述 |
|------|------|
|------|------|

|                            |                        |
|----------------------------|------------------------|
| DMAMUX_INT_FLAG_MUXCH0_SO  | DMAMUX请求路由通道0同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH1_SO  | DMAMUX请求路由通道1同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH2_SO  | DMAMUX请求路由通道2同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH3_SO  | DMAMUX请求路由通道3同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH4_SO  | DMAMUX请求路由通道4同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH5_SO  | DMAMUX请求路由通道5同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH6_SO  | DMAMUX请求路由通道6同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH7_SO  | DMAMUX请求路由通道7同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH8_SO  | DMAMUX请求路由通道8同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH9_SO  | DMAMUX请求路由通道9同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH10_SO | DMAMUX请求路由通道10同步溢出中断标志 |
| DMAMUX_INT_FLAG_MUXCH11_SO | DMAMUX请求路由通道11同步溢出中断标志 |
| DMAMUX_INT_FLAG_GENCH0_TO  | DMAMUX请求生成通道0触发溢出中断标志  |
| DMAMUX_INT_FLAG_GENCH1_TO  | DMAMUX请求生成通道1触发溢出中断标志  |
| DMAMUX_INT_FLAG_GENCH2_TO  | DMAMUX请求生成通道2触发溢出中断标志  |
| DMAMUX_INT_FLAG_GENCH3_TO  | DMAMUX请求生成通道3触发溢出中断标志  |

## 函数 dma\_deinit

函数 dma\_deinit 描述见下表：

**表 3-222. 函数 dma\_deinit**

|          |  |
|----------|--|
| 函数名称     | dma_deinit   |
| 函数原型     | void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx); |
| 功能描述     | 复位DMA通道x的所有寄存器   |
| 先决条件     | 无  |
| 被调用函数    | 无  |
| 输入参数{in} |  |

|                        |  |
|------------------------|--|
| <b>dma_periph</b>      | DMA外设  |
| <i>DMAx(x=0,1)</i>     | DMA外设选择  |
| <b>输入参数{in}</b>        |  |
| <b>channelx</b>        | DMA通道  |
| <i>DMA_CHx(x=0..6)</i> | DMA通道选择, 参考 <a href="#">表3-216. 枚举dma_channel_enum</a> |
| <b>输出参数{out}</b>       |  |
| -                      | -  |
| <b>返回值</b>             |  |
| -                      | -  |

例如:

```
/* deinitialize DMA0 channel 0 registers */
dma_deinit(DMA0, DMA_CH0);
```

### 函数 dma\_struct\_para\_init

函数 dma\_struct\_para\_init 描述见下表:

**表 3-223. 函数 dma\_struct\_para\_init**

|                    |   |
|--------------------|---|
| <b>函数名称</b>        | dma_struct_para_init  |
| <b>函数原型</b>        | void dma_struct_para_init(dma_parameter_struct* init_struct);               |
| <b>功能描述</b>        | 将DMA结构体中所有参数初始化为默认值   |
| <b>先决条件</b>        | 无   |
| <b>被调用函数</b>       | 无   |
| <b>输入参数{in}</b>    |   |
| -                  | -   |
| <b>输出参数{out}</b>   |   |
| <b>init_struct</b> | 用于初始化DMA通道的初始化数据的结构体地址, 参考 <a href="#">表3-213. 结构体 dma_parameter_struct</a> |
| <b>返回值</b>         |   |
| -                  | -   |

例如:

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

### 函数 dma\_init

函数 dma\_init 描述见下表:

**表 3-224. 函数 dma\_init**

|             |   |
|-------------|---|
| <b>函数名称</b> | dma_init  |
| <b>函数原型</b> | void dma_init(uint32_t dma_periph, dma_channel_enum channelx, |



|                              |  |
|------------------------------|--|
|                              | <code>dma_parameter_struct* init_struct);</code>                           |
| 功能描述                         | 初始化DMA通道x  |
| 先决条件                         | 无  |
| 被调用函数                        | 无  |
| 输入参数{in}                     |  |
| <code>dma_periph</code>      | DMA外设  |
| <code>DMAx(x=0,1)</code>     | DMA外设选择  |
| 输入参数{in}                     |  |
| <code>channelx</code>        | DMA通道  |
| <code>DMA_CHx(x=0..6)</code> | DMA通道选择, 参考 <a href="#">表3-216. 枚举dma_channel_enum</a>                     |
| 输入参数{in}                     |  |
| <code>init_struct</code>     | 用于初始化DMA通道的初始化数据的结构体地址, 参考 <a href="#">表3-213. 结构体dma_parameter_struct</a> |
| 输出参数{out}                    |  |
| -                            | -  |
| 返回值                          |  |
| -                            | -  |

例如:

```

/* DMA0 channel 0 initialize */
dma_parameter_struct dma_init_struct;

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

```

### 函数 dma\_circulation\_enable

函数 dma\_circulation\_enable 描述见下表:

表 3-225. 函数 dma\_circulation\_enable

|      |   |
|------|---|
| 函数名称 | <code>dma_circulation_enable</code>   |
| 函数原型 | <code>void dma_circulation_enable(uint32_t dma_periph, dma_channel_enum channelx);</code> |
| 功能描述 | DMA循环模式使能   |
| 先决条件 | 相应通道使能位CHEN需为0  |

|                        |  |
|------------------------|--|
| 被调用函数                  | 无  |
| 输入参数{in}               |  |
| <b>dma_periph</b>      | DMA外设  |
| <i>DMAx(x=0,1)</i>     | DMA外设选择  |
| 输入参数{in}               |  |
| <b>channelx</b>        | DMA通道  |
| <i>DMA_CHx(x=0..6)</i> | DMA通道选择, 参考 <a href="#">表3-216. 枚举dma_channel_enum</a> |
| 输出参数{out}              |  |
| -                      | -  |
| 返回值                    |  |
| -                      | -  |

例如:

```
/* enable DMA0 channel 0 circulation mode */
dma_circulation_enable(DMA0, DMA_CH0);
```

### 函数 dma\_circulation\_disable

函数 dma\_circulation\_disable 描述见下表:

表 3-226. 函数 dma\_circulation\_disable

|                        |   |
|------------------------|---|
| 函数名称                   | dma_circulation_disable   |
| 函数原型                   | void dma_circulation_disable(uint32_t dma_periph, dma_channel_enum channelx); |
| 功能描述                   | DMA循环模式禁能   |
| 先决条件                   | 相应通道使能位CHEN需为0  |
| 被调用函数                  | 无   |
| 输入参数{in}               |   |
| <b>dma_periph</b>      | DMA外设   |
| <i>DMAx(x=0,1)</i>     | DMA外设选择   |
| 输入参数{in}               |   |
| <b>channelx</b>        | DMA通道   |
| <i>DMA_CHx(x=0..6)</i> | DMA通道选择, 参考 <a href="#">表3-216. 枚举dma_channel_enum</a>                        |
| 输出参数{out}              |   |
| -                      | -   |
| 返回值                    |   |
| -                      | -   |

例如:

```
/* disable DMA0 channel 0 circulation mode */
dma_circulation_disable(DMA0, DMA_CH0);
```

## 函数 dma\_memory\_to\_memory\_enable

函数 dma\_memory\_to\_memory\_enable 描述见下表：

表 3-227. 函数 dma\_memory\_to\_memory\_enable

|                 |  |
|-----------------|--|
| 函数名称            | dma_memory_to_memory_enable  |
| 函数原型            | void dma_memory_to_memory_enable(uint32_t dma_periph,<br>dma_channel_enum channelx); |
| 功能描述            | 存储器到存储器DMA传输使能   |
| 先决条件            | 相应通道使能位CHEN需为0   |
| 被调用函数           | 无  |
| 输入参数{in}        |  |
| dma_periph      | DMA外设  |
| DMAx(x=0,1)     | DMA外设选择  |
| 输入参数{in}        |  |
| channelx        | DMA通道  |
| DMA_CHx(x=0..6) | DMA通道选择, 参考 <a href="#">表3-216. 枚举dma_channel_enum</a>                               |
| 输出参数{out}       |  |
| -               | -  |
| 返回值             |  |
| -               | -  |

例如：

```
/* enable DMA0 channel 0 memory to memory mode */
dma_memory_to_memory_enable(DMA0, DMA_CH0);
```

## 函数 dma\_memory\_to\_memory\_disable

函数 dma\_memory\_to\_memory\_disable 描述见下表：

表 3-228. 函数 dma\_memory\_to\_memory\_disable

|                 |   |
|-----------------|---|
| 函数名称            | dma_memory_to_memory_disable  |
| 函数原形            | void dma_memory_to_memory_disable(uint32_t dma_periph,<br>dma_channel_enum channelx); |
| 功能描述            | 存储器到存储器DMA传输禁能  |
| 先决条件            | 相应通道使能位CHEN需为0  |
| 被调用函数           | 无   |
| 输入参数{in}        |   |
| dma_periph      | DMA外设   |
| DMAx(x=0,1)     | DMA外设选择   |
| 输入参数{in}        |   |
| channelx        | DMA通道   |
| DMA_CHx(x=0..6) | DMA通道选择, 参考 <a href="#">表3-216. 枚举dma_channel_enum</a>                                |
| 输出参数{out}       |   |

|     |   |
|-----|---|
| -   | - |
| 返回值 |   |
| -   | - |

例如：

```
/* disable DMA0 channel 0 memory to memory mode */
dma_memory_to_memory_disable(DMA0, DMA_CH0);
```

### 函数 dma\_channel\_enable

函数 dma\_channel\_enable 描述见下表：

表 3-229. 函数 dma\_channel\_enable

|                 |  |
|-----------------|--|
| 函数名称            | dma_channel_enable   |
| 函数原型            | void dma_channel_enable(uint32_t dma_periph, dma_channel_enum channelx); |
| 功能描述            | DMA通道x传输使能   |
| 先决条件            | 无  |
| 被调用函数           | 无  |
| 输入参数{in}        |  |
| dma_periph      | DMA外设  |
| DMAx(x=0,1)     | DMA外设选择  |
| 输入参数{in}        |  |
| channelx        | DMA通道  |
| DMA_CHx(x=0..6) | DMA通道选择，参考 <a href="#">表3-216. 枚举dma_channel_enum</a>                    |
| 输出参数{out}       |  |
| -               | -  |
| 返回值             |  |
| -               | -  |

例如：

```
/* enable DMA0 channel 0 */
dma_channel_enable(DMA0, DMA_CH0)
```

### 函数 dma\_channel\_disable

函数 dma\_channel\_disable 描述见下表：

表 3-230. 函数 dma\_channel\_disable

|      |   |
|------|---|
| 函数名称 | dma_channel_disable   |
| 函数原型 | void dma_channel_disable(uint32_t dma_periph, dma_channel_enum channelx); |
| 功能描述 | DMA通道x传输禁能  |
| 先决条件 | 无   |

|                        |  |
|------------------------|--|
| 被调用函数                  | 无  |
| 输入参数{in}               |  |
| <b>dma_periph</b>      | DMA外设  |
| <i>DMAx(x=0,1)</i>     | DMA外设选择  |
| 输入参数{in}               |  |
| <b>channelx</b>        | DMA通道  |
| <i>DMA_CHx(x=0..6)</i> | DMA通道选择, 参考 <a href="#">表3-216. 枚举dma_channel_enum</a> |
| 输出参数{out}              |  |
| -                      | -  |
| 返回值                    |  |
| -                      | -  |

例如:

```
/* disable DMA0 channel 0 */
dma_channel_disable(DMA0, DMA_CH0);
```

### 函数 dma\_periph\_address\_config

函数 dma\_periph\_address\_config 描述见下表:

表 3-231. 函数 dma\_periph\_address\_config

|                        |   |
|------------------------|---|
| 函数名称                   | dma_periph_address_config   |
| 函数原型                   | void dma_periph_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address); |
| 功能描述                   | DMA通道x传输的外设基地址配置  |
| 先决条件                   | 相应通道使能位CHEN需为0  |
| 被调用函数                  | 无   |
| 输入参数{in}               |   |
| <b>dma_periph</b>      | DMA外设   |
| <i>DMAx(x=0,1)</i>     | DMA外设选择   |
| 输入参数{in}               |   |
| <b>channelx</b>        | DMA通道   |
| <i>DMA_CHx(x=0..6)</i> | DMA通道选择, 参考 <a href="#">表3-216. 枚举dma_channel_enum</a>  |
| 输入参数{in}               |   |
| <b>address</b>         | 外设基地址   |
| 输出参数{out}              |   |
| -                      | -   |
| 返回值                    |   |
| -                      | -   |

例如:

```
/* configure DMA0 channel 0 periph address */
#define BANK0_WRITE_START_ADDR ((uint32_t)0x08004000)
```

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

### 函数 dma\_memory\_address\_config

函数 dma\_memory\_address\_config 描述见下表：

表 3-232. 函数 dma\_memory\_address\_config

|                 |   |
|-----------------|---|
| 函数名称            | dma_memory_address_config   |
| 函数原型            | void dma_memory_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address); |
| 功能描述            | DMA通道x传输的存储器基地址配置   |
| 先决条件            | 相应通道使能位CHEN需为0  |
| 被调用函数           | 无   |
| 输入参数{in}        |   |
| dma_periph      | DMA外设   |
| DMAx(x=0,1)     | DMA外设选择   |
| 输入参数{in}        |   |
| channelx        | DMA通道   |
| DMA_CHx(x=0..6) | DMA通道选择，参考 <a href="#">表3-216. 枚举dma_channel_enum</a>   |
| 输入参数{in}        |   |
| address         | 存储器基地址，0 – 0xFFFFFFFF   |
| 输出参数{out}       |   |
| -               | -   |
| 返回值             |   |
| -               | -   |

例如：

```
/* configure DMA0 channel 0 memory address */
uint8_t g_destbuf[TRANSFER_NUM];
dma_memory_address_config(DMA0, DMA_CH0, (uint32_t) g_destbuf);
```

### 函数 dma\_transfer\_number\_config

函数 dma\_transfer\_number\_config 描述见下表：

表 3-233. 函数 dma\_transfer\_number\_config

|            |   |
|------------|---|
| 函数名称       | dma_transfer_number_config  |
| 函数原型       | void dma_transfer_number_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t number); |
| 功能描述       | 配置DMA通道x还有多少数据要传输   |
| 先决条件       | 相应通道使能位CHEN需为0  |
| 被调用函数      | 无   |
| 输入参数{in}   |   |
| dma_periph | DMA外设   |

|                        |  |
|------------------------|--|
| <i>DMAx(x=0,1)</i>     | DMA外设选择  |
| 输入参数{in}               |  |
| <b>channelx</b>        | DMA通道  |
| <i>DMA_CHx(x=0..6)</i> | DMA通道选择, 参考 <a href="#">表3-216. 枚举dma_channel_enum</a> |
| 输入参数{in}               |  |
| <b>number</b>          | 数据传输数量 (0x0 – 0xFFFF)                                  |
| 输出参数{out}              |  |
| -                      | -  |
| 返回值                    |  |
| -                      | -  |

例如:

```
/* configure DMA0 channel 0 transfer number */

#define TRANSFER_NUM                0x400
dma_transfer_number_config(DMA0, DMA_CH0, TRANSFER_NUM);
```

### 函数 dma\_transfer\_number\_get

函数 dma\_transfer\_number\_get 描述见下表:

表 3-234. 函数 dma\_transfer\_number\_get

|                        |   |
|------------------------|---|
| 函数名称                   | dma_transfer_number_get   |
| 函数原型                   | uint32_t dma_transfer_number_get(uint32_t dma_periph, dma_channel_enum channelx); |
| 功能描述                   | 获取DMA通道x还有多少数据要传输   |
| 先决条件                   | 相应通道使能位CHEN需为0  |
| 被调用函数                  | 无   |
| 输入参数{in}               |   |
| <b>dma_periph</b>      | DMA外设   |
| <i>DMAx(x=0,1)</i>     | DMA外设选择   |
| 输入参数{in}               |   |
| <b>channelx</b>        | DMA通道   |
| <i>DMA_CHx(x=0..6)</i> | DMA通道选择, 参考 <a href="#">表3-216. 枚举dma_channel_enum</a>                            |
| 输出参数{out}              |   |
| -                      | -   |
| 返回值                    |   |
| <b>uint32_t</b>        | DMA数据传输剩余数量 (0x0 – 0xFFFF)  |

例如:

```
/* get DMA0 channel 0 transfer number */

uint32_t number = 0;

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

## 函数 dma\_priority\_config

函数 dma\_priority\_config 描述见下表：

表 3-235. 函数 dma\_priority\_config

|                         |  |
|-------------------------|--|
| 函数名称                    | dma_priority_config  |
| 函数原型                    | void dma_priority_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t priority); |
| 功能描述                    | DMA通道x的传输软件优先级配置   |
| 先决条件                    | 相应通道使能位CHEN需为0   |
| 被调用函数                   | 无  |
| 输入参数{in}                |  |
| dma_periph              | DMA外设  |
| DMAx(x=0,1)             | DMA外设选择  |
| 输入参数{in}                |  |
| channelx                | DMA通道  |
| DMA_CHx(x=0..6)         | DMA通道选择，参考 <a href="#">表3-216. 枚举dma_channel_enum</a>  |
| 输入参数{in}                |  |
| priority                | DMA通道软件优先级   |
| DMA_PRIORITY_LOW        | 低优先级   |
| DMA_PRIORITY_MEDIUM     | 中优先级   |
| DMA_PRIORITY_HIGH       | 高优先级   |
| DMA_PRIORITY_ULTRA_HIGH | 极高优先级  |
| 输出参数{out}               |  |
| -                       | -  |
| 返回值                     |  |
| -                       | -  |

例如：

```
/* configure DMA0 channel 0 priority */
dma_priority_config(DMA0, DMA_CH0, DMA_PRIORITY_ULTRA_HIGH);
```

## 函数 dma\_memory\_width\_config

函数 dma\_memory\_width\_config 描述见下表：

表 3-236. 函数 dma\_memory\_width\_config

|      |  |
|------|--|
| 函数名称 | dma_memory_width_config  |
| 函数原型 | void dma_memory_width_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t mwidth); |



|                               |  |
|-------------------------------|--|
| 功能描述                          | DMA通道x传输的存储器数据宽度配置                                     |
| 先决条件                          | 相应通道使能位CHEN需为0   |
| 被调用函数                         | 无  |
| 输入参数{in}                      |  |
| <b>dma_periph</b>             | DMA外设  |
| <i>DMAx(x=0,1)</i>            | DMA外设选择  |
| 输入参数{in}                      |  |
| <b>channelx</b>               | DMA通道  |
| <i>DMA_CHx(x=0..6)</i>        | DMA通道选择, 参考 <a href="#">表3-216. 枚举dma_channel_enum</a> |
| 输入参数{in}                      |  |
| <b>mwidth</b>                 | 存储器数据传输宽度  |
| <i>DMA_MEMORY_WIDTH_8BIT</i>  | 8位数据传输宽度   |
| <i>DMA_MEMORY_WIDTH_16BIT</i> | 16位数据传输宽度  |
| <i>DMA_MEMORY_WIDTH_32BIT</i> | 32位数据传输宽度  |
| 输出参数{out}                     |  |
| -                             | -  |
| 返回值                           |  |
| -                             | -  |

例如:

```
/* configure DMA0 channel 0 memory width */
dma_memory_width_config(DMA0, DMA_CH0, DMA_MEMORY_WIDTH_8BIT);
```

### 函数 dma\_periph\_width\_config

函数 dma\_periph\_width\_config 描述见下表:

表 3-237. 函数 dma\_periph\_width\_config

|                        |  |
|------------------------|--|
| 函数名称                   | dma_periph_width_config  |
| 函数原型                   | void dma_periph_width_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t pwidth); |
| 功能描述                   | DMA通道x传输的外设数据宽度配置  |
| 先决条件                   | 相应通道使能位CHEN需为0   |
| 被调用函数                  | 无  |
| 输入参数{in}               |  |
| <b>dma_periph</b>      | DMA外设  |
| <i>DMAx(x=0,1)</i>     | DMA外设选择  |
| 输入参数{in}               |  |
| <b>channelx</b>        | DMA通道  |
| <i>DMA_CHx(x=0..6)</i> | DMA通道选择, 参考 <a href="#">表3-216. 枚举dma_channel_enum</a>   |

| 输入参数{in}                          |           |
|-----------------------------------|-----------|
| <b>pwidth</b>                     | 外设数据传输宽度  |
| <i>DMA_PERIPHERAL_WIDTH_8BIT</i>  | 8位数据传输宽度  |
| <i>DMA_PERIPHERAL_WIDTH_16BIT</i> | 16位数据传输宽度 |
| <i>DMA_PERIPHERAL_WIDTH_32BIT</i> | 32位数据传输宽度 |
| 输出参数{out}                         |           |
| -                                 | -         |
| 返回值                               |           |
| -                                 | -         |

例如：

```
/* configure DMA0 channel 0 periph width */
dma_periph_width_config(DMA0, DMA_CH0, DMA_PERIPHERAL_WIDTH_8BIT);
```

### 函数 dma\_memory\_increase\_enable

函数 dma\_memory\_increase\_enable 描述见下表：

表 3-238. 函数 dma\_memory\_increase\_enable

| 函数名称                   | dma_memory_increase_enable   |
|------------------------|--|
| 函数原型                   | void dma_memory_increase_enable(uint32_t dma_periph, dma_channel_enum channelx); |
| 功能描述                   | DMA通道x传输的存储器地址生成算法增量模式使能   |
| 先决条件                   | 相应通道使能位CHEN需为0   |
| 被调用函数                  | 无  |
| 输入参数{in}               |  |
| <b>dma_periph</b>      | DMA外设  |
| <i>DMAx(x=0,1)</i>     | DMA外设选择  |
| 输入参数{in}               |  |
| <b>channelx</b>        | DMA通道  |
| <i>DMA_CHx(x=0..6)</i> | DMA通道选择，参考 <a href="#">表3-216. 枚举dma_channel_enum</a>                            |
| 输出参数{out}              |  |
| -                      | -  |
| 返回值                    |  |
| -                      | -  |

例如：

```
/* enable DMA0 channel 0 memory increase */
dma_memory_increase_enable(DMA0, DMA_CH0);
```

## 函数 dma\_memory\_increase\_disable

函数 dma\_memory\_increase\_disable 描述见下表：

表 3-239. 函数 dma\_memory\_increase\_disable

|                 |   |
|-----------------|---|
| 函数名称            | dma_memory_increase_disable   |
| 函数原型            | void dma_memory_increase_disable(uint32_t dma_periph, dma_channel_enum channelx); |
| 功能描述            | DMA通道x传输的存储器地址生成算法增量模式禁能  |
| 先决条件            | 相应通道使能位CHEN需为0  |
| 被调用函数           | 无   |
| 输入参数{in}        |   |
| dma_periph      | DMA外设   |
| DMAx(x=0,1)     | DMA外设选择   |
| 输入参数{in}        |   |
| channelx        | DMA通道   |
| DMA_CHx(x=0..6) | DMA通道选择, 参考 <a href="#">表3-216. 枚举dma_channel_enum</a>                            |
| 输出参数{out}       |   |
| -               | -   |
| 返回值             |   |
| -               | -   |

例如：

```
/* disable DMA channel 0 memory increase */
```

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

## 函数 dma\_periph\_increase\_enable

函数 dma\_periph\_increase\_enable 描述见下表：

表 3-240. 函数 dma\_periph\_increase\_enable

|                 |  |
|-----------------|--|
| 函数名称            | dma_periph_increase_enable   |
| 函数原型            | void dma_periph_increase_enable(uint32_t dma_periph, dma_channel_enum channelx); |
| 功能描述            | DMA通道x传输的外设地址生成算法增量模式使能  |
| 先决条件            | 相应通道使能位CHEN需为0   |
| 被调用函数           | 无  |
| 输入参数{in}        |  |
| dma_periph      | DMA外设  |
| DMAx(x=0,1)     | DMA外设选择  |
| 输入参数{in}        |  |
| channelx        | DMA通道  |
| DMA_CHx(x=0..6) | DMA通道选择, 参考 <a href="#">表3-216. 枚举dma_channel_enum</a>                           |
| 输出参数{out}       |  |

|     |   |
|-----|---|
| -   | - |
| 返回值 |   |
| -   | - |

例如：

```
/* enable next address increasement algorithm of DMA channel 0 */
dma_periph_increase_enable(DMA0, DMA_CH0);
```

### 函数 dma\_periph\_increase\_disable

函数 dma\_periph\_increase\_disable 描述见下表：

**表 3-241. 函数 dma\_periph\_increase\_disable**

|                 |   |
|-----------------|---|
| 函数名称            | dma_periph_increase_disable   |
| 函数原型            | void dma_periph_increase_disable(uint32_t dma_periph, dma_channel_enum channelx); |
| 功能描述            | DMA通道x传输的外设地址生成算法增量模式禁能   |
| 先决条件            | 相应通道使能位CHEN需为0  |
| 被调用函数           | 无   |
| 输入参数{in}        |   |
| dma_periph      | DMA外设   |
| DMAx(x=0,1)     | DMA外设选择   |
| 输入参数{in}        |   |
| channelx        | DMA通道   |
| DMA_CHx(x=0..6) | DMA通道选择，参考 <a href="#">表3-216. 枚举dma_channel_enum</a>                             |
| 输出参数{out}       |   |
| -               | -   |
| 返回值             |   |
| -               | -   |

例如：

```
/* disable next address increasement algorithm of DMA0 channel 0*/
dma_periph_increase_disable(DMA0, DMA_CH0);
```

### 函数 dma\_transfer\_direction\_config

函数 dma\_transfer\_direction\_config 描述见下表：

**表 3-242. 函数 dma\_transfer\_direction\_config**

|      |   |
|------|---|
| 函数名称 | dma_transfer_direction_config   |
| 函数原型 | void dma_transfer_direction_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t direction); |
| 功能描述 | DMA通道x的传输方向配置   |
| 先决条件 | 相应通道使能位CHEN需为0  |

|                                 |  |
|---------------------------------|--|
| 被调用函数                           | 无  |
| 输入参数{in}                        |  |
| <b>dma_periph</b>               | DMA外设  |
| <i>DMAx(x=0,1)</i>              | DMA外设选择  |
| 输入参数{in}                        |  |
| <b>channelx</b>                 | DMA通道  |
| <i>DMA_CHx(x=0..6)</i>          | DMA通道选择, 参考 <a href="#">表3-216. 枚举dma_channel_enum</a> |
| 输入参数{in}                        |  |
| <b>direction</b>                | 数据传输方向   |
| <i>DMA_PERIPHERAL_TO_MEMORY</i> | 读取外设中数据, 写入存储器   |
| <i>DMA_MEMORY_TO_PERIPHERAL</i> | 读取存储器中数据, 写入外设   |
| 输出参数{out}                       |  |
| -                               | -  |
| 返回值                             |  |
| -                               | -  |

例如:

```
/* configure DMA0 channel 0 transfer direction*/
dma_transfer_direction_config(DMA0, DMA_CH0, DMA_PERIPHERAL_TO_MEMORY);
```

## 函数 dma\_flag\_get

函数 dma\_flag\_get 描述见下表:

表 3-243. 函数 dma\_flag\_get

|                        |   |
|------------------------|---|
| 函数名称                   | dma_flag_get  |
| 函数原型                   | FlagStatus dma_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag); |
| 功能描述                   | 获取DMA通道x标志位状态   |
| 先决条件                   | 无   |
| 被调用函数                  | 无   |
| 输入参数{in}               |   |
| <b>dma_periph</b>      | DMA外设   |
| <i>DMAx(x=0,1)</i>     | DMA外设选择   |
| 输入参数{in}               |   |
| <b>channelx</b>        | DMA通道   |
| <i>DMA_CHx(x=0..6)</i> | DMA通道选择, 参考 <a href="#">表3-216. 枚举dma_channel_enum</a>                                  |
| 输入参数{in}               |   |
| <b>flag</b>            | DMA标志   |
| <i>DMA_FLAG_G</i>      | DMA通道全局中断标志   |
| <i>DMA_FLAG_FTF</i>    | DMA通道传输完成标志   |

|                     |              |
|---------------------|--------------|
| <i>DMA_FLAG_HTF</i> | DMA通道半传输完成标志 |
| <i>DMA_FLAG_ERR</i> | DMA通道错误标志    |
| 输出参数{out}           |              |
| -                   | -            |
| 返回值                 |              |
| FlagStatus          | SET或RESET    |

例如：

```
/* get DMA0 channel 0 flag*/
FlagStatus flag = RESET;
flag = dma_flag_get(DMA0, DMA_CH0, DMA_FLAG_FTF);
```

### 函数 dma\_flag\_clear

函数 dma\_flag\_clear 描述见下表：

表 3-244. 函数 dma\_flag\_clear

|                        |   |
|------------------------|---|
| 函数名称                   | dma_flag_clear  |
| 函数原型                   | void dma_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag); |
| 功能描述                   | 清除DMA通道x标志位状态   |
| 先决条件                   | 无   |
| 被调用函数                  | 无   |
| 输入参数{in}               |   |
| dma_periph             | DMA外设   |
| <i>DMAx(x=0,1)</i>     | DMA外设选择   |
| 输入参数{in}               |   |
| channelx               | DMA通道   |
| <i>DMA_CHx(x=0..6)</i> | DMA通道选择，参考 <a href="#">表3-216. 枚举dma_channel_enum</a>                               |
| 输入参数{in}               |   |
| flag                   | DMA标志   |
| <i>DMA_FLAG_G</i>      | DMA通道全局中断标志   |
| <i>DMA_FLAG_FTF</i>    | DMA通道传输完成标志   |
| <i>DMA_FLAG_HTF</i>    | DMA通道半传输完成标志  |
| <i>DMA_FLAG_ERR</i>    | DMA通道错误标志   |
| 输出参数{out}              |   |
| -                      | -   |
| 返回值                    |   |
| -                      | -   |

例如：

```
/* clear DMA0 channel 0 flag*/
dma_flag_clear(DMA0, DMA_CH0, DMA_FLAG_FTF);
```

## 函数 dma\_interrupt\_enable

函数 dma\_interrupt\_enable 描述见下表：

表 3-245. 函数 dma\_interrupt\_enable

|                 |   |
|-----------------|---|
| 函数名称            | dma_interrupt_enable  |
| 函数原型            | void dma_interrupt_enable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source); |
| 功能描述            | DMA通道x中断使能  |
| 先决条件            | 无   |
| 被调用函数           | 无   |
| 输入参数{in}        |   |
| dma_periph      | DMA外设   |
| DMAx(x=0,1)     | DMA外设选择   |
| 输入参数{in}        |   |
| channelx        | DMA通道   |
| DMA_CHx(x=0..6) | DMA通道选择，参考 <a href="#">表3-216. 枚举dma_channel_enum</a>                                       |
| 输入参数{in}        |   |
| source          | DMA中断源  |
| DMA_INT_FTF     | DMA通道传输完成中断   |
| DMA_INT_HTF     | DMA通道半传输完成中断  |
| DMA_INT_ERR     | DMA通道错误中断   |
| 输出参数{out}       |   |
| -               | -   |
| 返回值             |   |
| -               | -   |

例如：

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

## 函数 dma\_interrupt\_disable

函数 dma\_interrupt\_disable 描述见下表：

表 3-246. 函数 dma\_interrupt\_disable

|            |  |
|------------|--|
| 函数名称       | dma_interrupt_disable  |
| 函数原型       | void dma_interrupt_disable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source); |
| 功能描述       | DMA通道x中断禁能   |
| 先决条件       | 无  |
| 被调用函数      | 无  |
| 输入参数{in}   |  |
| dma_periph | DMA外设  |

|                        |  |
|------------------------|--|
| <i>DMAx(x=0,1)</i>     | DMA外设选择  |
| 输入参数{in}               |  |
| <b>channelx</b>        | DMA通道  |
| <i>DMA_CHx(x=0..6)</i> | DMA通道选择, 参考 <a href="#">表3-216. 枚举dma_channel_enum</a> |
| 输入参数{in}               |  |
| <b>source</b>          | DMA中断源   |
| <i>DMA_INT_FTF</i>     | DMA通道传输完成中断  |
| <i>DMA_INT_HTF</i>     | DMA通道半传输完成中断   |
| <i>DMA_INT_ERR</i>     | DMA通道错误中断  |
| 输出参数{out}              |  |
| -                      | -  |
| 返回值                    |  |
| -                      | -  |

例如:

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

### 函数 dma\_interrupt\_flag\_get

函数 dma\_interrupt\_flag\_get 描述见下表:

表 3-247. 函数 dma\_interrupt\_flag\_get

|                         |   |
|-------------------------|---|
| 函数名称                    | dma_interrupt_flag_get  |
| 函数原型                    | FlagStatus dma_interrupt_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag); |
| 功能描述                    | 获取DMA通道x中断标志位状态   |
| 先决条件                    | 无   |
| 被调用函数                   | 无   |
| 输入参数{in}                |   |
| <b>dma_periph</b>       | DMA外设   |
| <i>DMAx(x=0,1)</i>      | DMA外设选择   |
| 输入参数{in}                |   |
| <b>channelx</b>         | DMA通道   |
| <i>DMA_CHx(x=0..6)</i>  | DMA通道选择, 参考 <a href="#">表3-216. 枚举dma_channel_enum</a>  |
| 输入参数{in}                |   |
| <b>flag</b>             | DMA标志   |
| <i>DMA_INT_FLAG_FTF</i> | DMA通道传输完成中断标志   |
| <i>DMA_INT_FLAG_HTF</i> | DMA通道半传输完成中断标志  |
| <i>DMA_INT_FLAG_ERR</i> | DMA通道错误中断标志   |



| 输出参数{out}  |           |
|------------|-----------|
| -          | -         |
| 返回值        |           |
| FlagStatus | SET或RESET |

例如：

```

/* get DMA0 channel 3 interrupt flag*/
if(dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_G);
}

```

### 函数 dma\_interrupt\_flag\_clear

函数 dma\_interrupt\_flag\_clear 描述见下表：

表 3-248. 函数 dma\_interrupt\_flag\_clear

| 函数名称                 | dma_interrupt_flag_clear  |
|----------------------|---|
| 函数原型                 | void dma_interrupt_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag); |
| 功能描述                 | 清除DMA通道x中断标志位状态   |
| 先决条件                 | 无   |
| 被调用函数                | 无   |
| 输入参数{in}             |   |
| dma_periph           | DMA外设   |
| DMAx(x=0,1)          | DMA外设选择   |
| 输入参数{in}             |   |
| channelx             | DMA通道   |
| DMA_CHx(x=0..6)      | DMA通道选择，参考 <a href="#">表3-216. 枚举dma_channel_enum</a>   |
| 输入参数{in}             |   |
| flag                 | DMA标志   |
| DMA_INT_FLAG_G       | DMA通道全局中断标志   |
| DMA_INT_FLAG_FTF     | DMA通道传输完成中断标志   |
| DMA_INT_FLAG_HT<br>F | DMA通道半传输完成中断标志  |
| DMA_INT_FLAG_ER<br>R | DMA通道错误中断标志   |
| 输出参数{out}            |   |
| -                    | -   |
| 返回值                  |   |
| -                    | -   |

例如：

```

/* clear DMA channel 3 interrupt flag*/
if(dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_FTF)){

```

```

dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_G);
}

```

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

函数 dmamux\_sync\_struct\_para\_init 描述见下表：

**表 3-249. 函数 dmamux\_sync\_struct\_para\_init**

|             |  |
|-------------|--|
| 函数名称        | dmamux_sync_struct_para_init   |
| 函数原型        | void dmamux_sync_struct_para_init(dmamux_sync_parameter_struct *init_struct);                          |
| 功能描述        | 将DMAMUX同步结构体中所有参数初始化为默认值   |
| 先决条件        | 无  |
| 被调用函数       | 无  |
| 输入参数{in}    |  |
| -           | -  |
| 输出参数{out}   |  |
| init_struct | 一个已经定义的dmamux_sync_parameter_struct结构体变量地址，参考 <a href="#">表 3-214. 结构体dmamux_sync_parameter_struct</a> |
| 返回值         |  |
| -           | -  |

例如：

```

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

```

### 函数 dmamux\_synchronization\_init

函数 dmamux\_synchronization\_init 描述见下表：

**表 3-250. 函数 dmamux\_synchronization\_init**

|                        |  |
|------------------------|--|
| 函数名称                   | dmamux_synchronization_init  |
| 函数原型                   | void dmamux_synchronization_init(dmamux_multiplexer_channel_enum channelx, dmamux_sync_parameter_struct *init_struct); |
| 功能描述                   | 初始化DMAMUX同步结构体通道x  |
| 先决条件                   | 无  |
| 被调用函数                  | 无  |
| 输入参数{in}               |  |
| channelx               | 指定要初始化的DMAMUX请求路由通道  |
| DMAMUX_MUXCHx(x=0..11) | DMAMUX通道选择，参考 <a href="#">表3-217. 枚举 dmamux_multiplexer_channel_enum</a>   |
| 输入参数{in}               |  |
| init_struct            | 一个已经定义的dmamux_sync_parameter_struct结构体变量地址，参考 <a href="#">表</a>  |

|           |  |
|-----------|--|
|           | <a href="#">3-214. 结构体dmamux_sync_parameter_struct</a> |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如：

```

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

```

### 函数 dmamux\_synchronization\_enable

函数 dmamux\_synchronization\_enable 描述见下表：

**表 3-251. 函数 dmamux\_synchronization\_enable**

|                        |   |
|------------------------|---|
| 函数名称                   | dmamux_synchronization_enable   |
| 函数原型                   | void dmamux_synchronization_enable(dmamux_multiplexer_channel_enum channelx); |
| 功能描述                   | 使能DMAMUX同步模式  |
| 先决条件                   | 无   |
| 被调用函数                  | 无   |
| 输入参数{in}               |   |
| channelx               | 指定要初始化的DMAMUX请求路由通道   |
| DMAMUX_MUXCHx(x=0..11) | DMAMUX通道选择，参考 <a href="#">表3-217. 枚举 dmamux_multiplexer_channel_enum</a>      |
| 输出参数{out}              |   |
| -                      | -   |
| 返回值                    |   |
| -                      | -   |

例如：

```

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

```

### 函数 dmamux\_synchronization\_disable

函数 dmamux\_synchronization\_disable 描述见下表：

表 3-252. 函数 `dmamux_synchronization_disable`

|                                    |   |
|------------------------------------|---|
| 函数名称                               | <code>dmamux_synchronization_disable</code>   |
| 函数原型                               | <code>void dmamux_synchronization_disable(dmamux_multiplexer_channel_enum channelx);</code> |
| 功能描述                               | 禁能DMAMUX同步模式  |
| 先决条件                               | 无   |
| 被调用函数                              | 无   |
| 输入参数{in}                           |   |
| <b>channelx</b>                    | 指定要初始化的DMAMUX请求路由通道   |
| <i>DMAMUX_MUXCHx</i> ( $x=0..11$ ) | DMAMUX通道选择, 参考 <a href="#">表3-217. 枚举 <code>dmamux_multiplexer_channel_enum</code></a>      |
| 输出参数{out}                          |   |
| -                                  | -   |
| 返回值                                |   |
| -                                  | -   |

例如:

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

### 函数 `dmamux_event_generation_enable`

函数 `dmamux_event_generation_enable` 描述见下表:

表 3-253. 函数 `dmamux_event_generation_enable`

|                                    |   |
|------------------------------------|---|
| 函数名称                               | <code>dmamux_event_generation_enable</code>   |
| 函数原型                               | <code>void dmamux_event_generation_enable(dmamux_multiplexer_channel_enum channelx);</code> |
| 功能描述                               | 使能DMAMUX事件输出  |
| 先决条件                               | 无   |
| 被调用函数                              | 无   |
| 输入参数{in}                           |   |
| <b>channelx</b>                    | 指定要初始化的DMAMUX请求路由通道   |
| <i>DMAMUX_MUXCHx</i> ( $x=0..11$ ) | DMAMUX通道选择, 参考 <a href="#">表3-217. 枚举 <code>dmamux_multiplexer_channel_enum</code></a>      |
| 输出参数{out}                          |   |
| -                                  | -   |
| 返回值                                |   |
| -                                  | -   |

例如:

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

**函数 dmamux\_event\_generation\_disable**

函数 dmamux\_event\_generation\_disable 描述见下表：

**表 3-254. 函数 dmamux\_event\_generation\_disable**

|                        |   |
|------------------------|---|
| 函数名称                   | dmamux_event_generation_disable   |
| 函数原型                   | void dmamux_event_generation_disable(dmamux_multiplexer_channel_enum channelx); |
| 功能描述                   | 禁能DMAMUX事件输出  |
| 先决条件                   | 无   |
| 被调用函数                  | 无   |
| 输入参数{in}               |   |
| channelx               | 指定要初始化的DMAMUX请求路由通道   |
| DMAMUX_MUXCHx(x=0..11) | DMAMUX通道选择，参考 <a href="#">表3-217. 枚举 dmamux_multiplexer_channel_enum</a>        |
| 输出参数{out}              |   |
| -                      | -   |
| 返回值                    |   |
| -                      | -   |

例如：

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

**函数 dmamux\_gen\_struct\_para\_init**

函数 dmamux\_gen\_struct\_para\_init 描述见下表：

**表 3-255. 函数 dmamux\_gen\_struct\_para\_init**

|             |   |
|-------------|---|
| 函数名称        | dmamux_gen_struct_para_init   |
| 函数原型        | void dmamux_gen_struct_para_init(dmamux_gen_parameter_struct *init_struct);                         |
| 功能描述        | 将DMAMUX请求生成结构体中所有参数初始化为默认值  |
| 先决条件        | 无   |
| 被调用函数       | 无   |
| 输入参数{in}    |   |
| -           | -   |
| 输出参数{out}   |   |
| init_struct | 一个已经定义的dmamux_gen_parameter_struct结构体变量地址，参考 <a href="#">表3-215. 结构体dmamux_gen_parameter_struct</a> |
| 返回值         |   |
| -           | -   |

例如：

```

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

```

### 函数 dmamux\_request\_generator\_init

函数 dmamux\_request\_generator\_init 描述见下表：

**表 3-256. 函数 dmamux\_request\_generator\_init**

|                           |   |
|---------------------------|---|
| 函数名称                      | dmamux_request_generator_init   |
| 函数原型                      | void dmamux_request_generator_init(dmamux_generator_channel_enum channelx, dmamux_gen_parameter_struct *init_struct); |
| 功能描述                      | 初始化DMAMUX请求生成结构体通道x   |
| 先决条件                      | 无   |
| 被调用函数                     | 无   |
| 输入参数{in}                  |   |
| channelx                  | 指定要初始化的DMAMUX请求生成通道   |
| DMAMUX_GENCHx<br>(x=0..3) | DMAMUX请求生成通道选择，参考 <a href="#">表3-218. 枚举 dmamux_generator_channel_enum</a>  |
| 输入参数{in}                  |   |
| init_struct               | 一个已经定义的dmamux_gen_parameter_struct结构体变量地址，参考 <a href="#">表3-215. 结构体dmamux_gen_parameter_struct</a>                   |
| 输出参数{out}                 |   |
| -                         | -   |
| 返回值                       |   |
| -                         | -   |

例如：

```

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

```

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

函数 dmamux\_request\_generator\_channel\_enable 描述见下表：

**表 3-257. 函数 dmamux\_request\_generator\_channel\_enable**

|      |   |
|------|---|
| 函数名称 | dmamux_request_generator_channel_enable   |
| 函数原型 | void dmamux_request_generator_channel_enable(dmamux_generator_channel_enum channelx); |

|                           |   |
|---------------------------|---|
| 功能描述                      | 使能DMAMUX请求生成通道x   |
| 先决条件                      | 无   |
| 被调用函数                     | 无   |
| 输入参数{in}                  |   |
| channelx                  | 指定要初始化的DMAMUX请求生成通道   |
| DMAMUX_GENCHx<br>(x=0..3) | DMAMUX请求生成通道选择, 参考 <a href="#">表3-218. 枚举 dmamux_generator_channel_enum</a> |
| 输出参数{out}                 |   |
| -                         | -   |
| 返回值                       |   |
| -                         | -   |

例如:

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

### 函数 dmamux\_request\_generator\_channel\_disable

函数 dmamux\_request\_generator\_channel\_disable 描述见下表:

表 3-258. 函数 dmamux\_request\_generator\_channel\_disable

|                           |   |
|---------------------------|---|
| 函数名称                      | dmamux_request_generator_channel_disable  |
| 函数原型                      | void<br>dmamux_request_generator_channel_disable(dmamux_generator_channel_enum channelx); |
| 功能描述                      | 禁能DMAMUX请求生成通道x   |
| 先决条件                      | 无   |
| 被调用函数                     | 无   |
| 输入参数{in}                  |   |
| channelx                  | 指定要初始化的DMAMUX请求生成通道   |
| DMAMUX_GENCHx<br>(x=0..3) | DMAMUX请求生成通道选择, 参考 <a href="#">表3-218. 枚举 dmamux_generator_channel_enum</a>               |
| 输出参数{out}                 |   |
| -                         | -   |
| 返回值                       |   |
| -                         | -   |

例如:

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

### 函数 dmamux\_synchronization\_polarity\_config

函数 dmamux\_synchronization\_polarity\_config 描述见下表:

表 3-259. 函数 `dmamux_synchronization_polarity_config`

|   |   |
|---|---|
| 函数名称  | <code>dmamux_synchronization_polarity_config</code>   |
| 函数原型  | void<br><code>dmamux_synchronization_polarity_config(dmamux_multiplexer_channel_enum channelx, uint32_t polarity);</code> |
| 功能描述  | 配置DMAMUX同步输入的有效边沿   |
| 先决条件  | 无   |
| 被调用函数                                       | 无   |
| 输入参数{in}                                    |   |
| <b>channelx</b>                             | 指定要初始化的DMAMUX请求路由通道   |
| <code>DMAMUX_MUXCHx</code><br>( $x=0..11$ ) | DMAMUX通道选择, 参考 <a href="#">表3-217. 枚举 <code>dmamux_multiplexer_channel_enum</code></a>                                    |
| 输入参数{in}                                    |   |
| <b>polarity</b>                             | 同步输入有效边沿  |
| <code>DMAMUX_SYNC_NO_EVENT</code>           | 不检测边沿   |
| <code>DMAMUX_SYNC_RISING</code>             | 上升沿   |
| <code>DMAMUX_SYNC_FALLING</code>            | 下降沿   |
| <code>DMAMUX_SYNC_RISING_FALLING</code>     | 上升和下降沿  |
| 输出参数{out}                                   |   |
| -   | -   |
| 返回值   |   |
| -   | -   |

例如:

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

### 函数 `dmamux_request_forward_number_config`

函数 `dmamux_request_forward_number_config` 描述见下表:

表 3-260. 函数 `dmamux_request_forward_number_config`

|       |   |
|-------|---|
| 函数名称  | <code>dmamux_request_forward_number_config</code>   |
| 函数原型  | void<br><code>dmamux_request_forward_number_config(dmamux_multiplexer_channel_enum channelx, uint32_t number);</code> |
| 功能描述  | 配置DMAMUX通道x要传输多少个DMA请求  |
| 先决条件  | 无   |
| 被调用函数 | 无   |



| 输入参数{in}                                 |   |
|--|---|
| <b>channelx</b>                          | 指定要初始化的DMAMUX请求路由通道   |
| <i>DMAMUX_MUXCH</i><br><i>x(x=0..11)</i> | DMAMUX通道选择, 参考 <a href="#">表3-217. 枚举 <i>dmamux_muxchannel_enum</i></a> |
| 输入参数{in}                                 |   |
| <b>number</b>                            | 要传输的DMA请求数量 (1 - 32)  |
| 输出参数{out}                                |   |
| -  | -   |
| 返回值                                      |   |
| -  | -   |

例如:

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

### 函数 dmamux\_sync\_id\_config

函数 dmamux\_sync\_id\_config 描述见下表:

**表 3-261. 函数 dmamux\_sync\_id\_config**

| 函数名称                                     | dmamux_sync_id_config   |
|--|---|
| 函数原型                                     | void dmamux_sync_id_config(dmamux_muxchannel_enum channelx, uint32_t id); |
| 功能描述                                     | 配置DMAMUX同步输入标识  |
| 先决条件                                     | 无   |
| 被调用函数                                    | 无   |
| 输入参数{in}                                 |   |
| <b>channelx</b>                          | 指定要初始化的DMAMUX请求路由通道   |
| <i>DMAMUX_MUXCH</i><br><i>x(x=0..11)</i> | DMAMUX通道选择, 参考 <a href="#">表3-217. 枚举 <i>dmamux_muxchannel_enum</i></a>   |
| 输入参数{in}                                 |   |
| <b>id</b>                                | 同步输入标识  |
| <i>DMAMUX_SYNC_EXTI0</i>                 | 同步输入信号为EXTI0  |
| <i>DMAMUX_SYNC_EXTI1</i>                 | 同步输入信号为EXTI1  |
| <i>DMAMUX_SYNC_EXTI2</i>                 | 同步输入信号为EXTI2  |
| <i>DMAMUX_SYNC_EXTI3</i>                 | 同步输入信号为EXTI3  |
| <i>DMAMUX_SYNC_EXTI4</i>                 | 同步输入信号为EXTI4  |
| <i>DMAMUX_SYNC_EXTI5</i>                 | 同步输入信号为EXTI5  |

|                                |                      |
|--------------------------------|----------------------|
| XTI5                           |                      |
| DMAMUX_SYNC_E<br>XTI6          | 同步输入信号为EXTI6         |
| DMAMUX_SYNC_E<br>XTI7          | 同步输入信号为EXTI7         |
| DMAMUX_SYNC_E<br>XTI8          | 同步输入信号为EXTI8         |
| DMAMUX_SYNC_E<br>XTI9          | 同步输入信号为EXTI9         |
| DMAMUX_SYNC_E<br>XTI10         | 同步输入信号为EXTI10        |
| DMAMUX_SYNC_E<br>XTI11         | 同步输入信号为EXTI11        |
| DMAMUX_SYNC_E<br>XTI12         | 同步输入信号为EXTI12        |
| DMAMUX_SYNC_E<br>XTI13         | 同步输入信号为EXTI13        |
| DMAMUX_SYNC_E<br>XTI14         | 同步输入信号为EXTI14        |
| DMAMUX_SYNC_E<br>XTI15         | 同步输入信号为EXTI15        |
| DMAMUX_SYNC_E<br>VT0_OUT0      | 同步输入信号为Evt_out0      |
| DMAMUX_SYNC_E<br>VT1_OUT1      | 同步输入信号为Evt_out1      |
| DMAMUX_SYNC_E<br>VT2_OUT2      | 同步输入信号为Evt_out2      |
| DMAMUX_SYNC_E<br>VT3_OUT3      | 同步输入信号为Evt_out3      |
| DMAMUX_SYNC_T<br>TIMER20_CH0_O | 同步输入信号为TIMER20_CH0_O |
| 输出参数{out}                      |                      |
| -                              | -                    |
| 返回值                            |                      |
| -                              | -                    |

例如：

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

### 函数 dmamux\_request\_id\_config

函数 dmamux\_request\_id\_config 描述见下表：

表 3-262. 函数 dmamux\_request\_id\_config

|                        |   |
|------------------------|---|
| 函数名称                   | dmamux_request_id_config  |
| 函数原型                   | void dmamux_request_id_config(dmamux_multiplexer_channel_enum channelx, uint32_t id); |
| 功能描述                   | 配置DMAMUX请求路由通道输入标识  |
| 先决条件                   | 无   |
| 被调用函数                  | 无   |
| 输入参数{in}               |   |
| channelx               | 指定要初始化的DMAMUX请求路由通道   |
| DMAMUX_MUXCHx(x=0..11) | DMAMUX通道选择, 参考 <a href="#">表3-217. 枚举 dmamux_multiplexer_channel_enum</a>             |
| 输入参数{in}               |   |
| id                     | DMA请求输入标识   |
| DMA_REQUEST_M2M        | 内存到内存传输   |
| DMA_REQUEST_GENERATOR0 | DMAMUX请求生成通道0请求   |
| DMA_REQUEST_GENERATOR1 | DMAMUX请求生成通道1请求   |
| DMA_REQUEST_GENERATOR2 | DMAMUX请求生成通道2请求   |
| DMA_REQUEST_GENERATOR3 | DMAMUX请求生成通道3请求   |
| DMA_REQUEST_ADC        | DMAMUX ADC请求  |
| DMA_REQUEST_DAC_CH0    | DMAMUX DAC CH0请求  |
| DMA_REQUEST_I2C1_RX    | DMAMUX I2C1 RX请求  |
| DMA_REQUEST_I2C1_TX    | DMAMUX I2C1 TX请求  |
| DMA_REQUEST_I2C0_RX    | DMAMUX I2C0 RX请求  |
| DMA_REQUEST_I2C0_TX    | DMAMUX I2C0 TX请求  |
| DMA_REQUEST_SSTAT0     | DMAMUX SSTAT0请求   |
| DMA_REQUEST_SSTAT1     | DMAMUX SSTAT1请求   |
| DMA_REQUEST_SSTAT2     | DMAMUX SSTAT2请求   |
| DMA_REQUEST_SSTAT3     | DMAMUX SSTAT3请求   |

|                             |                      |
|-----------------------------|----------------------|
| DMA_REQUEST_S<br>PI0_RX     | DMAMUX SPI0 RX请求     |
| DMA_REQUEST_S<br>PI0_TX     | DMAMUX SPI0 TX请求     |
| DMA_REQUEST_S<br>PI1_RX     | DMAMUX SPI1 RX请求     |
| DMA_REQUEST_S<br>PI1_TX     | DMAMUX SPI1 TX请求     |
| DMA_REQUEST_TI<br>MER0_CH0  | DMAMUX TIMER0 CH0请求  |
| DMA_REQUEST_TI<br>MER0_CH1  | DMAMUX TIMER0 CH1请求  |
| DMA_REQUEST_TI<br>MER0_CH2  | DMAMUX TIMER0 CH2请求  |
| DMA_REQUEST_TI<br>MER0_CH3  | DMAMUX TIMER0 CH3请求  |
| DMA_REQUEST_TI<br>MER0_TI   | DMAMUX TIMER0 TI请求   |
| DMA_REQUEST_TI<br>MER0_UP   | DMAMUX TIMER0 UP请求   |
| DMA_REQUEST_TI<br>MER0_CO   | DMAMUX TIMER0 CO请求   |
| DMA_REQUEST_TI<br>MER0_MCH0 | DMAMUX TIMER0 MCH0请求 |
| DMA_REQUEST_TI<br>MER0_MCH1 | DMAMUX TIMER0 MCH1请求 |
| DMA_REQUEST_TI<br>MER0_MCH2 | DMAMUX TIMER0 MCH2请求 |
| DMA_REQUEST_TI<br>MER0_MCH3 | DMAMUX TIMER0 MCH3请求 |
| DMA_REQUEST_TI<br>MER1_CH   | DMAMUX TIMER1 CH0请求  |
| DMA_REQUEST_TI<br>MER1_CH1  | DMAMUX TIMER1 CH1请求  |
| DMA_REQUEST_TI<br>MER1_CH2  | DMAMUX TIMER1 CH2请求  |
| DMA_REQUEST_TI<br>MER1_CH3  | DMAMUX TIMER1 CH3请求  |
| DMA_REQUEST_TI<br>MER1_TI   | DMAMUX TIMER1 TI请求   |
| DMA_REQUEST_TI<br>MER1_UP   | DMAMUX TIMER1 UP请求   |
| DMA_REQUEST_TI              | DMAMUX TIMER7 CH0请求  |

|                                |                      |
|--------------------------------|----------------------|
| <i>MER7_CH0</i>                |                      |
| <i>DMA_REQUEST_TIMER7_CH1</i>  | DMAMUX TIMER7 CH1请求  |
| <i>DMA_REQUEST_TIMER7_CH2</i>  | DMAMUX TIMER7 CH2请求  |
| <i>DMA_REQUEST_TIMER7_CH3</i>  | DMAMUX TIMER7 CH3请求  |
| <i>DMA_REQUEST_TIMER7_TI</i>   | DMAMUX TIMER7 TI请求   |
| <i>DMA_REQUEST_TIMER7_UP</i>   | DMAMUX TIMER7 UP请求   |
| <i>DMA_REQUEST_TIMER7_CO</i>   | DMAMUX TIMER7 CO请求   |
| <i>DMA_REQUEST_TIMER7_MCH0</i> | DMAMUX TIMER7 MCH0请求 |
| <i>DMA_REQUEST_TIMER7_MCH1</i> | DMAMUX TIMER7 MCH1请求 |
| <i>DMA_REQUEST_TIMER7_MCH2</i> | DMAMUX TIMER7 MCH2请求 |
| <i>DMA_REQUEST_TIMER7_MCH3</i> | DMAMUX TIMER7 MCH3请求 |
| <i>DMA_REQUEST_CAN1</i>        | DMAMUX CAN1请求        |
| <i>DMA_REQUEST_CAN0</i>        | DMAMUX CAN0请求        |
| <i>DMA_REQUEST_USART0_RX</i>   | DMAMUX USART0 RX请求   |
| <i>DMA_REQUEST_USART0_TX</i>   | DMAMUX USART0 TX请求   |
| <i>DMA_REQUEST_USART1_RX</i>   | DMAMUX USART1 RX请求   |
| <i>DMA_REQUEST_USART1_TX</i>   | DMAMUX USART1 TX请求   |
| <i>DMA_REQUEST_USART2_RX</i>   | DMAMUX USART2 RX请求   |
| <i>DMA_REQUEST_USART2_TX</i>   | DMAMUX USART2 TX请求   |
| <i>DMA_REQUEST_TIMER5_UP</i>   | DMAMUX TIMER5 UP请求   |
| <i>DMA_REQUEST_TIMER6_UP</i>   | DMAMUX TIMER6 UP请求   |
| <i>DMA_REQUEST_TIMER19_CH0</i> | DMAMUX TIMER19 CH0请求 |

|                          |                        |
|--------------------------|------------------------|
| DMA_REQUEST_TIMER19_CH1  | DMAMUX TIMER19 CH1 请求  |
| DMA_REQUEST_TIMER19_CH2  | DMAMUX TIMER19 CH2 请求  |
| DMA_REQUEST_TIMER19_CH3  | DMAMUX TIMER19 CH3 请求  |
| DMA_REQUEST_TIMER19_TI   | DMAMUX TIMER19 TI 请求   |
| DMA_REQUEST_TIMER19_UP   | DMAMUX TIMER19 UP 请求   |
| DMA_REQUEST_TIMER19_CO   | DMAMUX TIMER19 CO 请求   |
| DMA_REQUEST_TIMER19_MCH0 | DMAMUX TIMER19 MCH0 请求 |
| DMA_REQUEST_TIMER19_MCH1 | DMAMUX TIMER19 MCH1 请求 |
| DMA_REQUEST_TIMER19_MCH2 | DMAMUX TIMER19 MCH2 请求 |
| DMA_REQUEST_TIMER19_MCH3 | DMAMUX TIMER19 MCH3 请求 |
| DMA_REQUEST_TIMER20_CH0  | DMAMUX TIMER20 CH0 请求  |
| DMA_REQUEST_TIMER20_CH1  | DMAMUX TIMER20 CH1 请求  |
| DMA_REQUEST_TIMER20_CH2  | DMAMUX TIMER20 CH2 请求  |
| DMA_REQUEST_TIMER20_CH3  | DMAMUX TIMER20 CH3 请求  |
| DMA_REQUEST_TIMER20_TI   | DMAMUX TIMER20 TI 请求   |
| DMA_REQUEST_TIMER20_UP   | DMAMUX TIMER20 UP 请求   |
| DMA_REQUEST_TIMER20_CO   | DMAMUX TIMER20 CO 请求   |
| DMA_REQUEST_TIMER20_MCH0 | DMAMUX TIMER20 MCH0 请求 |
| DMA_REQUEST_TIMER20_MCH1 | DMAMUX TIMER20 MCH1 请求 |
| DMA_REQUEST_TIMER20_MCH2 | DMAMUX TIMER20 MCH2 请求 |
| DMA_REQUEST_TIMER20_MCH3 | DMAMUX TIMER20 MCH3 请求 |
| 输出参数{out}                |                        |

|     |   |
|-----|---|
| -   | - |
| 返回值 |   |
| -   | - |

例如：

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

### 函数 dmamux\_trigger\_polarity\_config

函数 dmamux\_trigger\_polarity\_config 描述见下表：

表 3-263. 函数 dma\_interrupt\_disable

|                           |   |
|---------------------------|---|
| 函数名称                      | dmamux_trigger_polarity_config  |
| 函数原型                      | void dmamux_trigger_polarity_config(dmamux_generator_channel_enum channelx, uint32_t polarity); |
| 功能描述                      | 配置DMAMUX触发输入的有效边沿   |
| 先决条件                      | 无   |
| 被调用函数                     | 无   |
| 输入参数{in}                  |   |
| channelx                  | 指定要初始化的DMAMUX请求生成通道   |
| DMAMUX_GENCHx<br>(x=0..3) | DMAMUX请求生成通道选择，参考 <a href="#">表3-218. 枚举 dmamux_generator_channel_enum</a>                      |
| 输入参数{in}                  |   |
| polarity                  | 触发输入信号有效边沿  |
| DMAMUX_GEN_NO_EVENT       | 不检测边沿   |
| DMAMUX_GEN_RISING         | 上升沿   |
| DMAMUX_GEN_FALLING        | 下降沿   |
| DMAMUX_GEN_RISING_FALLING | 上升和下降沿  |
| 输出参数{out}                 |   |
| -                         | -   |
| 返回值                       |   |
| -                         | -   |

例如：

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

函数 **dmamux\_request\_generate\_number\_config**

函数 **dmamux\_request\_generate\_number\_config** 描述见下表：

表 3-264. 函数 **dmamux\_request\_generate\_number\_config**

|                           |   |
|---------------------------|---|
| 函数名称                      | dmamux_request_generate_number_config   |
| 函数原型                      | void<br>dmamux_request_generate_number_config(dmamux_generator_channel_enum channelx, uint32_t number); |
| 功能描述                      | 配置DMAMUX请求生成器生成请求的数量  |
| 先决条件                      | 无   |
| 被调用函数                     | 无   |
| 输入参数{in}                  |   |
| channelx                  | 指定要初始化的DMAMUX请求生成通道   |
| DMAMUX_GENCHx<br>(x=0..3) | DMAMUX请求生成通道选择，参考 <a href="#">表3-218. 枚举 dmamux_generator_channel_enum</a>                              |
| 输入参数{in}                  |   |
| number                    | 要生成的DMA请求数量（1 - 32）   |
| 输出参数{out}                 |   |
| -                         | -   |
| 返回值                       |   |
| -                         | -   |

例如：

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

函数 **dmamux\_trigger\_id\_config**

函数 **dmamux\_trigger\_id\_config** 描述见下表：

表 3-265. 函数 **dmamux\_trigger\_id\_config**

|                           |   |
|---------------------------|---|
| 函数名称                      | dmamux_trigger_id_config  |
| 函数原型                      | void dmamux_trigger_id_config(dmamux_generator_channel_enum channelx, uint32_t id); |
| 功能描述                      | 配置DMAMUX触发输入标识  |
| 先决条件                      | 无   |
| 被调用函数                     | 无   |
| 输入参数{in}                  |   |
| channelx                  | 指定要初始化的DMAMUX请求生成通道   |
| DMAMUX_GENCHx<br>(x=0..3) | DMAMUX请求生成通道选择，参考 <a href="#">表3-218. 枚举 dmamux_generator_channel_enum</a>          |
| 输入参数{in}                  |   |
| id                        | 触发输入标识  |



|                                      |                    |
|--------------------------------------|--------------------|
| DMAMUX_TRIGGE<br>R_EXTI0             | 触发输入为EXTI0         |
| DMAMUX_TRIGGE<br>R_EXTI1             | 触发输入为EXTI1         |
| DMAMUX_TRIGGE<br>R_EXTI2             | 触发输入为EXTI2         |
| DMAMUX_TRIGGE<br>R_EXTI3             | 触发输入为EXTI3         |
| DMAMUX_TRIGGE<br>R_EXTI4             | 触发输入为EXTI4         |
| DMAMUX_TRIGGE<br>R_EXTI5             | 触发输入为EXTI5         |
| DMAMUX_TRIGGE<br>R_EXTI6             | 触发输入为EXTI6         |
| DMAMUX_TRIGGE<br>R_EXTI7             | 触发输入为EXTI7         |
| DMAMUX_TRIGGE<br>R_EXTI8             | 触发输入为EXTI8         |
| DMAMUX_TRIGGE<br>R_EXTI9             | 触发输入为EXTI9         |
| DMAMUX_TRIGGE<br>R_EXTI10            | 触发输入为EXTI10        |
| DMAMUX_TRIGGE<br>R_EXTI11            | 触发输入为EXTI11        |
| DMAMUX_TRIGGE<br>R_EXTI12            | 触发输入为EXTI12        |
| DMAMUX_TRIGGE<br>R_EXTI13            | 触发输入为EXTI13        |
| DMAMUX_TRIGGE<br>R_EXTI14            | 触发输入为EXTI14        |
| DMAMUX_TRIGGE<br>R_EXTI15            | 触发输入为EXTI15        |
| DMAMUX_TRIGGE<br>R_EVT_OUT0          | 触发输入为Evt_out0      |
| DMAMUX_TRIGGE<br>R_EVT_OUT1          | 触发输入为Evt_out1      |
| DMAMUX_TRIGGE<br>R_EVT_OUT2          | 触发输入为Evt_out2      |
| DMAMUX_TRIGGE<br>R_EVT_OUT3          | 触发输入为Evt_out3      |
| DMAMUX_TRIGGE<br>R_TIMER20_CH0_<br>O | 触发输入为TIMER20_CH0_O |

| 输出参数{out} |   |
|-----------|---|
| -         | - |
| 返回值       |   |
| -         | - |

例如:

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

### 函数 dmamux\_flag\_get

函数 dmamux\_flag\_get 描述见下表:

表 3-266. 函数 dmamux\_flag\_get

| 函数名称                   | dmamux_flag_get                                     |
|------------------------|---|
| 函数原型                   | FlagStatus dmamux_flag_get(dmamux_flag_enum flag);  |
| 功能描述                   | 获取DMAMUX通道x标志位状态                                    |
| 先决条件                   | 无   |
| 被调用函数                  | 无   |
| 输入参数{in}               |   |
| flag                   | 标志类型, 参考 <a href="#">表3-220. 枚举dmamux_flag_enum</a> |
| DMAMUX_FLAG_MUXCH0_SO  | DMAMUX请求路由通道0同步溢出标志                                 |
| DMAMUX_FLAG_MUXCH1_SO  | DMAMUX请求路由通道1同步溢出标志                                 |
| DMAMUX_FLAG_MUXCH2_SO  | DMAMUX请求路由通道2同步溢出标志                                 |
| DMAMUX_FLAG_MUXCH3_SO  | DMAMUX请求路由通道3同步溢出标志                                 |
| DMAMUX_FLAG_MUXCH4_SO  | DMAMUX请求路由通道4同步溢出标志                                 |
| DMAMUX_FLAG_MUXCH5_SO  | DMAMUX请求路由通道5同步溢出标志                                 |
| DMAMUX_FLAG_MUXCH6_SO  | DMAMUX请求路由通道6同步溢出标志                                 |
| DMAMUX_FLAG_MUXCH7_SO  | DMAMUX请求路由通道7同步溢出标志                                 |
| DMAMUX_FLAG_MUXCH8_SO  | DMAMUX请求路由通道8同步溢出标志                                 |
| DMAMUX_FLAG_MUXCH9_SO  | DMAMUX请求路由通道9同步溢出标志                                 |
| DMAMUX_FLAG_MUXCH10_SO | DMAMUX请求路由通道10同步溢出标志                                |
| DMAMUX_FLAG_MUXCH11_SO | DMAMUX请求路由通道11同步溢出标志                                |

|                           |                     |
|---------------------------|---------------------|
| UXCH11_SO                 |                     |
| DMAMUX_FLAG_G<br>ENCH0_TO | DMAMUX请求生成通道0触发溢出标志 |
| DMAMUX_FLAG_G<br>ENCH1_TO | DMAMUX请求生成通道1触发溢出标志 |
| DMAMUX_FLAG_G<br>ENCH2_TO | DMAMUX请求生成通道2触发溢出标志 |
| DMAMUX_FLAG_G<br>ENCH3_TO | DMAMUX请求生成通道3触发溢出标志 |
| 输出参数{out}                 |                     |
| -                         | -                   |
| 返回值                       |                     |
| FlagStatus                | SET或RESET           |

例如：

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

### 函数 dmamux\_flag\_clear

函数 dmamux\_flag\_clear 描述见下表：

表 3-267. 函数 dmamux\_flag\_clear

|                           |  |
|---------------------------|--|
| 函数名称                      | dmamux_flag_clear                                  |
| 函数原型                      | void dmamux_flag_clear(dmamux_flag_enum flag);     |
| 功能描述                      | 清除DMAMUX通道x标志位状态                                   |
| 先决条件                      | 无  |
| 被调用函数                     | 无  |
| 输入参数{in}                  |  |
| flag                      | 标志类型，参考 <a href="#">表3-220. 枚举dmamux_flag_enum</a> |
| DMAMUX_FLAG_M<br>UXCH0_SO | DMAMUX请求路由通道0同步溢出标志                                |
| DMAMUX_FLAG_M<br>UXCH1_SO | DMAMUX请求路由通道1同步溢出标志                                |
| DMAMUX_FLAG_M<br>UXCH2_SO | DMAMUX请求路由通道2同步溢出标志                                |
| DMAMUX_FLAG_M<br>UXCH3_SO | DMAMUX请求路由通道3同步溢出标志                                |
| DMAMUX_FLAG_M<br>UXCH4_SO | DMAMUX请求路由通道4同步溢出标志                                |
| DMAMUX_FLAG_M<br>UXCH5_SO | DMAMUX请求路由通道5同步溢出标志                                |
| DMAMUX_FLAG_M             | DMAMUX请求路由通道6同步溢出标志                                |

|                            |                      |
|----------------------------|----------------------|
| UXCH6_SO                   |                      |
| DMAMUX_FLAG_M<br>UXCH7_SO  | DMAMUX请求路由通道7同步溢出标志  |
| DMAMUX_FLAG_M<br>UXCH8_SO  | DMAMUX请求路由通道8同步溢出标志  |
| DMAMUX_FLAG_M<br>UXCH9_SO  | DMAMUX请求路由通道9同步溢出标志  |
| DMAMUX_FLAG_M<br>UXCH10_SO | DMAMUX请求路由通道10同步溢出标志 |
| DMAMUX_FLAG_M<br>UXCH11_SO | DMAMUX请求路由通道11同步溢出标志 |
| DMAMUX_FLAG_G<br>ENCH0_TO  | DMAMUX请求生成通道0触发溢出标志  |
| DMAMUX_FLAG_G<br>ENCH1_TO  | DMAMUX请求生成通道1触发溢出标志  |
| DMAMUX_FLAG_G<br>ENCH2_TO  | DMAMUX请求生成通道2触发溢出标志  |
| DMAMUX_FLAG_G<br>ENCH3_TO  | DMAMUX请求生成通道3触发溢出标志  |
| 输出参数{out}                  |                      |
| -                          | -                    |
| 返回值                        |                      |
| -                          | -                    |

例如：

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

## 函数 dmamux\_interrupt\_enable

函数 dmamux\_interrupt\_enable 描述见下表：

表 3-268. 函数 dmamux\_interrupt\_enable

|                          |  |
|--------------------------|--|
| 函数名称                     | dmamux_interrupt_enable  |
| 函数原型                     | void dmamux_interrupt_enable(dmamux_interrupt_enum interrupt); |
| 功能描述                     | 使能DMAMUX通道x中断  |
| 先决条件                     | 无  |
| 被调用函数                    | 无  |
| 输入参数{in}                 |  |
| interrupt                | 中断类型，参考 <a href="#">表3-219. 枚举dmamux_interrupt_enum</a>        |
| DMAMUX_INT_MU<br>XCH0_SO | DMAMUX请求路由通道0同步溢出中断  |
| DMAMUX_INT_MU<br>XCH1_SO | DMAMUX请求路由通道1同步溢出中断  |

|                           |                      |
|---------------------------|----------------------|
| DMAMUX_INT_MU<br>XCH2_SO  | DMAMUX请求路由通道2同步溢出中断  |
| DMAMUX_INT_MU<br>XCH3_SO  | DMAMUX请求路由通道3同步溢出中断  |
| DMAMUX_INT_MU<br>XCH4_SO  | DMAMUX请求路由通道4同步溢出中断  |
| DMAMUX_INT_MU<br>XCH5_SO  | DMAMUX请求路由通道5同步溢出中断  |
| DMAMUX_INT_MU<br>XCH6_SO  | DMAMUX请求路由通道6同步溢出中断  |
| DMAMUX_INT_MU<br>XCH7_SO  | DMAMUX请求路由通道7同步溢出中断  |
| DMAMUX_INT_MU<br>XCH8_SO  | DMAMUX请求路由通道8同步溢出中断  |
| DMAMUX_INT_MU<br>XCH9_SO  | DMAMUX请求路由通道9同步溢出中断  |
| DMAMUX_INT_MU<br>XCH10_SO | DMAMUX请求路由通道10同步溢出中断 |
| DMAMUX_INT_MU<br>XCH11_SO | DMAMUX请求路由通道11同步溢出中断 |
| DMAMUX_INT_GE<br>NCH0_TO  | DMAMUX请求生成通道0触发溢出中断  |
| DMAMUX_INT_GE<br>NCH1_TO  | DMAMUX请求生成通道1触发溢出中断  |
| DMAMUX_INT_GE<br>NCH2_TO  | DMAMUX请求生成通道2触发溢出中断  |
| DMAMUX_INT_GE<br>NCH3_TO  | DMAMUX请求生成通道3触发溢出中断  |
| 输出参数{out}                 |                      |
| -                         | -                    |
| 返回值                       |                      |
| -                         | -                    |

例如：

```
/* enable DMAMUX interrupt */
dmamux_interrupt_enable(DMAMUX_INT_MUXCH0_SO);
```

### 函数 dmamux\_interrupt\_disable

函数 dmamux\_interrupt\_disable 描述见下表：

表 3-269. 函数 dmamux\_interrupt\_disable

|      |   |
|------|---|
| 函数名称 | dmamux_interrupt_disable  |
| 函数原型 | void dmamux_interrupt_disable(dmamux_interrupt_enum interrupt); |

|                           |   |
|---------------------------|---|
| 功能描述                      | 禁能DMAMUX通道x中断   |
| 先决条件                      | 无   |
| 被调用函数                     | 无   |
| 输入参数{in}                  |   |
| interrupt                 | 中断类型，参考 <a href="#">表3-219. 枚举dmamux_interrupt_enum</a> |
| DMAMUX_INT_MU<br>XCH0_SO  | DMAMUX请求路由通道0同步溢出中断                                     |
| DMAMUX_INT_MU<br>XCH1_SO  | DMAMUX请求路由通道1同步溢出中断                                     |
| DMAMUX_INT_MU<br>XCH2_SO  | DMAMUX请求路由通道2同步溢出中断                                     |
| DMAMUX_INT_MU<br>XCH3_SO  | DMAMUX请求路由通道3同步溢出中断                                     |
| DMAMUX_INT_MU<br>XCH4_SO  | DMAMUX请求路由通道4同步溢出中断                                     |
| DMAMUX_INT_MU<br>XCH5_SO  | DMAMUX请求路由通道5同步溢出中断                                     |
| DMAMUX_INT_MU<br>XCH6_SO  | DMAMUX请求路由通道6同步溢出中断                                     |
| DMAMUX_INT_MU<br>XCH7_SO  | DMAMUX请求路由通道7同步溢出中断                                     |
| DMAMUX_INT_MU<br>XCH8_SO  | DMAMUX请求路由通道8同步溢出中断                                     |
| DMAMUX_INT_MU<br>XCH9_SO  | DMAMUX请求路由通道9同步溢出中断                                     |
| DMAMUX_INT_MU<br>XCH10_SO | DMAMUX请求路由通道10同步溢出中断                                    |
| DMAMUX_INT_MU<br>XCH11_SO | DMAMUX请求路由通道11同步溢出中断                                    |
| DMAMUX_INT_GE<br>NCH0_TO  | DMAMUX请求生成通道0触发溢出中断                                     |
| DMAMUX_INT_GE<br>NCH1_TO  | DMAMUX请求生成通道1触发溢出中断                                     |
| DMAMUX_INT_GE<br>NCH2_TO  | DMAMUX请求生成通道2触发溢出中断                                     |
| DMAMUX_INT_GE<br>NCH3_TO  | DMAMUX请求生成通道3触发溢出中断                                     |
| 输出参数{out}                 |   |
| -                         | -   |
| 返回值                       |   |
| -                         | -   |

例如：

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

```
dmamux_interrupt_disable(DMAMUX_INT_MUXCH0_SO);
```

## 函数 dmamux\_interrupt\_flag\_get

函数 dmamux\_interrupt\_flag\_get 描述见下表:

**表 3-270. 函数 dmamux\_interrupt\_flag\_get**

| 函数名称                       | dmamux_interrupt_flag_get  |
|----------------------------|--|
| 函数原型                       | FlagStatus dmamux_interrupt_flag_get(dmamux_interrupt_flag_enum int_flag); |
| 功能描述                       | 获取DMAMUX通道x中断标志位状态   |
| 先决条件                       | 无  |
| 被调用函数                      | 无  |
| 输入参数{in}                   |  |
| int_flag                   | 标志类型, 参考 <a href="#">表3-221. 枚举dmamux_interrupt_flag_enum</a>              |
| DMAMUX_INT_FLAG_MUXCH0_SO  | DMAMUX请求路由通道0同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH1_SO  | DMAMUX请求路由通道1同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH2_SO  | DMAMUX请求路由通道2同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH3_SO  | DMAMUX请求路由通道3同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH4_SO  | DMAMUX请求路由通道4同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH5_SO  | DMAMUX请求路由通道5同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH6_SO  | DMAMUX请求路由通道6同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH7_SO  | DMAMUX请求路由通道7同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH8_SO  | DMAMUX请求路由通道8同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH9_SO  | DMAMUX请求路由通道9同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH10_SO | DMAMUX请求路由通道10同步溢出中断标志   |
| DMAMUX_INT_FLAG_MUXCH11_SO | DMAMUX请求路由通道11同步溢出中断标志   |
| DMAMUX_INT_FLAG_GENCH0_TO  | DMAMUX请求生成通道0触发溢出中断标志  |
| DMAMUX_INT_FLAG_GENCH1_TO  | DMAMUX请求生成通道1触发溢出中断标志  |
| DMAMUX_INT_FLAG_GENCH2_TO  | DMAMUX请求生成通道2触发溢出中断标志  |

|                           |                       |
|---------------------------|-----------------------|
| G_GENCH2_TO               |                       |
| DMAMUX_INT_FLAG_GENCH3_TO | DMAMUX请求生成通道3触发溢出中断标志 |
| 输出参数{out}                 |                       |
| -                         | -                     |
| 返回值                       |                       |
| FlagStatus                | SET或RESET             |

例如：

```

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

```

### 函数 dmamux\_interrupt\_flag\_clear

函数 dmamux\_interrupt\_flag\_clear 描述见下表：

表 3-271. 函数 dmamux\_interrupt\_flag\_clear

|                           |  |
|---------------------------|--|
| 函数名称                      | dmamux_interrupt_flag_clear  |
| 函数原型                      | FlagStatus dmamux_interrupt_flag_get(dmamux_interrupt_flag_enum int_flag); |
| 功能描述                      | 清除DMAMUX通道x中断标志位状态   |
| 先决条件                      | 无  |
| 被调用函数                     | 无  |
| 输入参数{in}                  |  |
| int_flag                  | 标志类型，参考 <a href="#">表3-221. 枚举dmamux_interrupt_flag_enum</a>               |
| DMAMUX_INT_FLAG_MUXCH0_SO | DMAMUX请求路由通道0同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH1_SO | DMAMUX请求路由通道1同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH2_SO | DMAMUX请求路由通道2同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH3_SO | DMAMUX请求路由通道3同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH4_SO | DMAMUX请求路由通道4同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH5_SO | DMAMUX请求路由通道5同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH6_SO | DMAMUX请求路由通道6同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH7_SO | DMAMUX请求路由通道7同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH8_SO | DMAMUX请求路由通道8同步溢出中断标志  |



|                            |                        |
|----------------------------|------------------------|
| DMAMUX_INT_FLAG_MUXCH9_SO  | DMAMUX请求路由通道9同步溢出中断标志  |
| DMAMUX_INT_FLAG_MUXCH10_SO | DMAMUX请求路由通道10同步溢出中断标志 |
| DMAMUX_INT_FLAG_MUXCH11_SO | DMAMUX请求路由通道11同步溢出中断标志 |
| DMAMUX_INT_FLAG_GENCH0_TO  | DMAMUX请求生成通道0触发溢出中断标志  |
| DMAMUX_INT_FLAG_GENCH1_TO  | DMAMUX请求生成通道1触发溢出中断标志  |
| DMAMUX_INT_FLAG_GENCH2_TO  | DMAMUX请求生成通道2触发溢出中断标志  |
| DMAMUX_INT_FLAG_GENCH3_TO  | DMAMUX请求生成通道3触发溢出中断标志  |
| 输出参数{out}                  |                        |
| -                          | -                      |
| 返回值                        |                        |
| -                          | -                      |

例如：

```

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

```

## 3.10. EXTI

EXTI是MCU中的中断/事件控制器，包括25个相互独立的边沿检测电路并且能够向处理器内核产生中断请求或唤醒事件。章节[3.10.1](#)描述了EXTI的寄存器列表，章节[3.10.2](#)对EXTI库函数进行说明。

### 3.10.1. 外设寄存器说明

EXTI寄存器列表如下表所示：

表 3-272. EXTI 寄存器

| 寄存器名称      | 寄存器描述      |
|------------|------------|
| EXTI_INTEN | 中断使能寄存器    |
| EXTI_EVEN  | 事件使能寄存器    |
| EXTI_RTEN  | 上升沿触发使能寄存器 |
| EXTI_FTEN  | 下降沿触发使能寄存器 |
| EXTI_SWIEV | 软件中断事件寄存器  |
| EXTI_PD    | 挂起寄存器      |

### 3.10.2. 外设库函数说明

EXTI库函数列表如下表所示：

表 3-273. EXTI 库函数

| 库函数名称                           | 库函数描述          |
|---------------------------------|----------------|
| exti_deinit                     | 复位EXTI         |
| exti_init                       | 初始化EXTI线x      |
| exti_interrupt_enable           | EXTI线x中断使能     |
| exti_interrupt_disable          | EXTI线x中断禁能     |
| exti_event_enable               | EXTI线x事件使能     |
| exti_event_disable              | EXTI线x事件禁能     |
| exti_software_interrupt_enable  | EXTI线x软件中断事件使能 |
| exti_software_interrupt_disable | EXTI线x软件中断事件禁能 |
| exti_flag_get                   | 获取EXTI线x中断标志位  |
| exti_flag_clear                 | 清除EXTI线x中断标志位  |
| exti_interrupt_flag_get         | 获取EXTI线x中断标志位  |
| exti_interrupt_flag_clear       | 清除EXTI线x中断标志位  |

枚举类型 **exti\_line\_enum**

表 3-274. 枚举类型 exti\_line\_enum

| 枚举名称    | 枚举描述    |
|---------|---------|
| EXTI_0  | EXTI线0  |
| EXTI_1  | EXTI线1  |
| EXTI_2  | EXTI线2  |
| EXTI_3  | EXTI线3  |
| EXTI_4  | EXTI线4  |
| EXTI_5  | EXTI线5  |
| EXTI_6  | EXTI线6  |
| EXTI_7  | EXTI线7  |
| EXTI_8  | EXTI线8  |
| EXTI_9  | EXTI线9  |
| EXTI_10 | EXTI线10 |
| EXTI_11 | EXTI线11 |
| EXTI_12 | EXTI线12 |
| EXTI_13 | EXTI线13 |
| EXTI_14 | EXTI线14 |
| EXTI_15 | EXTI线15 |
| EXTI_16 | EXTI线16 |
| EXTI_17 | EXTI线17 |
| EXTI_18 | EXTI线18 |
| EXTI_19 | EXTI线19 |

| 枚举名称    | 枚举描述    |
|---------|---------|
| EXTI_20 | EXTI线20 |
| EXTI_21 | EXTI线21 |
| EXTI_22 | EXTI线22 |
| EXTI_23 | EXTI线23 |
| EXTI_24 | EXTI线24 |

### 枚举类型 `exti_mode_enum`

表 3-275. 枚举类型 `exti_mode_enum`

| 枚举名称           | 枚举描述     |
|----------------|----------|
| EXTI_INTERRUPT | EXTI中断模式 |
| EXTI_EVENT     | EXTI事件模式 |

### 枚举类型 `exti_trig_type_enum`

表 3-276. 枚举类型 `exti_trig_type_enum`

| 枚举名称              | 枚举描述        |
|-------------------|-------------|
| EXTI_TRIG_RISING  | EXTI上升沿触发   |
| EXTI_TRIG_FALLING | EXTI下降沿触发   |
| EXTI_TRIG_BOTH    | EXTI双边沿触发   |
| EXTI_TRIG_NONE    | EXTI双边沿均不触发 |

### 函数 `exti_deinit`

函数`exti_deinit`描述见下表：

表 3-277. 函数 `exti_deinit`

| 函数名称      | <code>exti_deinit</code>             |
|-----------|--------------------------------------|
| 函数原形      | <code>void exti_deinit(void);</code> |
| 功能描述      | 复位EXTI                               |
| 先决条件      | -                                    |
| 被调用函数     | -                                    |
| 输入参数{in}  |                                      |
| -         | -                                    |
| 输出参数{out} |                                      |
| -         | -                                    |
| 返回值       |                                      |
| -         | -                                    |

例如：

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

**函数 exti\_init**

函数exti\_init描述见下表：

**表 3-278. 函数 exti\_init**

|           |   |
|-----------|---|
| 函数名称      | exti_init   |
| 函数原型      | void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type); |
| 功能描述      | 初始化EXTI线x   |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| linex     | EXTI线x, 参考 <a href="#">表3-274. 枚举类型exti_line_enum</a>                                     |
| 输入参数{in}  |   |
| mode      | EXTI模式, 参考 <a href="#">表3-275. 枚举类型exti_mode_enum</a>                                     |
| 输入参数{in}  |   |
| trig_type | 触发类型, 参考 <a href="#">表3-276. 枚举类型exti_trig_type_enum</a>                                  |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

```
/* configure EXTI_0 */
```

```
exti_init(EXTI_0, EXTI_INTERRUPT, EXTI_TRIG_BOTH);
```

**函数 exti\_interrupt\_enable**

函数exti\_interrupt\_enable描述见下表：

**表 3-279. 函数 exti\_interrupt\_enable**

|           |   |
|-----------|---|
| 函数名称      | exti_interrupt_enable                                 |
| 函数原型      | void exti_interrupt_enable(exti_line_enum linex);     |
| 功能描述      | EXTI线x中断使能  |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| linex     | EXTI线x, 参考 <a href="#">表3-274. 枚举类型exti_line_enum</a> |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

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

```
exti_interrupt_enable(EXTI_0);
```

### 函数 exti\_interrupt\_disable

函数exti\_interrupt\_disable描述见下表：

表 3-280. 函数 exti\_interrupt\_disable

|           |   |
|-----------|---|
| 函数名称      | exti_interrupt_disable                                |
| 函数原型      | void exti_interrupt_disable(exti_line_enum linex);    |
| 功能描述      | EXTI线x中断禁能  |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| linex     | EXTI线x, 参考 <a href="#">表3-274. 枚举类型exti_line_enum</a> |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

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

```
exti_interrupt_disable(EXTI_0);
```

### 函数 exti\_event\_enable

函数exti\_event\_enable描述见下表：

表 3-281. 函数 exti\_event\_enable

|           |   |
|-----------|---|
| 函数名称      | exti_event_enable                                     |
| 函数原型      | void exti_event_enable(exti_line_enum linex);         |
| 功能描述      | EXTI线x事件使能  |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| linex     | EXTI线x, 参考 <a href="#">表3-274. 枚举类型exti_line_enum</a> |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

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

```
exti_event_enable(EXTI_0);
```

### 函数 exti\_event\_disable

函数exti\_event\_disable描述见下表：

**表 3-282. 函数 exti\_event\_disable**

|           |   |
|-----------|---|
| 函数名称      | exti_event_disable                                    |
| 函数原型      | void exti_event_disable(exti_line_enum linex);        |
| 功能描述      | EXTI线x事件禁能  |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| linex     | EXTI线x, 参考 <a href="#">表3-274. 枚举类型exti_line_enum</a> |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

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

```
exti_event_disable(EXTI_0);
```

### 函数 exti\_software\_interrupt\_enable

函数exti\_software\_interrupt\_enable描述见下表：

**表 3-283. 函数 exti\_software\_interrupt\_enable**

|           |  |
|-----------|--|
| 函数名称      | exti_software_interrupt_enable                             |
| 函数原型      | void exti_software_interrupt_enable(exti_line_enum linex); |
| 功能描述      | EXTI线x软件中断事件使能   |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| linex     | EXTI线x, 参考 <a href="#">表3-274. 枚举类型exti_line_enum</a>      |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如：

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

```
exti_software_interrupt_enable(EXTI_0);
```

函数 `exti_software_interrupt_disable`

函数 `exti_software_interrupt_disable` 描述见下表：

表 3-284. 函数 `exti_software_interrupt_disable`

|              |  |
|--------------|--|
| 函数名称         | <code>exti_software_interrupt_disable</code>                             |
| 函数原型         | <code>void exti_software_interrupt_disable(exti_line_enum linex);</code> |
| 功能描述         | EXTI线x软件中断事件禁能   |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| <b>linex</b> | EXTI线x，参考 <a href="#">表3-274. 枚举类型exti_line_enum</a>                     |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| -            | -  |

例如：

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

```
exti_software_interrupt_disable(EXTI_0);
```

函数 `exti_flag_get`

函数 `exti_flag_get` 描述见下表：

表 3-285. 函数 `exti_flag_get`

|                   |  |
|-------------------|--|
| 函数名称              | <code>exti_flag_get</code>                                   |
| 函数原型              | <code>FlagStatus exti_flag_get(exti_line_enum linex);</code> |
| 功能描述              | 获取EXTI线x中断标志位  |
| 先决条件              | -  |
| 被调用函数             | -  |
| 输入参数{in}          |  |
| <b>linex</b>      | EXTI线x，参考 <a href="#">表3-274. 枚举类型exti_line_enum</a>         |
| 输出参数{out}         |  |
| -                 | -  |
| 返回值               |  |
| <b>FlagStatus</b> | SET或RESET  |

例如：

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

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

## 函数 exti\_flag\_clear

函数exti\_flag\_clear描述见下表:

表 3-286. 函数 exti\_flag\_clear

|           |   |
|-----------|---|
| 函数名称      | exti_flag_clear                                       |
| 函数原型      | void exti_flag_clear(exti_line_enum linex);           |
| 功能描述      | 清除EXTI线x中断标志位   |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| linex     | EXTI线x, 参考 <a href="#">表3-274. 枚举类型exti_line_enum</a> |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如:

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

```
exti_flag_clear(EXTI_0);
```

## 函数 exti\_interrupt\_flag\_get

函数exti\_interrupt\_flag\_get描述见下表:

表 3-287. 函数 exti\_interrupt\_flag\_get

|            |   |
|------------|---|
| 函数名称       | exti_interrupt_flag_get                                   |
| 函数原型       | FlagStatus exti_interrupt_flag_get(exti_line_enum linex); |
| 功能描述       | 获取EXTI线x中断标志位   |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| linex      | EXTI线x, 参考 <a href="#">表3-274. 枚举类型exti_line_enum</a>     |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| FlagStatus | SET或RESET   |

例如:

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

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



**函数 exti\_interrupt\_flag\_clear**

函数exti\_interrupt\_flag\_clear描述见下表：

**表 3-288. 函数 exti\_interrupt\_flag\_clear**

|           |   |
|-----------|---|
| 函数名称      | exti_interrupt_flag_clear                             |
| 函数原型      | void exti_interrupt_flag_clear(exti_line_enum linex); |
| 功能描述      | 清除EXTI线x中断标志位   |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| linex     | EXTI线x, 参考 <a href="#">表3-274. 枚举类型exti_line_enum</a> |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

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

## 3.11. FMC

FMC是MCU中的Flash控制器，其中包括存储数据的主编程块和选项字节。章节[3.11.1](#)描述了FMC的寄存器列表，章节[3.11.2](#)对FMC库函数进行说明。

### 3.11.1. 外设寄存器说明

FMC寄存器列表如下表所示：

**表 3-289. FMC 寄存器**

| 寄存器名称     | 寄存器描述           |
|-----------|-----------------|
| FMC_WS    | FMC等待状态寄存器      |
| FMC_ECCCS | FMC ECC控制和状态寄存器 |
| FMC_KEY0  | FMC解锁寄存器0       |
| FMC_STAT0 | FMC状态寄存器0       |
| FMC_CTL0  | FMC控制寄存器0       |
| FMC_ADDR0 | FMC地址寄存器0       |
| FMC_OBKEY | FMC选项字节操作解锁寄存器  |
| FMC_KEY1  | FMC解锁寄存器1       |
| FMC_STAT1 | FMC状态寄存器1       |
| FMC_CTL1  | FMC控制寄存器1       |
| FMC_ADDR1 | FMC地址寄存器1       |

| 寄存器名称      | 寄存器描述            |
|------------|------------------|
| FMC_OBSTAT | FMC选项字节状态寄存器     |
| FMC_WP0    | FMC擦除/编程保护寄存器0   |
| FMC_WP1    | FMC擦除/编程保护寄存器1   |
| FMC_OB1CS  | FMC选项字节1控制和状态寄存器 |
| FMC_PID    | FMC产品ID寄存器       |

### 3.11.2. 外设库函数说明

FMC库函数列表如下表所示：

**表 3-290. FMC 库函数**

| 库函数名称                   | 库函数描述                       |
|-------------------------|-----------------------------|
| fmc_unlock              | 解锁FMC主编程块操作                 |
| fmc_bank0_unlock        | 解锁FMC主编程块bank0操作            |
| fmc_bank1_unlock        | 解锁FMC主编程块bank1操作            |
| fmc_lock                | 锁定FMC主编程块操作                 |
| fmc_bank0_lock          | 锁定FMC主编程块bank0操作            |
| fmc_bank1_lock          | 锁定FMC主编程块bank1操作            |
| fmc_wsnt_set            | 设置FMC等待状态                   |
| fmc_prefetch_enable     | 使能预取                        |
| fmc_prefetch_disable    | 禁能预取                        |
| fmc_cache_enable        | 使能cache                     |
| fmc_cache_disable       | 禁能cache                     |
| fmc_cache_reset_enable  | 在禁能cache时，使能cache复位         |
| fmc_cache_reset_disable | 禁能cache复位                   |
| fmc_powerdown_mode_set  | 当MCU进入深度睡眠模式时，flash进入到低功耗模式 |
| fmc_sleep_mode_set      | 当MCU进入深度睡眠模式时，flash进入睡眠模式   |
| fmc_sram_mode_config    | 配置共享SRAM模式                  |
| fmc_sram_mode_get       | 获取共享SRAM模式                  |
| fmc_blank_check         | 通过查空命令检查flash页是否为空          |
| fmc_page_erase          | 页擦除主编程块                     |
| fmc_bank0_mass_erase    | 擦除主编程块bank0                 |
| fmc_bank1_mass_erase    | 擦除主编程块bank1                 |
| fmc_dflash_mass_erase   | 擦除数据闪存                      |
| fmc_mass_erase          | 全片擦除                        |
| fmc_doubleword_program  | 在主编程块相应地址双字编程               |
| fmc_fast_program        | 在主编程块相应地址快速编程一行数据（32个双字）    |
| otp_doubleword_program  | 在OTP区相应地址双字编程               |
| ob_unlock               | 解锁选项字节0操作                   |
| ob_lock                 | 锁定选项字节0操作                   |
| ob_reset                | 强制对选项字节0重加载                 |
| ob_erase                | 擦除选项字节0                     |

| 库函数名称                         | 库函数描述                                     |
|-------------------------------|---|
| ob_write_protection_enable    | 使能选项字节0写保护                                |
| ob_security_protection_config | 配置安全保护                                    |
| ob_user_write                 | 编程用户选项字节                                  |
| ob_data_program               | 编程数据选项字节                                  |
| ob_user_get                   | 获取FMC_OBSTAT寄存器中FMC选项字节OB_USER的值          |
| ob_data_get                   | 获取FMC_OBSTAT寄存器中FMC选项字节 OB_DATA的值         |
| ob_write_protection_get       | 获取FMC_WP0寄存器中FMC选项字节BK0WP的值               |
| ob_bk1_write_protection_get   | 获取FMC_WP1寄存器中FMC选项字节BK1WP的值               |
| ob_df_write_protection_get    | 获取FMC_WP1寄存器中FMC选项字节DFWP的值                |
| ob_plevel_get                 | 获取FMC_OBSTAT寄存器中FMC选项字节0安全保护级别 (PLEVEL)的值 |
| ob1_lock_config               | 配置选项字节1锁定值                                |
| ob1_parameter_config          | 配置选项字节1参数                                 |
| dflash_size_get               | 获取以字节为单位的数据闪存大小                           |
| fmc_flag_get                  | 获取FMC标志状态                                 |
| fmc_flag_clear                | 清除FMC标志                                   |
| fmc_interrupt_enable          | 使能FMC中断                                   |
| fmc_interrupt_disable         | 禁能FMC中断                                   |
| fmc_interrupt_flag_get        | 获取FMC中断标志状态                               |
| fmc_interrupt_flag_clear      | 清除FMC中断标志                                 |

### 枚举类型 fmc\_state\_enum

表 3-291. 枚举类型 fmc\_state\_enum

| 枚举名称         | 枚举描述                         |
|--------------|------------------------------|
| FMC_READY    | 操作完成                         |
| FMC_BUSY     | 操作进行中                        |
| FMC_PGSERR   | 编程序列错误                       |
| FMC_PGERR    | 编程错误                         |
| FMC_PGAERR   | 编程对齐错误                       |
| FMC_WPERR    | 擦写保护错误                       |
| FMC_TOERR    | 超时错误                         |
| FMC_CBCMDERR | 被选中区域不空白错误                   |
| FMC_RSTERR   | 在擦/写flash期间，BOR/POR或系统复位产生错误 |
| FMC_OB_HSPC  | 高安全保护级别                      |
| FMC_OB1_LK   | 选项字节1锁定                      |

### 枚举类型 fmc\_sram\_mode\_enum

表 3-292. 枚举类型 fmc\_sram\_mode\_enum

| 枚举名称         | 枚举描述      |
|--------------|-----------|
| NO_SRAM_MODE | SRAM模式未配置 |

| 枚举名称                 | 枚举描述       |
|----------------------|------------|
| FASTPG_SRAM_M<br>ODE | 快速编程SRAM模式 |
| BASIC_SRAM_MO<br>DE  | 基本SRAM模式   |

### 枚举类型 `fmc_area_enum`

表 3-293. 枚举类型 `fmc_area_enum`

| 枚举名称                | 枚举描述       |
|---------------------|------------|
| BANK0_AREA          | 主编程块bank0区 |
| BANK1_AREA          | 主编程块bank1区 |
| DATA_FLASH_ARE<br>A | 数据闪存区      |

### 枚举类型 `fmc_flag_enum`

表 3-294. 枚举类型 `fmc_flag_enum`

| 枚举名称                        | 枚举描述                          |
|-----------------------------|-------------------------------|
| FMC_BANK0_FLAG<br>_BUSY     | 闪存bank0忙标志                    |
| FMC_BANK0_FLAG<br>_PGSERR   | 闪存bank0编程序列错误标志               |
| FMC_BANK0_FLAG<br>_PGERR    | 闪存bank0编程错误标志                 |
| FMC_BANK0_FLAG<br>_PGAERR   | 闪存bank0编程对齐错误标志               |
| FMC_BANK0_FLAG<br>_WPERR    | 闪存bank0擦写保护错误标志               |
| FMC_BANK0_FLAG<br>_END      | 闪存bank0操作结束标志                 |
| FMC_BANK0_FLAG<br>_CBCMDERR | 闪存bank0被查空命令选中的区域都是0xFF或不是标志  |
| FMC_BANK0_FLAG<br>_RSTERR   | 闪存bank0 BOR/POR或系统复位期间擦除/程序标志 |
| FMC_BANK1_FLAG<br>_BUSY     | 闪存bank1忙标志                    |
| FMC_BANK1_FLAG<br>_PGSERR   | 闪存bank1编程序列错误标志               |
| FMC_BANK1_FLAG<br>_PGERR    | 闪存bank1编程错误标志                 |
| FMC_BANK1_FLAG<br>_PGAERR   | 闪存bank1编程对齐错误标志               |

| 枚举名称                    | 枚举描述                            |
|-------------------------|---------------------------------|
| FMC_BANK1_FLAG_WPERR    | 闪存bank1擦写保护错误标志                 |
| FMC_BANK1_FLAG_END      | 闪存bank1操作结束标志                   |
| FMC_BANK1_FLAG_CBCMDERR | 闪存bank1被查空命令选中的区域都是0xFF或不是标志    |
| FMC_BANK1_FLAG_RSTERR   | 闪存bank1 BOR/POR或系统复位期间擦除/程序标志   |
| FMC_FLAG_OB0ECC         | 选项字节0检测到ECC位错误标志                |
| FMC_FLAG_BK1ECC         | bank1检测到ECC位错误标志                |
| FMC_FLAG_SYSECC         | system区检测到ECC位错误标志              |
| FMC_FLAG_DFEC           | 数据闪存区检测到ECC位错误标志                |
| FMC_FLAG_OTPECC         | OTP区检测到ECC位错误标志                 |
| FMC_FLAG_OB1ECCDET      | 选项字节1检测到两个位ECC错误标志              |
| FMC_FLAG_OB0ECCDET      | 选项字节0检测到两个位ECC错误标志              |
| FMC_FLAG_ECCCOR         | 一个位错误检测和纠正标志                    |
| FMC_FLAG_ECCDET         | OTP/数据闪存/system区/bank1检测到两位错误标志 |
| FMC_FLAG_OBERR          | 选项字节0错误标志                       |
| FMC_FLAG_OB1ERR         | 选项字节1读取错误标志                     |

### 枚举类型 `fmc_interrupt_flag_enum`

表 3-295. 枚举类型 `fmc_interrupt_flag_enum`

| 枚举名称                      | 枚举描述              |
|---------------------------|-------------------|
| FMC_BANK0_INT_FLAG_PGSERR | 闪存bank0编程序列错误中断标志 |
| FMC_BANK0_INT_FLAG_PGERR  | 闪存bank0编程错误中断标志   |
| FMC_BANK0_INT_FLAG_PGAERR | 闪存bank0编程对齐错误中断标志 |
| FMC_BANK0_INT_FLAG_WPERR  | 闪存bank0擦写保护错误中断标志 |

| 枚举名称                        | 枚举描述                              |
|-----------------------------|-----------------------------------|
| FLAG_WPERR                  |                                   |
| FMC_BANK0_INT_FLAG_END      | 闪存bank0操作结束中断标志                   |
| FMC_BANK0_INT_FLAG_CBCMDERR | 闪存bank0被查空命令选中的区域都是0xFF或不是中断标志    |
| FMC_BANK0_INT_FLAG_RSTERR   | 闪存bank0 BOR/POR或系统复位期间擦除/程序中断标志   |
| FMC_BANK1_INT_FLAG_PGSERR   | 闪存bank1编程序列错误中断标志                 |
| FMC_BANK1_INT_FLAG_PGERR    | 闪存bank1编程错误中断标志                   |
| FMC_BANK1_INT_FLAG_PGAERR   | 闪存bank1编程对齐错误中断标志                 |
| FMC_BANK1_INT_FLAG_WPERR    | 闪存bank1擦写保护错误中断标志                 |
| FMC_BANK1_INT_FLAG_END      | 闪存bank1操作结束中断标志                   |
| FMC_BANK1_INT_FLAG_CBCMDERR | 闪存bank1被查空命令选中的区域都是0xFF或不是中断标志    |
| FMC_BANK1_INT_FLAG_RSTERR   | 闪存bank1 BOR/POR或系统复位期间擦除/程序中断标志   |
| FMC_INT_FLAG_OB1ECCDET      | 选项字节1检测到两个位ECC错误中断标志              |
| FMC_INT_FLAG_OB0ECCDET      | 选项字节0检测到两个位ECC错误中断标志              |
| FMC_INT_FLAG_ECCOR          | 一个位错误检测和纠正中断标志                    |
| FMC_INT_FLAG_ECCDET         | OTP/数据闪存/system区/bank1检测到两位错误中断标志 |

### 枚举类型 `fmc_interrupt_enum`

表 3-296. 枚举类型 `fmc_interrupt_enum`

| 枚举名称              | 枚举描述            |
|-------------------|-----------------|
| FMC_BANK0_INT_ERR | FMC bank0错误中断   |
| FMC_BANK0_INT_END | FMC bank0操作结束中断 |
| FMC_BANK1_INT_ERR | FMC bank1错误中断   |
| FMC_BANK1_INT_END | FMC bank1操作结束中断 |

| 枚举名称           | 枚举描述          |
|----------------|---------------|
| FMC_INT_ECCCOR | FMC一个位错误并纠正中断 |
| FMC_INT_ECCDET | FMC两个位错误中断    |

## 函数 fmc\_unlock

函数fmc\_unlock描述见下表：

**表 3-297. 函数 fmc\_unlock**

|           |                        |
|-----------|------------------------|
| 函数名称      | fmc_unlock             |
| 函数原形      | void fmc_unlock(void); |
| 功能描述      | 解锁FMC主编程块操作            |
| 先决条件      | -                      |
| 被调用函数     | -                      |
| 输入参数{in}  |                        |
| -         | -                      |
| 输出参数{out} |                        |
| -         | -                      |
| 返回值       |                        |
| -         | -                      |

例如：

```
/* unlock the main flash operation */
fmc_unlock();
```

## 函数 fmc\_bank0\_unlock

函数fmc\_bank0\_unlock描述见下表：

**表 3-298. 函数 fmc\_bank0\_unlock**

|           |                              |
|-----------|------------------------------|
| 函数名称      | fmc_bank0_unlock             |
| 函数原形      | void fmc_bank0_unlock(void); |
| 功能描述      | 解锁FMC主编程块bank0操作             |
| 先决条件      | -                            |
| 被调用函数     | -                            |
| 输入参数{in}  |                              |
| -         | -                            |
| 输出参数{out} |                              |
| -         | -                            |
| 返回值       |                              |
| -         | -                            |

例如：

```
/* unlock the main flash bank0 operation */
```

```
fmc_bank0_unlock();
```

### 函数 fmc\_bank1\_unlock

函数fmc\_bank1\_unlock描述见下表：

**表 3-299. 函数 fmc\_bank1\_unlock**

|           |                              |
|-----------|------------------------------|
| 函数名称      | fmc_bank1_unlock             |
| 函数原形      | void fmc_bank1_unlock(void); |
| 功能描述      | 解锁FMC主编程块bank1操作             |
| 先决条件      | -                            |
| 被调用函数     | -                            |
| 输入参数{in}  |                              |
| -         | -                            |
| 输出参数{out} |                              |
| -         | -                            |
| 返回值       |                              |
| -         | -                            |

例如：

```
/* unlock the main flash bank1 operation */
```

```
fmc_bank1_unlock();
```

### 函数 fmc\_lock

函数fmc\_lock描述见下表：

**表 3-300. 函数 fmc\_lock**

|           |                      |
|-----------|----------------------|
| 函数名称      | fmc_lock             |
| 函数原形      | void fmc_lock(void); |
| 功能描述      | 锁定FMC主编程块操作          |
| 先决条件      | -                    |
| 被调用函数     | -                    |
| 输入参数{in}  |                      |
| -         | -                    |
| 输出参数{out} |                      |
| -         | -                    |
| 返回值       |                      |
| -         | -                    |

例如：

```
/* lock the main flash operation */
```



```
fmc_lock();
```

### 函数 fmc\_bank0\_lock

函数fmc\_bank0\_lock描述见下表：

**表 3-301. 函数 fmc\_bank0\_lock**

|                       |                            |
|-----------------------|----------------------------|
| Function name         | fmc_bank0_lock             |
| Function prototype    | void fmc_bank0_lock(void); |
| Function descriptions | 锁定FMC主编程块bank0操作           |
| Precondition          | -                          |
| The called functions  | -                          |
| 输入参数{in}              |                            |
| -                     | -                          |
| 输出参数{out}             |                            |
| -                     | -                          |
| 返回值                   |                            |
| -                     | -                          |

例如：

```
/* lock the main flash bank0 operation */
```

```
fmc_bank0_lock();
```

### 函数 fmc\_bank1\_lock

函数fmc\_bank1\_lock描述见下表：

**表 3-302. 函数 fmc\_bank1\_lock**

|           |                            |
|-----------|----------------------------|
| 函数名称      | fmc_bank1_lock             |
| 函数原形      | void fmc_bank1_lock(void); |
| 功能描述      | 锁定FMC主编程块bank1操作           |
| 先决条件      | -                          |
| 被调用函数     | -                          |
| 输入参数{in}  |                            |
| -         | -                          |
| 输出参数{out} |                            |
| -         | -                          |
| 返回值       |                            |
| -         | -                          |

例如：

```
/* lock the main flash bank1 operation */
```

```
fmc_bank1_lock();
```

## 函数 fmc\_wscnt\_set

函数fmc\_wscnt\_set描述见下表:

表 3-303. 函数 fmc\_wscnt\_set

|            |                                     |
|------------|-------------------------------------|
| 函数名称       | fmc_wscnt_set                       |
| 函数原形       | void fmc_wscnt_set(uint32_t wscnt); |
| 功能描述       | 设置FMC等待状态                           |
| 先决条件       | -                                   |
| 被调用函数      | -                                   |
| 输入参数{in}   |                                     |
| wscnt      | 等待状态计数值                             |
| WS_WSCNT_0 | 0个等待状态                              |
| WS_WSCNT_1 | 1个等待状态                              |
| WS_WSCNT_2 | 2个等待状态                              |
| WS_WSCNT_3 | 3个等待状态                              |
| 输出参数{out}  |                                     |
| -          | -                                   |
| 返回值        |                                     |
| -          | -                                   |

例如:

```
/* set the wait state counter value */
fmc_wscnt_set(WS_WSCNT_1);
```

## 函数 fmc\_prefetch\_enable

函数fmc\_prefetch\_enable描述见下表:

表 3-304. 函数 fmc\_prefetch\_enable

|           |                                 |
|-----------|---------------------------------|
| 函数名称      | fmc_prefetch_enable             |
| 函数原形      | void fmc_prefetch_enable(void); |
| 功能描述      | 使能预取                            |
| 先决条件      | -                               |
| 被调用函数     | -                               |
| 输入参数{in}  |                                 |
| -         | -                               |
| 输出参数{out} |                                 |
| -         | -                               |
| 返回值       |                                 |
| -         | -                               |

例如:

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

```
fmc_prefetch_enable();
```

### 函数 fmc\_prefetch\_disable

函数fmc\_prefetch\_disable描述见下表：

**表 3-305. 函数 fmc\_prefetch\_disable**

|           |                                  |
|-----------|----------------------------------|
| 函数名称      | fmc_prefetch_disable             |
| 函数原形      | void fmc_prefetch_disable(void); |
| 功能描述      | 禁能预取                             |
| 先决条件      | -                                |
| 被调用函数     | -                                |
| 输入参数{in}  |                                  |
| -         | -                                |
| 输出参数{out} |                                  |
| -         | -                                |
| 返回值       |                                  |
| -         | -                                |

例如：

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

```
fmc_prefetch_disable();
```

### 函数 fmc\_cache\_enable

函数fmc\_cache\_enable描述见下表：

**表 3-306. 函数 fmc\_cache\_enable**

|           |                              |
|-----------|------------------------------|
| 函数名称      | fmc_cache_enable             |
| 函数原形      | void fmc_cache_enable(void); |
| 功能描述      | 使能cache                      |
| 先决条件      | -                            |
| 被调用函数     | -                            |
| 输入参数{in}  |                              |
| -         | -                            |
| 输出参数{out} |                              |
| -         | -                            |
| 返回值       |                              |
| -         | -                            |

例如：

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

```
fmc_cache_enable();
```

**函数 fmc\_cache\_disable**

函数fmc\_cache\_enable描述见下表：

**表 3-307. 函数 fmc\_cache\_disable**

|           |                               |
|-----------|-------------------------------|
| 函数名称      | fmc_cache_disable             |
| 函数原形      | void fmc_cache_disable(void); |
| 功能描述      | 禁能cache                       |
| 先决条件      | -                             |
| 被调用函数     | -                             |
| 输入参数{in}  |                               |
| -         | -                             |
| 输出参数{out} |                               |
| -         | -                             |
| 返回值       |                               |
| -         | -                             |

例如：

```
/* disable cache */  
  
fmc_cache_disable();
```

**函数 fmc\_cache\_reset\_enable**

函数fmc\_cache\_reset\_enable描述见下表：

**表 3-308. 函数 fmc\_cache\_reset\_enable**

|           |                                    |
|-----------|------------------------------------|
| 函数名称      | fmc_cache_reset_enable             |
| 函数原形      | void fmc_cache_reset_enable(void); |
| 功能描述      | 在禁能cache时，使能cache复位                |
| 先决条件      | -                                  |
| 被调用函数     | -                                  |
| 输入参数{in}  |                                    |
| -         | -                                  |
| 输出参数{out} |                                    |
| -         | -                                  |
| 返回值       |                                    |
| -         | -                                  |

例如：

```
/* enable cache reset if cache is disabled */  
  
fmc_cache_reset_enable();
```

**函数 fmc\_cache\_reset\_disable**

函数fmc\_cache\_reset\_disable描述见下表:

**表 3-309. 函数 fmc\_cache\_reset\_disable**

|           |                                     |
|-----------|-------------------------------------|
| 函数名称      | fmc_cache_reset_disable             |
| 函数原形      | void fmc_cache_reset_disable(void); |
| 功能描述      | 禁能cache复位                           |
| 先决条件      | -                                   |
| 被调用函数     | -                                   |
| 输入参数{in}  |                                     |
| -         | -                                   |
| 输出参数{out} |                                     |
| -         | -                                   |
| 返回值       |                                     |
| -         | -                                   |

例如:

```
/* disable cache reset */
```

```
fmc_cache_reset_disable();
```

**函数 fmc\_powerdown\_mode\_set**

函数fmc\_powerdown\_mode\_set描述见下表:

**表 3-310. 函数 fmc\_powerdown\_mode\_set**

|           |                                    |
|-----------|------------------------------------|
| 函数名称      | fmc_powerdown_mode_set             |
| 函数原形      | void fmc_powerdown_mode_set(void); |
| 功能描述      | 当MCU进入深度睡眠模式时, flash进入到低功耗模式       |
| 先决条件      | -                                  |
| 被调用函数     | -                                  |
| 输入参数{in}  |                                    |
| -         | -                                  |
| 输出参数{out} |                                    |
| -         | -                                  |
| 返回值       |                                    |
| -         | -                                  |

例如:

```
/* flash goto power-down mode when MCU enters deepsleep mode */
```

```
fmc_powerdown_mode_set();
```

**函数 fmc\_sleep\_mode\_set**

函数fmc\_sleep\_mode\_set描述见下表:

**表 3-311. 函数 fmc\_sleep\_mode\_set**

|           |                                |
|-----------|--------------------------------|
| 函数名称      | fmc_sleep_mode_set             |
| 函数原形      | void fmc_sleep_mode_set(void); |
| 功能描述      | 当MCU进入深度睡眠模式时，flash进入睡眠模式      |
| 先决条件      | -                              |
| 被调用函数     | -                              |
| 输入参数{in}  |                                |
| -         | -                              |
| 输出参数{out} |                                |
| -         | -                              |
| 返回值       |                                |
| -         | -                              |

例如:

```
/* flash goto sleep mode when MCU enters deepsleep mode */
```

```
fmc_sleep_mode_set();
```

**函数 fmc\_sram\_mode\_config**

函数fmc\_sram\_mode\_config描述见下表:

**表 3-312. 函数 fmc\_sram\_mode\_config**

|                      |  |
|----------------------|--|
| 函数名称                 | fmc_sram_mode_config                                     |
| 函数原形                 | void fmc_sram_mode_config(fmc_sram_mode_enum sram_mode); |
| 功能描述                 | 配置共享SRAM模式   |
| 先决条件                 | -  |
| 被调用函数                | -  |
| 输入参数{in}             |  |
| sram_mode            | 共享SRAM模式   |
| FASTPG_SRAM_MOD<br>E | 快速编程SRAM模式   |
| BASIC_SRAM_MODE      | 基本SRAM模式   |
| 输出参数{out}            |  |
| -                    | -  |
| 返回值                  |  |
| -                    | -  |

例如:

```
/* configure shared SRAM as fast program SRAM */
```

fmc\_sram\_mode\_config(FASTPG\_SRAM\_MODE);

### 函数 fmc\_sram\_mode\_get

函数fmc\_sram\_mode\_get描述见下表:

表 3-313. 函数 fmc\_sram\_mode\_get

|                    |   |
|--------------------|---|
| 函数名称               | fmc_sram_mode_get                           |
| 函数原形               | fmc_sram_mode_enum fmc_sram_mode_get(void); |
| 功能描述               | 获取共享SRAM模式                                  |
| 先决条件               | -   |
| 被调用函数              | -   |
| 输入参数{in}           |   |
| -                  | -   |
| 输出参数{out}          |   |
| fmc_sram_mode_enum | 共享SRAM模式                                    |
| FASTPG_SRAM_MODE   | 快速编程SRAM模式                                  |
| BASIC_SRAM_MODE    | 基本SRAM模式                                    |
| 返回值                |   |
| -                  | -   |

例如:

```
/* get shared SRAM mode */
```

```
fmc_sram_mode_enum mode;
```

```
mode = fmc_sram_mode_get();
```

### 函数 fmc\_blank\_check

函数fmc\_blank\_check描述见下表:

表 3-314. 函数 fmc\_blank\_check

|          |   |
|----------|---|
| 函数名称     | fmc_blank_check   |
| 函数原形     | fmc_state_enum fmc_blank_check(uint32_t address, uint8_t length); |
| 功能描述     | 通过查空命令检查flash页是否为空  |
| 先决条件     | -   |
| 被调用函数    | -   |
| 输入参数{in} |   |
| address  | 检查的起始地址   |
| 输入参数{in} |   |
| length   | 读取长度为2^length个双字, 需要检查的flash区域必须在一个页面内, 且不应超过1KB的边界               |

| 输出参数{out}             |                              |
|-----------------------|------------------------------|
| -                     | -                            |
| 返回值                   |                              |
| <b>fmc_state_enum</b> | FMC状态                        |
| <i>FMC_READY</i>      | 操作完成                         |
| <i>FMC_BUSY</i>       | 操作进行中                        |
| <i>FMC_PGSERR</i>     | 编程序列错误                       |
| <i>FMC_PGERR</i>      | 编程错误                         |
| <i>FMC_PGAERR</i>     | 编程对齐错误                       |
| <i>FMC_WPERR</i>      | 擦写保护错误                       |
| <i>FMC_TOERR</i>      | 超时错误                         |
| <i>FMC_CBCMDERR</i>   | 被选中区域不空白错误                   |
| <i>FMC_RSTERR</i>     | 在擦/写flash期间，BOR/POR或系统复位产生错误 |

例如：

```
/* check whether flash page is blank or not by check blank command */
```

```
fmc_state_enum state;
```

```
state = fmc_blank_check(0x8004000, 4);
```

### 函数 fmc\_page\_erase

函数fmc\_blank\_check描述见下表：

表 3-315. 函数 fmc\_page\_erase

| 函数名称                  | fmc_page_erase  |
|-----------------------|---|
| 函数原形                  | fmc_state_enum fmc_page_erase(uint32_t page_address); |
| 功能描述                  | 页擦除主编程块   |
| 先决条件                  | fmc_unlock  |
| 被调用函数                 | -   |
| 输入参数{in}              |   |
| <b>page_address</b>   | 页擦除地址   |
| 输出参数{out}             |   |
| -                     | -   |
| 返回值                   |   |
| <b>fmc_state_enum</b> | FMC状态   |
| <i>FMC_READY</i>      | 操作完成  |
| <i>FMC_BUSY</i>       | 操作进行中   |
| <i>FMC_PGSERR</i>     | 编程序列错误  |
| <i>FMC_PGERR</i>      | 编程错误  |
| <i>FMC_PGAERR</i>     | 编程对齐错误  |
| <i>FMC_WPERR</i>      | 擦写保护错误  |
| <i>FMC_TOERR</i>      | 超时错误  |



|                     |                              |
|---------------------|------------------------------|
| <i>FMC_CBCMDERR</i> | 被选中区域不空白错误                   |
| <i>FMC_RSTERR</i>   | 在擦/写flash期间，BOR/POR或系统复位产生错误 |

例如：

```
fmc_unlock();
```

```
/* erase page */
```

```
fmc_state_enum state = fmc_page_erase(0x08004000);
```

### 函数 **fmc\_bank0\_mass\_erase**

函数fmc\_bank0\_mass\_erase描述见下表：

**表 3-316. 函数 fmc\_bank0\_mass\_erase**

|                       |  |
|-----------------------|--|
| 函数名称                  | fmc_bank0_mass_erase                       |
| 函数原形                  | fmc_state_enum fmc_bank0_mass_erase(void); |
| 功能描述                  | 擦除主编程块bank0                                |
| 先决条件                  | fmc_unlock                                 |
| 被调用函数                 | -  |
| 输入参数{in}              |  |
| -                     | -  |
| 输出参数{out}             |  |
| -                     | -  |
| 返回值                   |  |
| <b>fmc_state_enum</b> | FMC状态                                      |
| <i>FMC_READY</i>      | 操作完成                                       |
| <i>FMC_BUSY</i>       | 操作进行中                                      |
| <i>FMC_PGSERR</i>     | 编程序列错误                                     |
| <i>FMC_PGERR</i>      | 编程错误                                       |
| <i>FMC_PGAERR</i>     | 编程对齐错误                                     |
| <i>FMC_WPERR</i>      | 擦写保护错误                                     |
| <i>FMC_TOERR</i>      | 超时错误                                       |
| <i>FMC_CBCMDERR</i>   | 被选中区域不空白错误                                 |
| <i>FMC_RSTERR</i>     | 在擦/写flash期间，BOR/POR或系统复位产生错误               |

例如：

```
fmc_unlock();
```

```
/* erase flash bank0 */
```

```
fmc_state_enum state = fmc_bank0_mass_erase();
```

### 函数 **fmc\_bank1\_mass\_erase**

函数fmc\_bank1\_mass\_erase描述见下表：

表 3-317. 函数 `fmc_bank1_mass_erase`

|                             |   |
|-----------------------------|---|
| 函数名称                        | <code>fmc_bank1_mass_erase</code>                       |
| 函数原形                        | <code>fmc_state_enum fmc_bank1_mass_erase(void);</code> |
| 功能描述                        | 擦除主编程块bank1   |
| 先决条件                        | <code>fmc_unlock</code>                                 |
| 被调用函数                       | -   |
| 输入参数{in}                    |   |
| -                           | -   |
| 输出参数{out}                   |   |
| -                           | -   |
| 返回值                         |   |
| <code>fmc_state_enum</code> | FMC状态   |
| <code>FMC_READY</code>      | 操作完成  |
| <code>FMC_BUSY</code>       | 操作进行中   |
| <code>FMC_PGSERR</code>     | 编程序列错误  |
| <code>FMC_PGERR</code>      | 编程错误  |
| <code>FMC_PGAERR</code>     | 编程对齐错误  |
| <code>FMC_WPERR</code>      | 擦写保护错误  |
| <code>FMC_TOERR</code>      | 超时错误  |
| <code>FMC_CBCMDERR</code>   | 被选中区域不空白错误  |
| <code>FMC_RSTERR</code>     | 在擦/写flash期间，BOR/POR或系统复位产生错误                            |

例如：

```
fmc_unlock();
```

```
/* erase flash bank1 */
```

```
fmc_state_enum state = fmc_bank1_mass_erase();
```

### 函数 `fmc_dflash_mass_erase`

函数`fmc_dflash_mass_erase`描述见下表：

表 3-318. 函数 `fmc_dflash_mass_erase`

|           |  |
|-----------|--|
| 函数名称      | <code>fmc_dflash_mass_erase</code>                       |
| 函数原形      | <code>fmc_state_enum fmc_dflash_mass_erase(void);</code> |
| 功能描述      | 擦除数据闪存   |
| 先决条件      | <code>fmc_unlock</code>                                  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |

|                       |                              |
|-----------------------|------------------------------|
| <b>fmc_state_enum</b> | FMC状态                        |
| <i>FMC_READY</i>      | 操作完成                         |
| <i>FMC_BUSY</i>       | 操作进行中                        |
| <i>FMC_PGSERR</i>     | 编程序列错误                       |
| <i>FMC_PGERR</i>      | 编程错误                         |
| <i>FMC_PGAERR</i>     | 编程对齐错误                       |
| <i>FMC_WPERR</i>      | 擦写保护错误                       |
| <i>FMC_TOERR</i>      | 超时错误                         |
| <i>FMC_CBCMDERR</i>   | 被选中区域不空白错误                   |
| <i>FMC_RSTERR</i>     | 在擦/写flash期间，BOR/POR或系统复位产生错误 |

例如：

```
fmc_unlock();
```

```
/* erase the data flash */
```

```
fmc_state_enum state = fmc_dflash_mass_erase();
```

### 函数 fmc\_mass\_erase

函数fmc\_mass\_erase描述见下表：

表 3-319. 函数 fmc\_mass\_erase

|                       |                                      |
|-----------------------|--------------------------------------|
| 函数名称                  | fmc_mass_erase                       |
| 函数原形                  | fmc_state_enum fmc_mass_erase(void); |
| 功能描述                  | 全片擦除                                 |
| 先决条件                  | fmc_unlock                           |
| 被调用函数                 | -                                    |
| 输入参数{in}              |                                      |
| -                     | -                                    |
| 输出参数{out}             |                                      |
| -                     | -                                    |
| 返回值                   |                                      |
| <b>fmc_state_enum</b> | FMC状态                                |
| <i>FMC_READY</i>      | 操作完成                                 |
| <i>FMC_BUSY</i>       | 操作进行中                                |
| <i>FMC_PGSERR</i>     | 编程序列错误                               |
| <i>FMC_PGERR</i>      | 编程错误                                 |
| <i>FMC_PGAERR</i>     | 编程对齐错误                               |
| <i>FMC_WPERR</i>      | 擦写保护错误                               |
| <i>FMC_TOERR</i>      | 超时错误                                 |
| <i>FMC_CBCMDERR</i>   | 被选中区域不空白错误                           |
| <i>FMC_RSTERR</i>     | 在擦/写flash期间，BOR/POR或系统复位产生错误         |

例如：

```
fmc_unlock();

/* erase whole chip */

fmc_state_enum state = fmc_mass_erase();
```

### 函数 fmc\_doubleword\_program

函数fmc\_doubleword\_program描述见下表：

表 3-320. 函数 fmc\_doubleword\_program

| 函数名称                  | fmc_doubleword_program  |
|-----------------------|---|
| 函数原形                  | fmc_state_enum fmc_doubleword_program(uint32_t address, uint64_t data); |
| 功能描述                  | 在主编程块相应地址双字编程   |
| 先决条件                  | fmc_unlock  |
| 被调用函数                 | -   |
| 输入参数{in}              |   |
| <b>address</b>        | 编程地址  |
| 输入参数{in}              |   |
| <b>data</b>           | 编程数据  |
| 输出参数{out}             |   |
| -                     | -   |
| 返回值                   |   |
| <b>fmc_state_enum</b> | FMC状态   |
| <i>FMC_READY</i>      | 操作完成  |
| <i>FMC_BUSY</i>       | 操作进行中   |
| <i>FMC_PGSERR</i>     | 编程序列错误  |
| <i>FMC_PGERR</i>      | 编程错误  |
| <i>FMC_PGAERR</i>     | 编程对齐错误  |
| <i>FMC_WPERR</i>      | 擦写保护错误  |
| <i>FMC_TOERR</i>      | 超时错误  |
| <i>FMC_CBCMDERR</i>   | 被选中区域不空白错误  |
| <i>FMC_RSTERR</i>     | 在擦/写flash期间，BOR/POR或系统复位产生错误  |

例如：

```
fmc_unlock();

fmc_page_erase(0x08004000);

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

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

函数 `fmc_fast_program`

函数 `fmc_fast_program` 描述见下表：

表 3-321. 函数 `fmc_fast_program`

|                       |  |
|-----------------------|--|
| 函数名称                  | <code>fmc_fast_program</code>  |
| 函数原形                  | <code>fmc_state_enum fmc_fast_program(uint32_t address, uint64_t data[]);</code> |
| 功能描述                  | 在主编程块相应地址快速编程一行数据（32个双字）   |
| 先决条件                  | <code>fmc_unlock</code>  |
| 被调用函数                 | -  |
| 输入参数{in}              |  |
| <b>address</b>        | 编程地址   |
| 输入参数{in}              |  |
| <b>data</b>           | 编程数据   |
| 输出参数{out}             |  |
| -                     | -  |
| 返回值                   |  |
| <b>fmc_state_enum</b> | FMC状态  |
| <i>FMC_READY</i>      | 操作完成   |
| <i>FMC_BUSY</i>       | 操作进行中  |
| <i>FMC_PGSERR</i>     | 编程序列错误   |
| <i>FMC_PGERR</i>      | 编程错误   |
| <i>FMC_PGAERR</i>     | 编程对齐错误   |
| <i>FMC_WPERR</i>      | 擦写保护错误   |
| <i>FMC_TOERR</i>      | 超时错误   |
| <i>FMC_CBCMDERR</i>   | 被选中区域不空白错误   |
| <i>FMC_RSTERR</i>     | 在擦/写flash期间，BOR/POR或系统复位产生错误   |

例如：

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

```
static uint64_t data_buffer[32] = {
    0x0000000000000000U, 0x1111111111111111U, 0x2222222222222222U, 0x3333333333333333U,
    0x4444444444444444U, 0x5555555555555555U, 0x6666666666666666U, 0x7777777777777777U,
    0x8888888888888888U, 0x9999999999999999U, 0xAAAAAAAAAAAAAAAAAU, 0xBBBBBBBBBBBBBBBU,
    0xCCCCCCCCCCCCCCCCCU, 0xDDDDDDDDDDDDDDDDDU, 0xEEEEEEEEEEEEEEEEU, 0xFFFFFFFFFFFUFU,
    0x0011001100110011U, 0x2233223322332233U, 0x4455445544554455U, 0x667766
```

```
7766776677U,
```

```
0x8899889988998899U, 0xAABBAABBAABBAABBU, 0xCCDDCCDDCCDDCCDDU,
0xEEFFEEFFEEFFEEFFU,
```

```
0x2200220022002200U, 0x3311331133113311U, 0x6644664466446644U, 0x775577
5577557755U,
```

```
0xAA88AA88AA88AA88U, 0xBB99BB99BB99BB99U, 0xEECC EECC EECC EECCU,
0xFFDDFFDDFFDDFFDDU
```

```
};
```

```
fmc_unlock();
```

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

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

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

### 函数 otp\_doubleword\_program

函数otp\_doubleword\_program描述见下表：

表 3-322. 函数 otp\_doubleword\_program

|                |   |
|----------------|---|
| 函数名称           | otp_doubleword_program  |
| 函数原形           | fmc_state_enum otp_doubleword_program(uint32_t address, uint64_t data); |
| 功能描述           | 在OTP区相应地址双字编程   |
| 先决条件           | fmc_unlock  |
| 被调用函数          | -   |
| 输入参数{in}       |   |
| address        | 编程地址  |
| 输入参数{in}       |   |
| data           | 编程数据  |
| 输出参数{out}      |   |
| -              | -   |
| 返回值            |   |
| fmc_state_enum | FMC状态   |
| FMC_READY      | 操作完成  |
| FMC_BUSY       | 操作进行中   |
| FMC_PGSERR     | 编程序列错误  |
| FMC_PGERR      | 编程错误  |
| FMC_PGAERR     | 编程对齐错误  |
| FMC_WPERR      | 擦写保护错误  |
| FMC_TOERR      | 超时错误  |
| FMC_CBCMDERR   | 被选中区域不空白错误  |

*FMC\_RSTERR*

在擦/写flash期间，BOR/POR或系统复位产生错误

例如：

```
fmc_unlock();
```

```
/* program a double word at the corresponding address in OTP */
```

```
fmc_state_enum fmc_state = otp_doubleword_program (0x1FFF7000, 0x11223344aabbccdd);
```

### 函数 **ob\_unlock**

函数ob\_unlock描述见下表：

**表 3-323. 函数 ob\_unlock**

|           |                       |
|-----------|-----------------------|
| 函数名称      | ob_unlock             |
| 函数原形      | void ob_unlock(void); |
| 功能描述      | 解锁选项字节操作              |
| 先决条件      | fmc_unlock            |
| 被调用函数     | -                     |
| 输入参数{in}  |                       |
| -         | -                     |
| 输出参数{out} |                       |
| -         | -                     |
| 返回值       |                       |
| -         | -                     |

例如：

```
fmc_unlock();
```

```
/* unlock the option byte 0 operation */
```

```
ob_unlock();
```

### 函数 **ob\_lock**

函数ob\_lock描述见下表：

**表 3-324. 函数 ob\_lock**

|           |                     |
|-----------|---------------------|
| 函数名称      | ob_lock             |
| 函数原形      | void ob_lock(void); |
| 功能描述      | 锁定选项字节操作            |
| 先决条件      | -                   |
| 被调用函数     | -                   |
| 输入参数{in}  |                     |
| -         | -                   |
| 输出参数{out} |                     |

|     |   |
|-----|---|
| -   | - |
| 返回值 |   |
| -   | - |

例如：

```
/*lock the option byte 0 operation */
```

```
ob_lock();
```

### 函数 ob\_reset

函数ob\_reset描述见下表：

**表 3-325. 函数 ob\_reset**

|           |                      |
|-----------|----------------------|
| 函数名称      | ob_reset             |
| 函数原形      | void ob_reset(void); |
| 功能描述      | 强制对选项字节0重加载          |
| 先决条件      | -                    |
| 被调用函数     | -                    |
| 输入参数{in}  |                      |
| -         | -                    |
| 输出参数{out} |                      |
| -         | -                    |
| 返回值       |                      |
| -         | -                    |

例如：

```
/* force to reload the option byte 0 */
```

```
ob_reset();
```

### 函数 ob\_erase

函数ob\_erase描述见下表：

**表 3-326. 函数 ob\_erase**

|           |                      |
|-----------|----------------------|
| 函数名称      | ob_erase             |
| 函数原形      | void ob_erase(void); |
| 功能描述      | 擦除选项字节0              |
| 先决条件      | ob_unlock            |
| 被调用函数     | -                    |
| 输入参数{in}  |                      |
| -         | -                    |
| 输出参数{out} |                      |
| -         | -                    |



| 返回值                   |                              |
|-----------------------|------------------------------|
| <b>fmc_state_enum</b> | FMC状态                        |
| <i>FMC_READY</i>      | 操作完成                         |
| <i>FMC_BUSY</i>       | 操作进行中                        |
| <i>FMC_PGSEERR</i>    | 编程序列错误                       |
| <i>FMC_PGERR</i>      | 编程错误                         |
| <i>FMC_PGAERR</i>     | 编程对齐错误                       |
| <i>FMC_WPERR</i>      | 擦写保护错误                       |
| <i>FMC_TOERR</i>      | 超时错误                         |
| <i>FMC_CBCMDERR</i>   | 被选中区域不空白错误                   |
| <i>FMC_RSTERR</i>     | 在擦/写flash期间，BOR/POR或系统复位产生错误 |
| <i>FMC_OB_HSPC</i>    | 高安全保护级别                      |

例如：

```
fmc_unlock();

ob_unlock();

/* erase the option byte 0 */

fmc_state_enum fmc_state = ob_erase();
```

### 函数 ob\_write\_protection\_enable

函数ob\_write\_protection\_enable描述见下表：

表 3-327. 函数 ob\_write\_protection\_enable

|                       |   |
|-----------------------|---|
| 函数名称                  | ob_write_protection_enable  |
| 函数原形                  | fmc_state_enum ob_write_protection_enable(fmc_area_enum wp_area, uint32_t ob_wp); |
| 功能描述                  | 使能选项字节0写保护  |
| 先决条件                  | ob_unlock   |
| 被调用函数                 | -   |
| 输入参数{in}              |   |
| <b>wp_area</b>        | 写保护区域，参考 <a href="#">表3-293. 枚举类型fmc_area_enum</a> .                              |
| 输入参数{in}              |   |
| <b>ob_wp</b>          | 写保护配置数据。注意，如果您想保护相应的页面，可以将位设置为1。最低8位在除bank0以外的区域内有效。                              |
| 输出参数{out}             |   |
| -                     | -   |
| 返回值                   |   |
| <b>fmc_state_enum</b> | FMC状态   |
| <i>FMC_READY</i>      | 操作完成  |
| <i>FMC_BUSY</i>       | 操作进行中   |
| <i>FMC_PGSEERR</i>    | 编程序列错误  |

|                     |                              |
|---------------------|------------------------------|
| <i>FMC_PGERR</i>    | 编程错误                         |
| <i>FMC_PGAERR</i>   | 编程对齐错误                       |
| <i>FMC_WPERR</i>    | 擦写保护错误                       |
| <i>FMC_TOERR</i>    | 超时错误                         |
| <i>FMC_CBCMDERR</i> | 被选中区域不空白错误                   |
| <i>FMC_RSTERR</i>   | 在擦/写flash期间，BOR/POR或系统复位产生错误 |
| <i>FMC_OB_HSPC</i>  | 高安全保护级别                      |

例如：

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* enable option byte 0 write protection */
```

```
fmc_state_enum fmc_state = ob_write_protection_enable(BANK0_AREA, 0x00100000);
```

### 函数 ob\_security\_protection\_config

函数ob\_security\_protection\_config描述见下表：

表 3-328. 函数 ob\_security\_protection\_config

| 函数名称                  | ob_security_protection_config                                  |
|-----------------------|--|
| 函数原形                  | fmc_state_enum ob_security_protection_config(uint16_t ob_spc); |
| 功能描述                  | 配置安全保护   |
| 先决条件                  | ob_unlock  |
| 被调用函数                 | -  |
| 输入参数{in}              |  |
| <b>ob_spc</b>         | 安全保护   |
| <i>FMC_NSPC</i>       | 无安全保护  |
| <i>FMC_LSPC</i>       | 低保护级别  |
| <i>FMC_HSPC</i>       | 高保护级别  |
| 输出参数{out}             |  |
| -                     | -  |
| 返回值                   |  |
| <b>fmc_state_enum</b> | FMC状态  |
| <i>FMC_READY</i>      | 操作完成   |
| <i>FMC_BUSY</i>       | 操作进行中  |
| <i>FMC_PGSERR</i>     | 编程序列错误   |
| <i>FMC_PGERR</i>      | 编程错误   |
| <i>FMC_PGAERR</i>     | 编程对齐错误   |
| <i>FMC_WPERR</i>      | 擦写保护错误   |
| <i>FMC_TOERR</i>      | 超时错误   |
| <i>FMC_CBCMDERR</i>   | 被选中区域不空白错误   |
| <i>FMC_RSTERR</i>     | 在擦/写flash期间，BOR/POR或系统复位产生错误                                   |

例如:

```
fmc_state_enum fmc_state;

fmc_unlock();

ob_unlock();

/* enable security protection */

fmc_state = ob_security_protection_config(FMC_NSPPC);
```

### 函数 ob\_user\_write

函数ob\_user\_write描述见下表:

表 3-329. 函数 ob\_user\_write

| 函数名称                                   | ob_user_write                                  |
|--|--|
| 函数原形                                   | fmc_state_enum ob_user_write(uint8_t ob_user); |
| 功能描述                                   | 编程用户选项字节                                       |
| 先决条件                                   | ob_unlock                                      |
| 被调用函数                                  | -  |
| 输入参数{in}                               |  |
| <b>ob_user</b>                         | 用户选项字节   |
| OB_FWDGT_HW/OB_FWDGT_SW                | 独立看门狗定时器模式                                     |
| OB_DEEPSLEEP_RST/OB_DEEPSLEEP_N_RST    | 产生复位或进入深度睡眠模式                                  |
| OB_STDBY_RST/OB_STDBY_N_RST            | 产生复位或进入待机模式                                    |
| OB_BOOT_FROM_BANK0/OB_BOOT_FROM_BANK1  | boot模式   |
| OB_BOOT_OTA_ENABLE/OB_BOOT_OTA_DISABLE | OTA模式  |
| OB_BOR_DISABLE/OB_BOR_ENABLE           | BOR开/关   |
| 输出参数{out}                              |  |
| -                                      | -  |
| 返回值                                    |  |
| <b>fmc_state_enum</b>                  | FMC状态  |
| FMC_READY                              | 操作完成   |
| FMC_BUSY                               | 操作进行中  |
| FMC_PGSERR                             | 编程序列错误   |

|                     |                              |
|---------------------|------------------------------|
| <i>FMC_PGERR</i>    | 编程错误                         |
| <i>FMC_PGAERR</i>   | 编程对齐错误                       |
| <i>FMC_WPERR</i>    | 擦写保护错误                       |
| <i>FMC_TOERR</i>    | 超时错误                         |
| <i>FMC_CBCMDERR</i> | 被选中区域不空白错误                   |
| <i>FMC_RSTERR</i>   | 在擦/写flash期间，BOR/POR或系统复位产生错误 |
| <i>FMC_OB_HSPC</i>  | 高安全保护级别                      |

例如：

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* program the FMC user option byte */
```

```
fmc_state_enum fmc_state = ob_user_write(OB_FWDGT_HW | OB_BOOT_FROM_BANK1);
```

### 函数 ob\_data\_program

函数ob\_data\_program描述见下表：

表 3-330. 函数 ob\_data\_program

|                     |   |
|---------------------|---|
| 函数名称                | ob_data_program                                   |
| 函数原形                | fmc_state_enum ob_data_program(uint16_t ob_data); |
| 功能描述                | 编程数据选项字节  |
| 先决条件                | ob_unlock   |
| 被调用函数               | -   |
| 输入参数{in}            |   |
| ob_data             | 编程数据，OB_DATA[0:15]                                |
| 输出参数{out}           |   |
| -                   | -   |
| 返回值                 |   |
| fmc_state_enum      | FMC状态   |
| <i>FMC_READY</i>    | 操作完成  |
| <i>FMC_BUSY</i>     | 操作进行中   |
| <i>FMC_PGSERR</i>   | 编程序列错误  |
| <i>FMC_PGERR</i>    | 编程错误  |
| <i>FMC_PGAERR</i>   | 编程对齐错误  |
| <i>FMC_WPERR</i>    | 擦写保护错误  |
| <i>FMC_TOERR</i>    | 超时错误  |
| <i>FMC_CBCMDERR</i> | 被选中区域不空白错误  |
| <i>FMC_RSTERR</i>   | 在擦/写flash期间，BOR/POR或系统复位产生错误                      |
| <i>FMC_OB_HSPC</i>  | 高安全保护级别   |

例如：

```
fmc_unlock();

ob_unlock();

/* program option bytes data */

fmc_state_enum fmc_state = ob_data_program(0xdd22);
```

### 函数 ob\_user\_get

函数ob\_user\_get描述见下表：

表 3-331. 函数 ob\_user\_get

|           |                                  |
|-----------|----------------------------------|
| 函数名称      | ob_user_get                      |
| 函数原形      | uint8_t ob_user_get(void);       |
| 功能描述      | 获取FMC_OBSTAT寄存器中FMC选项字节OB_USER的值 |
| 先决条件      | -                                |
| 被调用函数     | -                                |
| 输入参数{in}  |                                  |
| -         | -                                |
| 输出参数{out} |                                  |
| -         | -                                |
| 返回值       |                                  |
| uint8_t   | 选项字节用户数值（0x00 – 0xFF）            |

例如：

```
/* get the value of FMC option byte OB_USER in FMC_OBSTAT register */

uint8_t user = ob_user_get();
```

### 函数 ob\_data\_get

函数ob\_data\_get描述见下表：

表 3-332. 函数 ob\_data\_get

|           |                                   |
|-----------|-----------------------------------|
| 函数名称      | ob_data_get                       |
| 函数原形      | uint16_t ob_data_get(void);       |
| 功能描述      | 获取FMC_OBSTAT寄存器中FMC选项字节 OB_DATA的值 |
| 先决条件      | -                                 |
| 被调用函数     | -                                 |
| 输入参数{in}  |                                   |
| -         | -                                 |
| 输出参数{out} |                                   |
| -         | -                                 |
| 返回值       |                                   |
| uint16_t  | 选项字节数据值（0x0 – 0xFFFF）             |

例如：

```
/* get the value of FMC option byte OB_DATA in FMC_OBSTAT register */
```

```
uint16_t data = ob_data_get();
```

### 函数 ob\_write\_protection\_get

函数ob\_write\_protection\_get描述见下表：

表 3-333. 函数 ob\_write\_protection\_get

|           |   |
|-----------|---|
| 函数名称      | ob_write_protection_get                 |
| 函数原形      | uint32_t ob_write_protection_get(void); |
| 功能描述      | 获取FMC_WP0寄存器中FMC选项字节BK0WP的值             |
| 先决条件      | -                                       |
| 被调用函数     | -                                       |
| 输入参数{in}  |   |
| -         | -                                       |
| 输出参数{out} |   |
| -         | -                                       |
| 返回值       |   |
| uint32_t  | 选项字节BK0WP数值（0x0 – 0xFFFF FFFF）          |

例如：

```
/* get the value of FMC option byte BK0WP in FMC_WP0 register */
```

```
uint32_t wp = ob_write_protection_get();
```

### 函数 ob\_bk1\_write\_protection\_get

函数ob\_bk1\_write\_protection\_get描述见下表：

表 3-334. 函数 ob\_bk1\_write\_protection\_get

|           |  |
|-----------|--|
| 函数名称      | ob_bk1_write_protection_get                |
| 函数原形      | uint8_t ob_bk1_write_protection_get(void); |
| 功能描述      | 获取FMC_WP1寄存器中FMC选项字节BK1WP的值                |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| uint8_t   | 选项字节BK1WP数值（0x0 – 0xFF）                    |

例如：

```
/* get the value of FMC option byte BK1WP in FMC_WP1 register */
```

```
uint8_t wp = ob_bk1_write_protection_get();
```

### 函数 ob\_df\_write\_protection\_get

函数ob\_df\_write\_protection\_get描述见下表：

**表 3-335. 函数 ob\_df\_write\_protection\_get**

|           |   |
|-----------|---|
| 函数名称      | ob_df_write_protection_get                |
| 函数原形      | uint8_t ob_df_write_protection_get(void); |
| 功能描述      | 获取FMC_WP1寄存器中FMC选项字节DFWP的值                |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| -         | -   |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| uint8_t   | 选项字节DFWP数值 (0x0 – 0xFF)                   |

例如：

```
/* get the value of FMC option byte DFWP in FMC_WP1 register */
```

```
uint8_t wp = ob_df_write_protection_get();
```

### 函数 ob\_plevel\_get

函数ob\_plevel\_get描述见下表：

**表 3-336. 函数 ob\_plevel\_get**

|                     |  |
|---------------------|--|
| 函数名称                | ob_plevel_get                            |
| 函数原形                | uint8_t ob_plevel_get(void);             |
| 功能描述                | 获取FMC_OBSTAT寄存器中FMC选项字节0安全保护级别(PLEVEL)的值 |
| 先决条件                | -  |
| 被调用函数               | -  |
| 输入参数{in}            |  |
| -                   | -  |
| 输出参数{out}           |  |
| -                   | -  |
| 返回值                 |  |
| uint8_t             | PLEVEL值                                  |
| OB_OBSTAT_PLEVEL_NO | 无安全保护                                    |
| OB_OBSTAT_PLEVEL    | 低级别安全保护                                  |

|   |         |
|---|---------|
| <code>_LOW</code>                                   |         |
| <code>OB_OBSTAT_PLEVEL</code><br><code>_HIGH</code> | 高级别安全保护 |

例如:

```
/* get the FMC option byte security protection level */
```

```
uint8_t spc = ob_plevel_get();
```

### 函数 `ob1_lock_config`

函数 `ob1_lock_config` 描述见下表:

**表 3-337. 函数 `ob1_lock_config`**

|                               |   |
|-------------------------------|---|
| 函数名称                          | <code>ob1_lock_config</code>                                    |
| 函数原形                          | <code>fmc_state_enum ob1_lock_config(uint32_t lk_value);</code> |
| 功能描述                          | 配置选项字节1锁定值  |
| 先决条件                          | -   |
| 被调用函数                         | -   |
| 输入参数{in}                      |   |
| <code>lk_value</code>         | 编程的LK值  |
| <code>OB1CS_OB1_LK</code>     | any more 当配置为 <code>OB1CS_OB1_LK</code> 时, 选项字节1不能再被修改          |
| <code>OB1CS_OB1_NOT_LK</code> | 选项字节1未被锁定   |
| 输出参数{out}                     |   |
| -                             | -   |
| 返回值                           |   |
| <code>fmc_state_enum</code>   | FMC状态   |
| <code>FMC_READY</code>        | 操作完成  |
| <code>FMC_BUSY</code>         | 操作进行中   |
| <code>FMC_PGSERR</code>       | 编程序列错误  |
| <code>FMC_PGERR</code>        | 编程错误  |
| <code>FMC_PGAERR</code>       | 编程对齐错误  |
| <code>FMC_WPERR</code>        | 擦写保护错误  |
| <code>FMC_TOERR</code>        | 超时错误  |
| <code>FMC_CBCMDERR</code>     | 被选中区域不空白错误  |
| <code>FMC_RSTERR</code>       | 在擦/写flash期间, BOR/POR或系统复位产生错误                                   |
| <code>FMC_OB1_LK</code>       | 选项字节1锁定   |

例如:

```
/* configure lock value in option byte 1 */
```

```
fmc_state_enum fmc_state = ob1_lock_config(OB1CS_OB1_NOT_LK);
```



函数 **ob1\_parameter\_config**

函数ob1\_parameter\_config描述见下表:

表 3-338. 函数 **ob1\_parameter\_config**

|                       |   |
|-----------------------|---|
| 函数名称                  | ob1_parameter_config  |
| 函数原形                  | fmc_state_enum ob1_parameter_config(uint32_t dflash_size);; |
| 功能描述                  | 配置选项字节1参数   |
| 先决条件                  | -   |
| 被调用函数                 | -   |
| 输入参数{in}              |   |
| <b>dflash_size</b>    | 编程的数据闪存大小   |
| <i>OB1CS_DF_64K</i>   | 数据闪存大小为64KB   |
| <i>OB1CS_DF_32K</i>   | 数据闪存大小为32KB   |
| 输出参数{out}             |   |
| -                     | -   |
| 返回值                   |   |
| <b>fmc_state_enum</b> | FMC状态   |
| <i>FMC_READY</i>      | 操作完成  |
| <i>FMC_BUSY</i>       | 操作进行中   |
| <i>FMC_PGSERR</i>     | 编程序列错误  |
| <i>FMC_PGERR</i>      | 编程错误  |
| <i>FMC_PGAERR</i>     | 编程对齐错误  |
| <i>FMC_WPERR</i>      | 擦写保护错误  |
| <i>FMC_TOERR</i>      | 超时错误  |
| <i>FMC_CBCMDERR</i>   | 被选中区域不空白错误  |
| <i>FMC_RSTERR</i>     | 在擦/写flash期间，BOR/POR或系统复位产生错误                                |
| <i>FMC_OB1_LK</i>     | 选项字节1锁定   |

例如:

```
/* configure option byte 1 parameters */
```

```
fmc_state_enum fmc_state = ob1_parameter_config(OB1CS_DF_64K);
```

函数 **dflash\_size\_get**

函数dflash\_size\_get描述见下表:

表 3-339. 函数 **dflash\_size\_get**

|          |                                 |
|----------|---------------------------------|
| 函数名称     | dflash_size_get                 |
| 函数原形     | uint32_t dflash_size_get(void); |
| 功能描述     | 获取以字节为单位的数据闪存大小                 |
| 先决条件     | -                               |
| 被调用函数    | -                               |
| 输入参数{in} |                                 |

|           |                              |
|-----------|------------------------------|
| -         | -                            |
| 输出参数{out} |                              |
| -         | -                            |
| 返回值       |                              |
| uint32_t  | 数据闪存字节数 (0x00 – 0xFFFF FFFF) |

例如:

```
/* get data flash size in byte unit */
```

```
uint32_t size = dflash_size_get();
```

### 函数 fmc\_flag\_get

函数fmc\_flag\_get描述见下表:

表 3-340. 函数 fmc\_flag\_get

|            |  |
|------------|--|
| 函数名称       | fmc_flag_get   |
| 函数原形       | FlagStatus fmc_flag_get(fmc_flag_enum flag);         |
| 功能描述       | 获取FMC标志状态  |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| flag       | FMC标志, 参考 <a href="#">表3-294. 枚举类型fmc_flag_enum.</a> |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| FlagStatus | SET或RESET  |

例如:

```
/* get FMC end flag */
```

```
FlagStatus flag = fmc_flag_get(FMC_BANK0_FLAG_END);
```

### 函数 fmc\_flag\_clear

函数fmc\_flag\_clear描述见下表:

表 3-341. 函数 fmc\_flag\_clear

|          |  |
|----------|--|
| 函数名称     | fmc_flag_clear                           |
| 函数原形     | void fmc_flag_clear(fmc_flag_enum flag); |
| 功能描述     | 清除FMC标志                                  |
| 先决条件     | -  |
| 被调用函数    | -  |
| 输入参数{in} |  |
| flag     | FMC标志                                    |

|                                |                                 |
|--------------------------------|---------------------------------|
| <i>FMC_BANK0_FLAG_PGSERR</i>   | 闪存bank0编程序列错误标志                 |
| <i>FMC_BANK0_FLAG_PGERR</i>    | 闪存bank0编程错误标志                   |
| <i>FMC_BANK0_FLAG_PGAERR</i>   | 闪存bank0编程对齐错误标志                 |
| <i>FMC_BANK0_FLAG_WPERR</i>    | 闪存bank0擦写保护错误标志                 |
| <i>FMC_BANK0_FLAG_END</i>      | 闪存bank0操作结束标志                   |
| <i>FMC_BANK0_FLAG_BCMCDERR</i> | 闪存bank0被查空命令选中的区域都是0xFF或不是标志    |
| <i>FMC_BANK0_FLAG_RSTERR</i>   | 闪存bank0 BOR/POR或系统复位期间擦除/程序标志   |
| <i>FMC_BANK1_FLAG_PGSERR</i>   | 闪存bank1编程序列错误标志                 |
| <i>FMC_BANK1_FLAG_PGERR</i>    | 闪存bank1编程错误标志                   |
| <i>FMC_BANK1_FLAG_PGAERR</i>   | 闪存bank1编程对齐错误标志                 |
| <i>FMC_BANK1_FLAG_WPERR</i>    | 闪存bank1擦写保护错误标志                 |
| <i>FMC_BANK1_FLAG_END</i>      | 闪存bank1操作结束标志                   |
| <i>FMC_BANK1_FLAG_BCMCDERR</i> | 闪存bank1被查空命令选中的区域都是0xFF或不是标志    |
| <i>FMC_BANK1_FLAG_RSTERR</i>   | 闪存bank1 BOR/POR或系统复位期间擦除/程序标志   |
| <i>FMC_FLAG_OB1ECCDET</i>      | 选项字节1检测到两个位ECC错误标志              |
| <i>FMC_FLAG_OB0ECCDET</i>      | 选项字节0检测到两个位ECC错误标志              |
| <i>FMC_FLAG_ECCCOR</i>         | 一个位错误检测和纠正标志                    |
| <i>FMC_FLAG_ECCDET</i>         | OTP/数据闪存/system区/bank1检测到两位错误标志 |
| 输出参数{out}                      |                                 |
| -                              | -                               |
| 返回值                            |                                 |
| -                              | -                               |

例如:

```
/* clear FMC end flag */
```

```
fmc_flag_clear(FMC_BANK0_FLAG_END);
```

## 函数 fmc\_interrupt\_enable

函数fmc\_interrupt\_enable描述见下表:

表 3-342. 函数 fmc\_interrupt\_enable

|           |   |
|-----------|---|
| 函数名称      | fmc_interrupt_enable                                      |
| 函数原形      | void fmc_interrupt_enable(fmc_interrupt_enum interrupt);  |
| 功能描述      | 使能FMC中断   |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| interrupt | FMC中断, 参考 <a href="#">表3-296. 枚举类型fmc_interrupt_enum.</a> |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如:

```
/* enable FMC end interrupt */  
fmc_interrupt_enable(FMC_BANK0_INT_END);
```

## 函数 fmc\_interrupt\_disable

函数fmc\_interrupt\_disable描述见下表:

表 3-343. 函数 fmc\_interrupt\_disable

|           |   |
|-----------|---|
| 函数名称      | fmc_interrupt_disable                                     |
| 函数原形      | void fmc_interrupt_disable(fmc_interrupt_enum interrupt); |
| 功能描述      | 禁能FMC中断   |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| interrupt | FMC中断, 参考 <a href="#">表3-296. 枚举类型fmc_interrupt_enum.</a> |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如:

```
/* disable FMC end interrupt */  
fmc_interrupt_disable(FMC_BANK0_INT_END);
```

### 函数 fmc\_interrupt\_flag\_get

函数fmc\_interrupt\_flag\_get描述见下表:

表 3-344. 函数 fmc\_interrupt\_flag\_get

|            |  |
|------------|--|
| 函数名称       | fmc_interrupt_flag_get   |
| 函数原形       | FlagStatus fmc_interrupt_flag_get(fmc_interrupt_flag_enum int_flag); |
| 功能描述       | 获取FMC中断标志状态  |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| int_flag   | FMC中断标志, 参考 <a href="#">表3-295. 枚举类型fmc_interrupt_flag_enum.</a>     |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| FlagStatus | SET或RESET  |

例如:

```
/* check FMC program operation error flag is set or not */
```

```
FlagStatus flag = fmc_interrupt_flag_get(FMC_BANK0_INT_FLAG_PGERR);
```

### 函数 fmc\_interrupt\_flag\_clear

函数fmc\_interrupt\_flag\_clear描述见下表:

表 3-345. 函数 fmc\_interrupt\_flag\_clear

|           |  |
|-----------|--|
| 函数名称      | fmc_interrupt_flag_clear   |
| 函数原形      | void fmc_interrupt_flag_clear(fmc_interrupt_flag_enum int_flag); |
| 功能描述      | 清除FMC中断标志  |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| int_flag  | FMC中断标志, 参考 <a href="#">表3-295. 枚举类型fmc_interrupt_flag_enum.</a> |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如:

```
/* clear FMC program operation error flag */
```

```
fmc_interrupt_flag_get(FMC_BANK0_INT_FLAG_PGERR);
```

## 3.12. FWDGT

独立看门狗定时器（FWDGT）是一个硬件计时电路，用来监测由软件故障导致的系统故障。适合于需要独立环境且对计时精度要求不高的场合。章节[3.12.1](#)描述了FWDGT的寄存器列表，章节[3.12.2](#)对FWDGT库函数进行说明。

### 3.12.1. 外设寄存器说明

FWDGT寄存器列表如下表所示：

表 3-346. FWDGT 寄存器

| 寄存器名称      | 寄存器描述  |
|------------|--------|
| FWDGT_CTL  | 控制寄存器  |
| FWDGT_PSC  | 预分频寄存器 |
| FWDGT_RLD  | 重装载寄存器 |
| FWDGT_STAT | 状态寄存器  |
| FWDGT_WND  | 窗口寄存器  |

### 3.12.2. 外设库函数说明

FWDGT库函数列表如下表所示：

表 3-347. FWDGT 库函数

| 库函数名称                        | 库函数描述                                    |
|------------------------------|--|
| fwdgt_write_enable           | 使能对寄存器FWDGT_PSC, FWDGT_RLD和FWDGT_WND的写操作 |
| fwdgt_write_disable          | 失能对寄存器FWDGT_PSC, FWDGT_RLD和FWDGT_WND的写操作 |
| fwdgt_enable                 | 使能FWDGT                                  |
| fwdgt_prescaler_value_config | 配置独立看门狗定时器时钟预分频数                         |
| fwdgt_reload_value_config    | 配置独立看门狗定时器计数器重装载值                        |
| fwdgt_window_value_config    | 配置独立看门狗定时器计数窗口值                          |
| fwdgt_counter_reload         | 按照FWDGT_RLD寄存器的值重装载FWDGT计数器              |
| fwdgt_config                 | 设置FWDGT重装载值、预分频值                         |
| fwdgt_flag_get               | 获取FWDGT标志位状态                             |

#### 函数 fwdgt\_write\_enable

函数fwdgt\_write\_enable描述见下表：

表 3-348. 函数 fwdgt\_write\_enable

|      |  |
|------|--|
| 函数名称 | fwdgt_write_enable                       |
| 函数原型 | void fwdgt_write_enable(void);           |
| 功能描述 | 使能对寄存器FWDGT_PSC, FWDGT_RLD和FWDGT_WND的写操作 |

|           |   |
|-----------|---|
| 先决条件      | - |
| 被调用函数     | - |
| 输入参数{in}  |   |
| -         | - |
| 输出参数{out} |   |
| -         | - |
| 返回值       |   |
| -         | - |

例如：

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

```
fwdgt_write_enable();
```

### 函数 fwdgt\_write\_disable

函数fwdgt\_write\_disable描述见下表：

**表 3-349. 函数 fwdgt\_write\_disable**

|           |  |
|-----------|--|
| 函数名称      | fwdgt_write_disable                      |
| 函数原型      | void fwdgt_write_disable(void);          |
| 功能描述      | 除能对寄存器FWDGT_PSC, FWDGT_RLD和FWDGT_WND的写操作 |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如：

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

```
fwdgt_write_disable();
```

### 函数 fwdgt\_enable

函数fwdgt\_enable描述见下表：

**表 3-350. 函数 fwdgt\_enable**

|      |                          |
|------|--------------------------|
| 函数名称 | fwdgt_enable             |
| 函数原型 | void fwdgt_enable(void); |
| 功能描述 | 使能FWDGT                  |
| 先决条件 | -                        |

|           |   |
|-----------|---|
| 被调用函数     | - |
| 输入参数{in}  |   |
| -         | - |
| 输出参数{out} |   |
| -         | - |
| 返回值       |   |
| -         | - |

例如：

```
/* start the FWDGT counter */
```

```
fwdgt_enable();
```

### 函数 fwdgt\_prescaler\_value\_config

函数fwdgt\_prescaler\_value\_config描述见下表：

表 3-351. 函数 fwdgt\_prescaler\_value\_config

|                 |   |
|-----------------|---|
| 函数名称            | fwdgt_prescaler_value_config                                      |
| 函数原型            | ErrStatus fwdgt_prescaler_value_config(uint16_t prescaler_value); |
| 功能描述            | 配置独立看门狗定时器时钟预分频数  |
| 先决条件            | -   |
| 被调用函数           | -   |
| 输入参数{in}        |   |
| prescaler_value | 预分频值  |
| FWDGT_PSC_DIVx  | FWDGT预分频值设为x（x=4,8,16,32,64,128,256）                              |
| 输出参数{out}       |   |
| -               | -   |
| 返回值             |   |
| ErrStatus       | ERROR / SUCCESS   |

例如：

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

```
ErrStatus flag;
```

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

### 函数 fwdgt\_reload\_value\_config

函数fwdgt\_reload\_value\_config描述见下表：

表 3-352. 函数 fwdgt\_reload\_value\_config

|      |   |
|------|---|
| 函数名称 | fwdgt_reload_value_config                                   |
| 函数原型 | ErrStatus fwdgt_reload_value_config(uint16_t reload_value); |
| 功能描述 | 配置独立看门狗定时器计数器重装载值   |



|              |                           |
|--------------|---------------------------|
| 先决条件         | -                         |
| 被调用函数        | -                         |
| 输入参数{in}     |                           |
| reload_value | 重载值, 数值范围为0x0000 - 0x0FFF |
| 输出参数{out}    |                           |
| -            | -                         |
| 返回值          |                           |
| ErrStatus    | ERROR / SUCCESS           |

例如:

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

```
ErrStatus flag;
```

```
flag = fwdgt_reloadr_value_config(0xFFFF);
```

### 函数 fwdgt\_window\_value\_reload

函数fwdgt\_window\_value\_config描述见下表:

表 3-353. 函数 fwdgt\_window\_value\_config

|              |   |
|--------------|---|
| 函数名称         | fwdgt_window_value_config                                   |
| 函数原型         | ErrStatus fwdgt_window_value_config(uint16_t window_value); |
| 功能描述         | 配置独立看门狗定时器计数器窗口值  |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| window_value | 窗口值,数值范围为0x0000 – 0x0FFF                                    |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| ErrStatus    | ERROR / SUCCESS   |

例如:

```
/* set FWDGT window value to 0xFFFF */
```

```
ErrStatus flag;
```

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

### 函数 fwdgt\_counter\_reload

函数fwdgt\_counter\_reload描述见下表:

表 3-354. 函数 fwdgt\_counter\_reload

|      |                      |
|------|----------------------|
| 函数名称 | fwdgt_counter_reload |
|------|----------------------|

|           |                                  |
|-----------|----------------------------------|
| 函数原型      | void fwdgt_counter_reload(void); |
| 功能描述      | 按照FWDGT_RLD寄存器的值重装载FWDGT计数器      |
| 先决条件      | -                                |
| 被调用函数     | -                                |
| 输入参数{in}  |                                  |
| -         | -                                |
| 输出参数{out} |                                  |
| -         | -                                |
| 返回值       |                                  |
| -         | -                                |

例如：

```
/* reload FWDGT counter */
```

```
fwdgt_counter_reload();
```

### 函数 fwdgt\_config

函数fwdgt\_config描述见下表：

表 3-355. 函数 fwdgt\_config

|                  |   |
|------------------|---|
| 函数名称             | fwdgt_config  |
| 函数原型             | ErrStatus fwdgt_config(uint16_t reload_value, uint8_t prescaler_div); |
| 功能描述             | 设置FWDGT重装载值、预分频值  |
| 先决条件             | -   |
| 被调用函数            | -   |
| 输入参数{in}         |   |
| reload_value     | 重装载值(0x0000 - 0x0FFF)   |
| 输入参数{in}         |   |
| prescaler_div    | FWDGT预分频值   |
| FWDGT_PSC_DIV4   | FWDGT预分频值设为4  |
| FWDGT_PSC_DIV8   | FWDGT预分频值设为8  |
| FWDGT_PSC_DIV16  | FWDGT预分频值设为16   |
| FWDGT_PSC_DIV32  | FWDGT预分频值设为32   |
| FWDGT_PSC_DIV64  | FWDGT预分频值设为64   |
| FWDGT_PSC_DIV128 | FWDGT预分频值设为128  |
| FWDGT_PSC_DIV256 | FWDGT预分频值设为256  |
| 输出参数{out}        |   |
| -                | -   |

| 返回值       |                   |
|-----------|-------------------|
| ErrStatus | ERROR or SUCCESS- |

例如:

```
/* configure FWDGT counter clock: 40KHz(IRC40K) / 64 = 0.625 KHz */
```

```
fwdgt_config(2*500, FWDGT_PSC_DIV64);
```

### 函数 fwdgt\_flag\_get

函数fwdgt\_flag\_get描述见下表:

表 3-356. 函数 fwdgt\_flag\_get

|                |   |
|----------------|---|
| 函数名称           | fwdgt_flag_get                            |
| 函数原型           | FlagStatus fwdgt_flag_get(uint16_t flag); |
| 功能描述           | 获取FWDGT标志位状态                              |
| 先决条件           | -   |
| 被调用函数          | -   |
| 输入参数{in}       |   |
| flag           | 需要获取状态的FWDGT标志位                           |
| FWDGT_FLAG_PUD | 预分频值更新进行中                                 |
| FWDGT_FLAG_RUD | 重装载值更新进行中                                 |
| FWDGT_FLAG_WUD | 窗口值更新进行中                                  |
| 输出参数{out}      |   |
| -              | -   |
| 返回值            |   |
| FlagStatus     | SET / RESET                               |

例如:

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

```
FlagStatus status;
```

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

## 3.13. GPIO

GPIO用来实现各片上设备的逻辑输入/输出功能。章节[3.13.1](#)描述了GPIO的寄存器列表，章节[3.13.2](#)对GPIO库函数进行说明。

### 3.13.1. 外设寄存器说明

GPIO寄存器列表如下表所示:

表 3-357. GPIO 寄存器

| 寄存器名称        | 寄存器描述      |
|--------------|------------|
| GPIOx_CTL    | 端口控制寄存器    |
| GPIOx_OMODE  | 端口输出模式寄存器  |
| GPIOx_OSPD   | 端口输出速度寄存器  |
| GPIOx_PUD    | 端口上拉/下拉寄存器 |
| GPIOx_ISTAT  | 端口输入状态寄存器  |
| GPIOx_OCTL   | 端口输出控制寄存器  |
| GPIOx_BOP    | 端口位操作寄存器   |
| GPIOx_LOCK   | 端口配置锁定寄存器  |
| GPIOx_AFSEL0 | 备用功能选择寄存器0 |
| GPIOx_AFSEL1 | 备用功能选择寄存器1 |
| GPIOx_BC     | 位清除寄存器     |
| GPIOx_TG     | 端口位翻转寄存器   |

### 3.13.2. 外设库函数说明

GPIO库函数列表如下表所示：

表 3-358. GPIO 库函数

| 库函数名称                   | 库函数描述         |
|-------------------------|---------------|
| gpio_deinit             | 复位外设GPIOx     |
| gpio_mode_set           | 设置GPIO模式      |
| gpio_output_options_set | 设置GPIO输出模式和速度 |
| gpio_bit_set            | 置位引脚值         |
| gpio_bit_reset          | 复位引脚值         |
| gpio_bit_write          | 将特定的值写入引脚     |
| gpio_port_write         | 将特定的值写入一组端口   |
| gpio_input_bit_get      | 获取引脚的输入值      |
| gpio_input_port_get     | 获取一组端口的输入值    |
| gpio_output_bit_get     | 获取引脚的输出值      |
| gpio_output_port_get    | 获取一组端口的输出值    |
| gpio_af_set             | 设置GPIO复用功能    |
| gpio_pin_lock           | 相应的引脚配置被锁定    |
| gpio_bit_toggle         | 翻转GPIO引脚状态    |
| gpio_port_toggle        | 翻转一组GPIO状态    |

#### 函数 gpio\_deinit

函数gpio\_deinit描述见下表：

表 3-359. 函数 gpio\_deinit

| 函数名称 | gpio_deinit                             |
|------|---|
| 函数原型 | void gpio_deinit(uint32_t gpio_periph); |

|             |  |
|-------------|--|
| 功能描述        | 复位外设GPIOx  |
| 先决条件        | -  |
| 被调用函数       | rcu_periph_reset_enable / rcu_periph_reset_disable |
| 输入参数{in}    |  |
| gpio_periph | GPIO端口   |
| GPIOx       | 端口选择(x = A,B,C,D,E,F)                              |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如:

```
/* reset GPIOA */
```

```
gpio_deinit(GPIOA);
```

### 函数 gpio\_mode\_set

函数gpio\_mode\_set描述见下表:

表 3-360. 函数 gpio\_mode\_set

|                  |   |
|------------------|---|
| 函数名称             | gpio_mode_set   |
| 函数原型             | void gpio_mode_set(uint32_t gpio_periph, uint32_t mode, uint32_t pull_up_down, uint32_t pin); |
| 功能描述             | 设置GPIO模式  |
| 先决条件             | -   |
| 被调用函数            | -   |
| 输入参数{in}         |   |
| gpio_periph      | GPIO端口  |
| GPIOx            | GPIOx(x = A,B,C,D,E,F)  |
| 输入参数{in}         |   |
| mode             | GPIO引脚模式  |
| GPIO_MODE_INPUT  | 输入模式  |
| GPIO_MODE_OUTPUT | 输出模式  |
| GPIO_MODE_AF     | 备用功能模式  |
| GPIO_MODE_ANALOG | 模拟模式  |
| 输入参数{in}         |   |
| pull_up_down     | GPIO引脚上拉下拉电阻设置  |
| GPIO_PUPD_NONE   | 悬空模式, 无上拉和下拉  |
| GPIO_PUPD_PULLUP | 带上拉电阻   |

|                    |               |
|--------------------|---------------|
| UP                 |               |
| GPIO_PUPD_PULLDOWN | 带下拉电阻         |
| 输入参数{in}           |               |
| pin                | GPIO pin      |
| GPIO_PIN_x         | 引脚选择(x=0..15) |
| GPIO_PIN_ALL       | 所有引脚          |
| 输出参数{out}          |               |
| -                  | -             |
| 返回值                |               |
| -                  | -             |

例如:

```
/* config PA0 as input mode with pullup */
```

```
gpio_mode_set(GPIOA, GPIO_MODE_INPUT, GPIO_PUPD_PULLUP, GPIO_PIN_0);
```

### 函数 gpio\_output\_options\_set

函数gpio\_output\_options\_set描述见下表:

表 3-361. 函数 gpio\_output\_options\_set

|                   |  |
|-------------------|--|
| 函数名称              | gpio_output_options_set  |
| 函数原型              | void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t pin); |
| 功能描述              | 设置GPIO输出模式和速度  |
| 先决条件              | -  |
| 被调用函数             | -  |
| 输入参数{in}          |  |
| gpio_periph       | GPIO端口   |
| GPIOx             | 端口选择(x = A,B,C,D,E,F)  |
| 输入参数{in}          |  |
| otype             | GPIO引脚输出模式   |
| GPIO_OTYPE_PP     | 推挽输出模式   |
| GPIO_OTYPE_OD     | 开漏输出模式   |
| 输入参数{in}          |  |
| speed             | GPIO引脚输出最大速度   |
| GPIO_OSPEED_2MHZ  | 最大输出速度为2MHz  |
| GPIO_OSPEED_10MHZ | 最大输出速度为10MHz   |
| GPIO_OSPEED_50MHZ | 最大输出速度为50MHz   |
| 输入参数{in}          |  |

|                     |               |
|---------------------|---------------|
| <b>pin</b>          | GPIO引脚        |
| <i>GPIO_PIN_x</i>   | 引脚选择(x=0..15) |
| <i>GPIO_PIN_ALL</i> | 所有引脚          |
| <b>输出参数{out}</b>    |               |
| -                   | -             |
| <b>返回值</b>          |               |
| -                   | -             |

例如:

```
/* 配置PA0工作于推挽输出模式 */
```

```
gpio_output_options_set(GPIOA, GPIO_OTYPE_PP, GPIO_OSPEED_2MHZ,
GPIO_PIN_0);
```

### 函数 gpio\_bit\_set

函数gpio\_bit\_set描述见下表:

表 3-362. 函数 gpio\_bit\_set

|                     |  |
|---------------------|--|
| <b>函数名称</b>         | gpio_bit_set   |
| <b>函数原型</b>         | void gpio_bit_set(uint32_t gpio_periph, uint32_t pin); |
| <b>功能描述</b>         | 置位引脚值  |
| <b>先决条件</b>         | -  |
| <b>被调用函数</b>        | -  |
| <b>输入参数{in}</b>     |  |
| <b>gpio_periph</b>  | GPIO端口   |
| <i>GPIOx</i>        | 端口选择(x = A,B,C,D,E,F)                                  |
| <b>输入参数{in}</b>     |  |
| <b>pin</b>          | GPIO引脚   |
| <i>GPIO_PIN_x</i>   | 引脚选择(x=0..15)  |
| <i>GPIO_PIN_ALL</i> | 所有引脚   |
| <b>输出参数{out}</b>    |  |
| -                   | -  |
| <b>返回值</b>          |  |
| -                   | -  |

例如:

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

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

### 函数 gpio\_bit\_reset

函数gpio\_bit\_reset描述见下表:

表 3-363. 函数 `gpio_bit_reset`

|                           |   |
|---------------------------|---|
| 函数名称                      | <code>gpio_bit_reset</code>   |
| 函数原型                      | <code>void gpio_bit_reset(uint32_t gpio_periph, uint32_t pin);</code> |
| 功能描述                      | 复位引脚值   |
| 先决条件                      | -   |
| 被调用函数                     | -   |
| 输入参数{in}                  |   |
| <code>gpio_periph</code>  | GPIO端口  |
| <code>GPIOx</code>        | 端口选择(x = A,B,C,D,E,F)   |
| 输入参数{in}                  |   |
| <code>pin</code>          | GPIO引脚  |
| <code>GPIO_PIN_x</code>   | 引脚选择(x=0..15)   |
| <code>GPIO_PIN_ALL</code> | 所有引脚  |
| 输出参数{out}                 |   |
| -                         | -   |
| 返回值                       |   |
| -                         | -   |

例如:

```
/* reset PA0*/
```

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

### 函数 `gpio_bit_write`

函数`gpio_bit_write`描述见下表:

表 3-364. 函数 `gpio_bit_write`

|                           |   |
|---------------------------|---|
| 函数名称                      | <code>gpio_bit_write</code>   |
| 函数原型                      | <code>void gpio_bit_write(uint32_t gpio_periph, uint32_t pin, bit_status bit_value);</code> |
| 功能描述                      | 将特定的值写入引脚   |
| 先决条件                      | -   |
| 被调用函数                     | -   |
| 输入参数{in}                  |   |
| <code>gpio_periph</code>  | GPIO端口  |
| <code>GPIOx</code>        | 端口选择(x = A,B,C,D,E,F)   |
| 输入参数{in}                  |   |
| <code>pin</code>          | GPIO引脚  |
| <code>GPIO_PIN_x</code>   | 引脚选择(x=0..15)   |
| <code>GPIO_PIN_ALL</code> | 所有引脚  |
| 输入参数{in}                  |   |
| <code>bit_value</code>    | 设置或清除   |
| <code>RESET</code>        | 清除引脚值   |
| <code>SET</code>          | 设置引脚值   |



| 输出参数{out} |   |
|-----------|---|
| -         | - |
| 返回值       |   |
| -         | - |

例如:

```
/* write 1 to PA0*/
```

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

### 函数 gpio\_port\_write

函数gpio\_port\_write描述见下表:

表 3-365. 函数 gpio\_port\_write

| 函数名称        | gpio_port_write  |
|-------------|--|
| 函数原型        | void gpio_port_write(uint32_t gpio_periph, uint16_t data); |
| 功能描述        | 将特定的值写入端口  |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| gpio_periph | GPIO端口   |
| GPIOx       | 端口选择(x = A,B,C,D,E,F)                                      |
| 输入参数{in}    |  |
| data        | 将要写入的具体值   |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如:

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

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

### 函数 gpio\_input\_bit\_get

函数gpio\_input\_bit\_get描述见下表:

表 3-366. 函数 gpio\_input\_bit\_get

| 函数名称  | gpio_input_bit_get   |
|-------|--|
| 函数原型  | FlagStatus gpio_input_bit_get(uint32_t gpio_periph, uint32_t pin); |
| 功能描述  | 获取引脚的输入值   |
| 先决条件  | -  |
| 被调用函数 | -  |

| 输入参数{in}            |                       |
|---------------------|-----------------------|
| <b>gpio_periph</b>  | GPIO端口                |
| <i>GPIOx</i>        | 端口选择(x = A,B,C,D,E,F) |
| 输入参数{in}            |                       |
| <b>pin</b>          | GPIO引脚                |
| <i>GPIO_PIN_x</i>   | 引脚选择(x=0..15)         |
| <i>GPIO_PIN_ALL</i> | 所有引脚                  |
| 输出参数{out}           |                       |
| -                   | -                     |
| 返回值                 |                       |
| <b>FlagStatus</b>   | SET / RESET           |

例如:

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

```
FlagStatus bit_state;
```

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

### 函数 gpio\_input\_port\_get

函数gpio\_input\_port\_get描述见下表:

表 3-367. 函数 gpio\_input\_port\_get

| 函数名称               | gpio_input_port_get                                 |
|--------------------|---|
| 函数原型               | uint16_t gpio_input_port_get(uint32_t gpio_periph); |
| 功能描述               | 获取端口的输入值  |
| 先决条件               | -   |
| 被调用函数              | -   |
| 输入参数{in}           |   |
| <b>gpio_periph</b> | GPIO端口  |
| <i>GPIOx</i>       | 端口选择(x = A,B,C,D,E,F)                               |
| 输出参数{out}          |   |
| -                  | -   |
| 返回值                |   |
| <b>uint16_t</b>    | 0x0000-0xFFFF                                       |

例如:

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

```
uint16_t port_state;
```

```
port_state = gpio_input_port_get(GPIOA);
```

## 函数 gpio\_output\_bit\_get

函数gpio\_output\_bit\_get描述见下表:

表 3-368. 函数 gpio\_output\_bit\_get

|              |   |
|--------------|---|
| 函数名称         | gpio_output_bit_get   |
| 函数原型         | FlagStatus gpio_output_bit_get(uint32_t gpio_periph, uint32_t pin); |
| 功能描述         | 获取端口所有引脚的输出值  |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| gpio_periph  | GPIO端口  |
| GPIOx        | 端口选择(x = A,B,C,D,E,F)   |
| 输入参数{in}     |   |
| pin          | GPIO引脚  |
| GPIO_PIN_x   | 引脚选择(x=0..15)   |
| GPIO_PIN_ALL | 所有引脚  |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| FlagStatus   | SET / RESET   |

例如:

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

```
FlagStatus bit_state;
```

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

## 函数 gpio\_output\_port\_get

函数gpio\_output\_port\_get描述见下表:

表 3-369. 函数 gpio\_output\_port\_get

|             |  |
|-------------|--|
| 函数名称        | gpio_output_port_get                                 |
| 函数原型        | uint16_t gpio_output_port_get(uint32_t gpio_periph); |
| 功能描述        | 获取引脚的输出值   |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| gpio_periph | GPIO端口   |
| GPIOx       | 端口选择(x = A,B,C,D,E,F)                                |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |

|                 |               |
|-----------------|---------------|
| <b>uint16_t</b> | 0x0000-0xFFFF |
|-----------------|---------------|

例如:

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

```
uint16_t port_state;
```

```
port_state = gpio_output_port_get(GPIOA);
```

## 函数 gpio\_af\_set

函数gpio\_af\_set描述见下表:

表 3-370. 函数 gpio\_af\_set

|                     |  |
|---------------------|--|
| <b>函数名称</b>         | gpio_af_set  |
| <b>函数原型</b>         | void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin); |
| <b>功能描述</b>         | 设置GPIO的备用功能  |
| <b>先决条件</b>         | -  |
| <b>被调用函数</b>        | -  |
| <b>输入参数{in}</b>     |  |
| <b>gpio_periph</b>  | GPIO端口   |
| <i>GPIOx</i>        | GPIOx(x = A,B,C,D,E,F)   |
| <b>输入参数{in}</b>     |  |
| <b>alt_func_num</b> | GPIO 引脚备用功能, 请参见特定设备的数据手册  |
| <i>GPIO_AF_0</i>    | SYSTEM   |
| <i>GPIO_AF_1</i>    | TIMER0, TIMER1, TIMER7, TIMER19, TIMER20                                     |
| <i>GPIO_AF_2</i>    | TIMER0, TIMER1, TIMER7, TIMER19, TIMER20                                     |
| <i>GPIO_AF_3</i>    | TIMER7, TIMER19, I2C0  |
| <i>GPIO_AF_4</i>    | SPI0, SPI1, I2S1, USART1   |
| <i>GPIO_AF_5</i>    | USART0, USART2, MFCOM, SPI1, I2C1  |
| <i>GPIO_AF_6</i>    | CAN0, CAN1, MFCOM, TRIGSEL   |
| <i>GPIO_AF_7</i>    | TRIGSEL, CMP, MFCOM  |
| <i>GPIO_AF_8</i>    | -  |
| <i>GPIO_AF_9</i>    | EVENTOUT   |
| <b>输入参数{in}</b>     |  |
| <b>pin</b>          | GPIO引脚   |
| <i>GPIO_PIN_x</i>   | 引脚选择(x=0..15)  |
| <i>GPIO_PIN_ALL</i> | 所有引脚   |
| <b>输出参数{out}</b>    |  |
| -                   | -  |
| <b>返回值</b>          |  |
| -                   | -  |

例如:

```
/*set PA0 alternate function 0*/
```

```
gpio_af_set(GPIOA, GPIO_AF_0, GPIO_PIN_0);
```

### 函数 gpio\_pin\_lock

函数gpio\_pin\_lock描述见下表：

表 3-371. 函数 gpio\_pin\_lock

|              |   |
|--------------|---|
| 函数名称         | gpio_pin_lock   |
| 函数原型         | void gpio_pin_lock(uint32_t gpio_periph, uint32_t pin); |
| 功能描述         | 相应的引脚配置被锁定  |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| gpio_periph  | GPIO端口  |
| GPIOx        | 端口选择(x = A,B,C,D,E,F)                                   |
| 输入参数{in}     |   |
| pin          | GPIO引脚  |
| GPIO_PIN_x   | 引脚选择(x=0..15)   |
| GPIO_PIN_ALL | 所有引脚  |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如：

```
/* lock PA0 */
```

```
gpio_pin_lock(GPIOA, GPIO_PIN_0);
```

### 函数 gpio\_bit\_toggle

函数gpio\_bit\_toggle描述见下表：

表 3-372. 函数 gpio\_bit\_toggle

|             |   |
|-------------|---|
| 函数名称        | gpio_bit_toggle   |
| 函数原型        | void gpio_bit_toggle(uint32_t gpio_periph, uint32_t pin); |
| 功能描述        | 翻转GPIO引脚状态  |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| gpio_periph | GPIO端口  |
| GPIOx       | GPIOx(x = A,B,C,D,E,F)                                    |
| 输入参数{in}    |   |

|                     |               |
|---------------------|---------------|
| <b>pin</b>          | GPIO引脚        |
| <i>GPIO_PIN_x</i>   | 引脚选择(x=0..15) |
| <i>GPIO_PIN_ALL</i> | 所有引脚          |
| <b>输出参数{out}</b>    |               |
| -                   | -             |
| <b>返回值</b>          |               |
| -                   | -             |

例如:

```
/* toggle PA0 */
```

```
gpio_bit_toggle(GPIOA, GPIO_PIN_0);
```

### 函数 gpio\_port\_toggle

函数gpio\_port\_toggle描述见下表:

**表 3-373. 函数 gpio\_port\_toggle**

|                    |  |
|--------------------|--|
| <b>函数名称</b>        | gpio_port_toggle                             |
| <b>函数原型</b>        | void gpio_port_toggle(uint32_t gpio_periph); |
| <b>功能描述</b>        | 翻转一组GPIO状态                                   |
| <b>先决条件</b>        | -  |
| <b>被调用函数</b>       | -  |
| <b>输入参数{in}</b>    |  |
| <b>gpio_periph</b> | GPIO端口                                       |
| <i>GPIOx</i>       | GPIOx(x = A,B,C,D,E,F)                       |
| <b>输出参数{out}</b>   |  |
| -                  | -  |
| <b>返回值</b>         |  |
| -                  | -  |

例如:

```
/* toggle GPIOA*/
```

```
gpio_port_toggle(GPIOA);
```

## 3.14. I2C

I2C（内部集成电路总线）模块提供了符合工业标准的两线串行制接口，可用于MCU和外部I2C设备的通讯。章节[3.14.1](#)描述了I2C的寄存器列表，章节[3.14.2](#)对I2C库函数进行说明。

### 3.14.1. 外设寄存器说明

I2C寄存器列表如下表所示:

表 3-374. I2C 寄存器

| 寄存器名称       | 寄存器描述     |
|-------------|-----------|
| I2C_CTL0    | 控制寄存器 0   |
| I2C_CTL1    | 控制寄存器 1   |
| I2C_SADDR0  | 从机地址寄存器 0 |
| I2C_SADDR1  | 从机地址寄存器 1 |
| I2C_TIMING  | 时序寄存器     |
| I2C_TIMEOUT | 超时寄存器     |
| I2C_STAT    | 状态寄存器     |
| I2C_STATC   | 状态清除寄存器   |
| I2C_PEC     | PEC 寄存器   |
| I2C_RDATA   | 接收数据寄存器   |
| I2C_TDATA   | 发送数据寄存器   |
| I2C_CTL2    | 控制寄存器 2   |

### 3.14.2. 外设库函数说明

I2C库函数列表如下表所示：

表 3-375. I2C 库函数

| 库函数名称                               | 库函数描述                          |
|-------------------------------------|--------------------------------|
| i2c_deinit                          | 复位外设 I2C                       |
| i2c_timing_config                   | 配置时序参数                         |
| i2c_digital_noise_filter_config     | 配置数字噪声过滤器                      |
| i2c_analog_noise_filter_enable      | 使能数字噪声过滤器                      |
| i2c_analog_noise_filter_disable     | 禁能数字噪声过滤器                      |
| i2c_master_clock_config             | 配置主机模式下 SCL 高低电平时钟周期           |
| i2c_master_addressing               | 配置 I2C 从机地址以及数据传输方向            |
| i2c_address10_header_enable         | 主机接收模式下，10 位地址头只执行读操作          |
| i2c_address10_header_disable        | 主机接收模式下，10 位地址头执行完整的读操作序列      |
| i2c_address10_enable                | 使能主机模式下 10 位地址寻址模式             |
| i2c_address10_disable               | 禁能主机模式下 10 位地址寻址模式             |
| i2c_automatic_end_enable            | 使能主机模式下 I2C 自动结束模式             |
| i2c_automatic_end_disable           | 禁能主机模式下 I2C 自动结束模式             |
| i2c_slave_response_to_gcall_enable  | 使能从机响应广播呼叫                     |
| i2c_slave_response_to_gcall_disable | 禁能从机响应广播呼叫                     |
| i2c_stretch_scl_low_enable          | 当从机数据没有准备好时拉低 SCL              |
| i2c_stretch_scl_low_disable         | 当从机数据没有准备好时不拉低 SCL             |
| i2c_address_config                  | 配置 I2C 从机地址                    |
| i2c_address_bit_compare_config      | 定义 ADDRESS[7:1]的哪些位和接收到的地址进行比较 |
| i2c_address_disable                 | 禁能从机模式下 I2C 地址                 |
| i2c_second_address_config           | 配置 I2C 从机第二个地址                 |
| i2c_second_address_disable          | 禁能 I2C 从机第二个地址                 |

| 库函数名称                              | 库函数描述           |
|------------------------------------|-----------------|
| i2c_receivied_address_get          | 获取从机模式下匹配成功的地址  |
| i2c_slave_byte_control_enable      | 使能从机字节控制        |
| i2c_slave_byte_control_disable     | 禁能从机字节控制        |
| i2c_nack_enable                    | 从机模式下产生 NACK    |
| i2c_nack_disable                   | 从机模式下产生 ACK     |
| i2c_enable                         | 使能 I2C          |
| i2c_disable                        | 禁能 I2C          |
| i2c_start_on_bus                   | 在 I2C 总线上生成起始位  |
| i2c_stop_on_bus                    | 在 I2C 总线上生成停止位  |
| i2c_data_transmit                  | 发送数据            |
| i2c_data_receive                   | 接收数据            |
| i2c_reload_enable                  | 使能 I2C 重载模式     |
| i2c_reload_disable                 | 禁能 I2C 重载模式     |
| i2c_transfer_byte_number_config    | 配置待发送字节数        |
| i2c_dma_enable                     | 使能发送/接收模式下 DMA  |
| i2c_dma_disable                    | 禁能发送/接收模式下 DMA  |
| i2c_pec_transfer                   | I2C 传输 PEC 值    |
| i2c_pec_enable                     | 使能报文错误校验        |
| i2c_pec_disable                    | 禁能报文错误校验        |
| i2c_pec_value_get                  | 获取报文错误校验值       |
| i2c_smbus_alert_enable             | 使能 SMBus 报警     |
| i2c_smbus_alert_disable            | 禁能 SMBus 报警     |
| i2c_smbus_default_addr_enable      | 使能 SMBus 设备默认地址 |
| i2c_smbus_default_addr_disable     | 禁能 SMBus 设备默认地址 |
| i2c_smbus_host_addr_enable         | 使能 SMBus 主机地址   |
| i2c_smbus_host_addr_disable        | 禁能 SMBus 主机地址   |
| i2c_extented_clock_timeout_enable  | 使能时钟信号延展超时检测    |
| i2c_extented_clock_timeout_disable | 禁能时钟信号延展超时检测    |
| i2c_clock_timeout_enable           | 使能时钟超时检测        |
| i2c_clock_timeout_disable          | 禁能时钟超时检测        |
| i2c_bus_timeout_b_config           | 配置总线超时 B        |
| i2c_bus_timeout_a_config           | 配置总线超时 A        |
| i2c_idle_clock_timeout_config      | 配置空闲时钟超时检测      |
| i2c_flag_get                       | 获取 I2C 标志位      |
| i2c_flag_clear                     | 清除 I2C 标志位      |
| i2c_interrupt_enable               | 中断使能            |
| i2c_interrupt_disable              | 中断除能            |
| i2c_interrupt_flag_get             | 获取中断标志位         |
| i2c_interrupt_flag_clear           | 清除中断标志位         |



枚举类型 `i2c_interrupt_flag_enum`表 3-376. 枚举类型 `i2c_interrupt_flag_enum`

| 枚举名称                              | 枚举描述                    |
|-----------------------------------|-------------------------|
| <code>I2C_INT_FLAG_TI</code>      | 发送中断标志                  |
| <code>I2C_INT_FLAG_RBNE</code>    | 接收期间I2C_RDATA非空中断标志     |
| <code>I2C_INT_FLAG_ADDSEND</code> | 从机模式下，接收到的地址与自身地址匹配中断标志 |
| <code>I2C_INT_FLAG_NACK</code>    | NACK中断标志                |
| <code>I2C_INT_FLAG_STPDET</code>  | 从机模式下检测到STOP信号中断标志      |
| <code>I2C_INT_FLAG_TC</code>      | 主机模式下传输完成中断标志           |
| <code>I2C_INT_FLAG_TCR</code>     | 传输完成重载中断标志              |
| <code>I2C_INT_FLAG_BERR</code>    | 总线错误中断标志                |
| <code>I2C_INT_FLAG_LOSTARB</code> | 仲裁丢失中断标志                |
| <code>I2C_INT_FLAG_OUERR</code>   | 从机模式下，过载/欠载错误中断标志       |
| <code>I2C_INT_FLAG_PECERR</code>  | PEC错误中断标志               |
| <code>I2C_INT_FLAG_TIMEOUT</code> | 超时中断标志                  |
| <code>I2C_INT_FLAG_SMBALT</code>  | SMBus报警中断标志             |

函数 `i2c_deinit`

函数*i2c\_deinit*描述见下表：

表 3-377. 函数 `i2c_deinit`

|                         |  |
|-------------------------|--|
| 函数名称                    | <code>i2c_deinit</code>  |
| 函数原型                    | <code>void i2c_deinit(uint32_t i2c_periph);</code>                           |
| 功能描述                    | 复位外设 I2C   |
| 先决条件                    | -  |
| 被调用函数                   | <code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code> |
| 输入参数{in}                |  |
| <code>i2c_periph</code> | I2C 外设   |
| <code>I2Cx</code>       | (x=0,1)  |
| 输出参数{out}               |  |
| -                       | -  |
| 返回值                     |  |
| -                       | -  |

例如：

```
/* reset I2C0 */
i2c_deinit(I2C0);
```

函数 `i2c_timing_config`

函数*i2c\_timing\_config*描述见下表：

表 3-378. 函数 i2c\_timing\_config

|            |  |
|------------|--|
| 函数名称       | i2c_timing_config  |
| 函数原型       | void i2c_timing_config(uint32_t i2c_periph, uint32_t psc, uint32_t scl_dely, uint32_t sda_dely); |
| 功能描述       | 配置时序参数   |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设   |
| I2Cx       | (x=0,1)  |
| 输入参数{in}   |  |
| psc        | 0-0x0000000F, 时序分频   |
| 输入参数{in}   |  |
| scl_dely   | 0-0x0000000F, 数据建立时间   |
| 输入参数{in}   |  |
| sda_dely   | 0-0x0000000F, 数据保持时间   |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如:

```
/* configure the timing parameters */
```

```
i2c_timing_config(I2C0, 0x1, 0x2, 0x1);
```

### 函数 i2c\_digital\_noise\_filter\_config

函数i2c\_digital\_noise\_filter\_config描述见下表:

表 3-379. 函数 i2c\_digital\_noise\_filter\_config

|                 |  |
|-----------------|--|
| 函数名称            | i2c_digital_noise_filter_config  |
| 函数原型            | void i2c_digital_noise_filter_config(uint32_t i2c_periph, uint32_t filter_length); |
| 功能描述            | 配置数字噪声过滤器  |
| 先决条件            | -  |
| 被调用函数           | -  |
| 输入参数{in}        |  |
| i2c_periph      | I2C 外设   |
| I2Cx            | (x=0,1)  |
| 输入参数{in}        |  |
| filter_length   | 过滤长度   |
| FILTER_DISABLE  | 数字噪声过滤器禁能  |
| FILTER_LENGTH_1 | 数字噪声滤波使能并且可以滤除脉宽宽度不大于 1 t <sub>12CCLK</sub> 的尖峰                                    |

|                         |   |
|-------------------------|---|
| <i>FILTER_LENGTH_2</i>  | 数字噪声滤波使能并且可以滤除脉宽宽度不大于 2 $t_{12CCLK}$ 的尖峰  |
| <i>FILTER_LENGTH_3</i>  | 数字噪声滤波使能并且可以滤除脉宽宽度不大于 3 $t_{12CCLK}$ 的尖峰  |
| <i>FILTER_LENGTH_4</i>  | 数字噪声滤波使能并且可以滤除脉宽宽度不大于 4 $t_{12CCLK}$ 的尖峰  |
| <i>FILTER_LENGTH_5</i>  | 数字噪声滤波使能并且可以滤除脉宽宽度不大于 5 $t_{12CCLK}$ 的尖峰  |
| <i>FILTER_LENGTH_6</i>  | 数字噪声滤波使能并且可以滤除脉宽宽度不大于 6 $t_{12CCLK}$ 的尖峰  |
| <i>FILTER_LENGTH_7</i>  | 数字噪声滤波使能并且可以滤除脉宽宽度不大于 7 $t_{12CCLK}$ 的尖峰  |
| <i>FILTER_LENGTH_8</i>  | 数字噪声滤波使能并且可以滤除脉宽宽度不大于 8 $t_{12CCLK}$ 的尖峰  |
| <i>FILTER_LENGTH_9</i>  | 数字噪声滤波使能并且可以滤除脉宽宽度不大于 9 $t_{12CCLK}$ 的尖峰  |
| <i>FILTER_LENGTH_10</i> | 数字噪声滤波使能并且可以滤除脉宽宽度不大于 10 $t_{12CCLK}$ 的尖峰 |
| <i>FILTER_LENGTH_11</i> | 数字噪声滤波使能并且可以滤除脉宽宽度不大于 11 $t_{12CCLK}$ 的尖峰 |
| <i>FILTER_LENGTH_12</i> | 数字噪声滤波使能并且可以滤除脉宽宽度不大于 12 $t_{12CCLK}$ 的尖峰 |
| <i>FILTER_LENGTH_13</i> | 数字噪声滤波使能并且可以滤除脉宽宽度不大于 13 $t_{12CCLK}$ 的尖峰 |
| <i>FILTER_LENGTH_14</i> | 数字噪声滤波使能并且可以滤除脉宽宽度不大于 14 $t_{12CCLK}$ 的尖峰 |
| <i>FILTER_LENGTH_15</i> | 数字噪声滤波使能并且可以滤除脉宽宽度不大于 15 $t_{12CCLK}$ 的尖峰 |
| 输出参数{out}               |   |
| -                       | -   |
| 返回值                     |   |
| -                       | -   |

例如：

```
/* I2C0 digital filter filters spikes with a length of up to 1  $t_{12CCLK}$  */
```

```
i2c_digital_noise_filter_config(I2C0, FILTER_LENGTH_1);
```

### 函数 i2c\_analog\_noise\_filter\_enable

函数 i2c\_analog\_noise\_filter\_enable 描述见下表：

表 3-380. 函数 i2c\_analog\_noise\_filter\_enable

|                   |   |
|-------------------|---|
| 函数名称              | i2c_analog_noise_filter_enable                            |
| 函数原型              | void i2c_analog_noise_filter_enable(uint32_t i2c_periph); |
| 功能描述              | 使能模拟噪声滤波器   |
| 先决条件              | -   |
| 被调用函数             | -   |
| 输入参数{in}          |   |
| <b>i2c_periph</b> | I2C 外设  |
| <i>I2Cx</i>       | (x=0,1)   |
| 输出参数{out}         |   |
| -                 | -   |
| 返回值               |   |
| -                 | -   |

例如：

```
/* enable analog noise filter */
```

```
i2c_analog_noise_filter_enable(I2C0);
```

### 函数 i2c\_analog\_noise\_filter\_disable

函数i2c\_analog\_noise\_filter\_disable描述见下表：

**表 3-381. 函数 i2c\_analog\_noise\_filter\_disable**

|            |  |
|------------|--|
| 函数名称       | i2c_analog_noise_filter_disable                            |
| 函数原型       | void i2c_analog_noise_filter_disable(uint32_t i2c_periph); |
| 功能描述       | 禁能模拟噪声滤波器  |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C外设  |
| I2Cx       | (x=0,1)  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如：

```
/* disable analog noise filter */
```

```
i2c_analog_noise_filter_disable(I2C0);
```

### 函数 i2c\_master\_clock\_config

函数i2c\_master\_clock\_config描述见下表：

**表 3-382. 函数 i2c\_master\_clock\_config**

|            |  |
|------------|--|
| 函数名称       | i2c_master_clock_config  |
| 函数原型       | void i2c_master_clock_config(uint32_t i2c_periph, uint32_t sclh, uint32_t scll); |
| 功能描述       | 配置主机模式下 SCL 高低电平周期   |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设   |
| I2Cx       | (x=0,1)  |
| 输入参数{in}   |  |
| sclh       | 0-0xff, SCL 高电平周期  |
| 输入参数{in}   |  |
| scll       | 0-0xff, SCL 低电平周期  |
| 输出参数{out}  |  |
| -          | -  |

| 返回值 |   |
|-----|---|
| -   | - |

例如：

```
/* configure the SCL and SDA period of clock in master mode */
```

```
i2c_master_clock_config(I2C0, 0x0f, 0x0f);
```

### 函数 i2c\_master\_addressing

函数i2c\_master\_addressing描述见下表：

**表 3-383. 函数 i2c\_master\_addressing**

| 函数名称                       | i2c_master_addressing  |
|----------------------------|--|
| 函数原型                       | void i2c_master_addressing(uint32_t i2c_periph, uint32_t address, uint32_t trans_direction); |
| 功能描述                       | 配置 I2C 从机地址以及数据传输方向  |
| 先决条件                       | -  |
| 被调用函数                      | -  |
| 输入参数{in}                   |  |
| <b>i2c_periph</b>          | I2C 外设   |
| <i>I2Cx</i>                | (x=0,1)  |
| 输入参数{in}                   |  |
| <b>address</b>             | 除保留地址外的地址，0-0x3FF，由主机发送给从机的地址  |
| 输入参数{in}                   |  |
| <b>trans_direction</b>     | 主机模式下，I2C 传输方向   |
| <i>I2C_MASTER_TRANSMIT</i> | 主机发送   |
| <i>I2C_MASTER_RECEIVE</i>  | 主机接收   |
| 输出参数{out}                  |  |
| -                          | -  |
| 返回值                        |  |
| -                          | -  |

例如：

```
/* send slave address to I2C bus */
```

```
i2c_master_addressing(I2C0, 0x82, I2C_MASTER_TRANSMIT);
```

### 函数 i2c\_address10\_header\_enable

函数i2c\_address10\_header\_enable描述见下表：

表 3-384. 函数 i2c\_address10\_header\_enable

|            |  |
|------------|--|
| 函数名称       | i2c_address10_header_enable                            |
| 函数原型       | void i2c_address10_header_enable(uint32_t i2c_periph); |
| 功能描述       | 主机接收模式下，10 位地址头只执行读操作                                  |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设   |
| I2Cx       | (x=0,1)  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如：

```
/* 10-bit address header executes read direction only in master receive mode */
```

```
i2c_address10_header_enable(I2C0);
```

### 函数 i2c\_address10\_header\_disable

函数i2c\_address10\_header\_disable描述见下表：

表 3-385. 函数 i2c\_address10\_header\_disable

|            |   |
|------------|---|
| 函数名称       | i2c_address10_header_disable                            |
| 函数原型       | void i2c_address10_header_disable(uint32_t i2c_periph); |
| 功能描述       | 主机接收模式下，10 位地址头执行完整的读操作序列                               |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设  |
| I2Cx       | (x=0,1)   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如：

```
/* 10-bit address header executes complete sequence in master receive mode */
```

```
i2c_address10_header_disable(I2C0);
```

### 函数 i2c\_address10\_enable

函数i2c\_address10\_enable描述见下表：

表 3-386. 函数 i2c\_address10\_enable

|            |   |
|------------|---|
| 函数名称       | i2c_address10_enable                            |
| 函数原型       | void i2c_address10_enable(uint32_t i2c_periph); |
| 功能描述       | 使能主机模式下 10 位地址寻址模式                              |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设  |
| I2Cx       | (x=0,1)   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如：

```
/* enable 10-bit addressing mode in master mode */
```

```
i2c_address10_enable(I2C0);
```

### 函数 i2c\_address10\_disable

函数i2c\_address10\_disable描述见下表：

表 3-387. 函数 i2c\_address10\_disable

|            |  |
|------------|--|
| 函数名称       | i2c_address10_disable                            |
| 函数原型       | void i2c_address10_disable(uint32_t i2c_periph); |
| 功能描述       | 禁能主机模式下 10 位地址寻址模式                               |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设   |
| I2Cx       | (x=0,1)  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如：

```
/* disable 10-bit addressing mode in master mode */
```

```
i2c_address10_disable(I2C0);
```

### 函数 i2c\_automatic\_end\_enable

函数i2c\_automatic\_end\_enable描述见下表：

表 3-388. 函数 i2c\_automatic\_end\_enable

|            |   |
|------------|---|
| 函数名称       | i2c_automatic_end_enable                            |
| 函数原型       | void i2c_automatic_end_enable(uint32_t i2c_periph); |
| 功能描述       | 使能主机模式下 I2C 自动结束模式                                  |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设  |
| I2Cx       | (x=0,1)   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如：

```
/* enable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_enable(I2C0);
```

### 函数 i2c\_automatic\_end\_disable

函数i2c\_automatic\_end\_disable描述见下表：

表 3-389. 函数 i2c\_automatic\_end\_disable

|            |  |
|------------|--|
| 函数名称       | i2c_automatic_end_disable                            |
| 函数原型       | void i2c_automatic_end_disable(uint32_t i2c_periph); |
| 功能描述       | 禁能主机模式下 I2C 自动结束模式                                   |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设   |
| I2Cx       | (x=0,1)  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如：

```
/* disable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_disable(I2C0);
```

### 函数 i2c\_slave\_response\_to\_gcall\_enable

函数i2c\_slave\_response\_to\_gcall\_enable描述见下表：



表 3-390. 函数 i2c\_slave\_response\_to\_gcall\_enable

|            |   |
|------------|---|
| 函数名称       | i2c_slave_response_to_gcall_enable                            |
| 函数原型       | void i2c_slave_response_to_gcall_enable(uint32_t i2c_periph); |
| 功能描述       | 使能从机响应广播呼叫  |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设  |
| I2Cx       | (x=0,1)   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如:

```
/* enable the response to a general call */
```

```
i2c_slave_response_to_gcall_enable(I2C0);
```

### 函数 i2c\_slave\_response\_to\_gcall\_disable

函数i2c\_slave\_response\_to\_gcall\_disable描述见下表:

表 3-391. 函数 i2c\_slave\_response\_to\_gcall\_disable

|            |  |
|------------|--|
| 函数名称       | i2c_slave_response_to_gcall_disable                            |
| 函数原型       | void i2c_slave_response_to_gcall_disable(uint32_t i2c_periph); |
| 功能描述       | 禁能从机响应广播呼叫   |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设   |
| I2Cx       | (x=0,1)  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如:

```
/* disable the response to a general call */
```

```
i2c_slave_response_to_gcall_disable(I2C0);
```

### 函数 i2c\_stretch\_scl\_low\_enable

函数i2c\_stretch\_scl\_low\_enable描述见下表:

表 3-392. 函数 i2c\_stretch\_scl\_low\_enable

|            |   |
|------------|---|
| 函数名称       | i2c_stretch_scl_low_enable                            |
| 函数原型       | void i2c_stretch_scl_low_enable(uint32_t i2c_periph); |
| 功能描述       | 当从机数据没有准备好时拉低 SCL                                     |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设  |
| I2Cx       | (x=0,1)   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如:

```
/* enable to stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_enable(I2C0);
```

### 函数 i2c\_stretch\_scl\_low\_disable

函数i2c\_stretch\_scl\_low\_disable描述见下表:

表 3-393. 函数 i2c\_stretch\_scl\_low\_disable

|            |  |
|------------|--|
| 函数名称       | i2c_stretch_scl_low_disable                            |
| 函数原型       | void i2c_stretch_scl_low_disable(uint32_t i2c_periph); |
| 功能描述       | 当从机数据没有准备好时不拉低 SCL                                     |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设   |
| I2Cx       | (x=0,1)  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如:

```
/* disable to stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_disable(I2C0);
```

### 函数 i2c\_address\_config

函数i2c\_address\_config描述见下表:

表 3-394. 函数 i2c\_address\_config

|                      |   |
|----------------------|---|
| 函数名称                 | i2c_address_config  |
| 函数原型                 | void i2c_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_format); |
| 功能描述                 | 配置 I2C 从机地址   |
| 先决条件                 | -   |
| 被调用函数                | -   |
| 输入参数{in}             |   |
| i2c_periph           | I2C 外设  |
| I2Cx                 | (x=0,1)   |
| 输入参数{in}             |   |
| address              | I2C 地址  |
| 输入参数{in}             |   |
| addr_format          | 7 位地址或 10 位地址   |
| I2C_ADDFORMAT_7BITS  | 7 位地址   |
| I2C_ADDFORMAT_10BITS | 10 位地址  |
| 输出参数{out}            |   |
| -                    | -   |
| 返回值                  |   |
| -                    | -   |

例如:

```
/* configure i2c slave address */
```

```
i2c_address_config(I2C0, 0x82, I2C_ADDFORMAT_7BITS);
```

### 函数 i2c\_address\_bit\_compare\_config

函数 i2c\_address\_bit\_compare\_config 描述见下表:

表 3-395. 函数 i2c\_address\_bit\_compare\_config

|              |  |
|--------------|--|
| 函数名称         | i2c_address_bit_compare_config   |
| 函数原型         | void i2c_address_bit_compare_config(uint32_t i2c_periph, uint32_t compare_bits); |
| 功能描述         | 定义 ADDRESS[7:1] 的哪些位和接收到的地址进行比较  |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| i2c_periph   | I2C peripheral   |
| I2Cx         | (x=0,1)  |
| 输入参数{in}     |  |
| compare_bits | 需要进行比较的位   |

|                      |                |
|----------------------|----------------|
| ADDRESS_BIT1_COMPARE | 地址的第 1 位需要进行比较 |
| ADDRESS_BIT2_COMPARE | 地址的第 2 位需要进行比较 |
| ADDRESS_BIT3_COMPARE | 地址的第 3 位需要进行比较 |
| ADDRESS_BIT4_COMPARE | 地址的第 4 位需要进行比较 |
| ADDRESS_BIT5_COMPARE | 地址的第 5 位需要进行比较 |
| ADDRESS_BIT6_COMPARE | 地址的第 6 位需要进行比较 |
| ADDRESS_BIT7_COMPARE | 地址的第 7 位需要进行比较 |
| 输出参数{out}            |                |
| -                    | -              |
| 返回值                  |                |
| -                    | -              |

例如：

```
/* define which bits of ADDRESS[7:1] need to compare with the incoming address byte */
```

```
i2c_address_bit_compare_config(I2C0, ADDRESS_BIT1_COMPARE);
```

### 函数 i2c\_address\_disable

函数i2c\_address\_disable描述见下表：

表 3-396. 函数 i2c\_address\_disable

|            |  |
|------------|--|
| 函数名称       | i2c_address_disable                            |
| 函数原型       | void i2c_address_disable(uint32_t i2c_periph); |
| 功能描述       | 禁能从机模式下 I2C 地址                                 |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设   |
| I2Cx       | (x=0,1)  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如：

```
/* disable i2c address in slave mode */
```

```
i2c_address_disable(I2C0);
```

### 函数 i2c\_second\_address\_config

函数i2c\_second\_address\_config描述见下表：

**表 3-397. 函数 i2c\_second\_address\_config**

|                      |  |
|----------------------|--|
| 函数名称                 | i2c_second_address_config  |
| 函数原型                 | void i2c_second_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_mask); |
| 功能描述                 | 配置 I2C 从机第二个地址   |
| 先决条件                 | -  |
| 被调用函数                | -  |
| 输入参数{in}             |  |
| i2c_periph           | I2C 外设   |
| I2Cx                 | (x=0,1)  |
| 输入参数{in}             |  |
| address              | I2C 地址   |
| 输入参数{in}             |  |
| addr_mask            | 不需要进行比较的地址   |
| ADDRESS2_NO_MASK     | 无屏蔽，全部都需要进行比较  |
| ADDRESS2_MASK_BIT1   | ADDRESS2[1]屏蔽， ADDRESS2[7:2]进行比较   |
| ADDRESS2_MASK_BIT1_2 | ADDRESS2[2:1]屏蔽， ADDRESS2[7:3]进行比较   |
| ADDRESS2_MASK_BIT1_3 | ADDRESS2[3:1]屏蔽， ADDRESS2[7:4]进行比较   |
| ADDRESS2_MASK_BIT1_4 | ADDRESS2[4:1]屏蔽， ADDRESS2[7:5]进行比较   |
| ADDRESS2_MASK_BIT1_5 | ADDRESS2[5:1]屏蔽， ADDRESS2[7:6]进行比较   |
| ADDRESS2_MASK_BIT1_6 | ADDRESS2[6:1]屏蔽， ADDRESS2[7]进行比较   |
| ADDRESS2_MASK_ALL    | ADDRESS2[7:1]屏蔽  |
| 输出参数{out}            |  |
| -                    | -  |
| 返回值                  |  |
| -                    | -  |

例如：

```
/* configure i2c second slave address */
```

```
i2c_second_address_config(I2C0, 0x82, ADDRESS2_MASK_BIT1_2);
```

### 函数 i2c\_second\_address\_disable

函数i2c\_second\_address\_disable描述见下表:

**表 3-398. 函数 i2c\_second\_address\_disable**

|            |   |
|------------|---|
| 函数名称       | i2c_second_address_disable                            |
| 函数原型       | void i2c_second_address_disable(uint32_t i2c_periph); |
| 功能描述       | 禁能 I2C 从机第二个地址  |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设  |
| I2Cx       | (x=0,1)   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如:

```
/* disable i2c second address in slave mode */
```

```
i2c_second_address_disable(I2C0);
```

### 函数 i2c\_receved\_address\_get

函数i2c\_receved\_address\_get描述见下表:

**表 3-399. 函数 i2c\_receved\_address\_get**

|            |  |
|------------|--|
| 函数名称       | i2c_receved_address_get                                |
| 函数原型       | uint32_t i2c_receved_address_get(uint32_t i2c_periph); |
| 功能描述       | 获取从机模式下匹配成功的地址   |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设   |
| I2Cx       | (x=0,1)  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| uint32_t   | 0x00000000..0x0000007F                                 |

例如:

```
/* get received match address in slave mode */
```

```
uint32_t address;
```

```
address = i2c_receivied_address_get(I2C0);
```

### 函数 i2c\_slave\_byte\_control\_enable

函数i2c\_slave\_byte\_control\_enable描述见下表：

表 3-400. 函数 i2c\_slave\_byte\_control\_enable

|            |  |
|------------|--|
| 函数名称       | i2c_slave_byte_control_enable                            |
| 函数原型       | void i2c_slave_byte_control_enable(uint32_t i2c_periph); |
| 功能描述       | 使能从机字节控制   |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设   |
| I2Cx       | (x=0,1)  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如：

```
/* enable slave byte control */
```

```
i2c_slave_byte_control_enable(I2C0);
```

### 函数 i2c\_slave\_byte\_control\_disable

函数i2c\_slave\_byte\_control\_disable描述见下表：

表 3-401. 函数 i2c\_slave\_byte\_control\_disable

|            |   |
|------------|---|
| 函数名称       | i2c_slave_byte_control_disable                            |
| 函数原型       | void i2c_slave_byte_control_disable(uint32_t i2c_periph); |
| 功能描述       | 禁能从机字节控制  |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设  |
| I2Cx       | (x=0,1)   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如：

```
/* disable slave byte control */
```

```
i2c_slave_byte_control_disable(I2C0);
```

### 函数 i2c\_nack\_enable

函数i2c\_nack\_enable描述见下表:

表 3-402. 函数 i2c\_nack\_enable

|            |  |
|------------|--|
| 函数名称       | i2c_nack_enable                            |
| 函数原型       | void i2c_nack_enable(uint32_t i2c_periph); |
| 功能描述       | 从机模式下产生 NACK                               |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设                                     |
| I2Cx       | (x=0,1)                                    |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如:

```
/* generate a NACK in slave mode */
```

```
i2c_nack_enable(I2C0);
```

### 函数 i2c\_nack\_disable

函数i2c\_nack\_disable描述见下表:

表 3-403. 函数 i2c\_nack\_disable

|            |   |
|------------|---|
| 函数名称       | i2c_nack_disable                            |
| 函数原型       | void i2c_nack_disable(uint32_t i2c_periph); |
| 功能描述       | 从机模式下产生 ACK                                 |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设                                      |
| I2Cx       | (x=0,1)                                     |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如:



```
/* generate a ACK in slave mode */
```

```
i2c_nack_disable(I2C0);
```

### 函数 i2c\_enable

函数i2c\_enable描述见下表：

表 3-404. 函数 i2c\_enable

|            |                                       |
|------------|---------------------------------------|
| 函数名称       | i2c_enable                            |
| 函数原型       | void i2c_enable(uint32_t i2c_periph); |
| 功能描述       | 使能 I2C                                |
| 先决条件       | -                                     |
| 被调用函数      | -                                     |
| 输入参数{in}   |                                       |
| i2c_periph | I2C 外设                                |
| I2Cx       | (x=0,1)                               |
| 输出参数{out}  |                                       |
| -          | -                                     |
| 返回值        |                                       |
| -          | -                                     |

例如：

```
/* enable I2C0 */
```

```
i2c_enable(I2C0);
```

### 函数 i2c\_disable

函数i2c\_disable描述见下表：

表 3-405. 函数 i2c\_disable

|            |  |
|------------|--|
| 函数名称       | i2c_disable                            |
| 函数原型       | void i2c_disable(uint32_t i2c_periph); |
| 功能描述       | 禁能 I2C                                 |
| 先决条件       | -                                      |
| 被调用函数      | -                                      |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设                                 |
| I2Cx       | (x=0,1)                                |
| 输出参数{out}  |  |
| -          | -                                      |
| 返回值        |  |
| -          | -                                      |

例如：

```
/* disable I2C0 */
```

```
i2c_disable(I2C0);
```

### 函数 i2c\_start\_on\_bus

函数i2c\_start\_on\_bus描述见下表:

表 3-406. 函数 i2c\_start\_on\_bus

|            |   |
|------------|---|
| 函数名称       | i2c_start_on_bus                            |
| 函数原型       | void i2c_start_on_bus(uint32_t i2c_periph); |
| 功能描述       | 在 I2C 总线上生成起始位                              |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设                                      |
| I2Cx       | (x=0,1)                                     |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如:

```
/* I2C0 send a start condition to I2C bus */
```

```
i2c_start_on_bus(I2C0);
```

### 函数 i2c\_stop\_on\_bus

函数i2c\_stop\_on\_bus描述见下表:

表 3-407. 函数 i2c\_stop\_on\_bus

|            |  |
|------------|--|
| 函数名称       | i2c_stop_on_bus                            |
| 函数原型       | void i2c_stop_on_bus(uint32_t i2c_periph); |
| 功能描述       | 在 I2C 总线上生成停止位                             |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设                                     |
| I2Cx       | (x=0,1)                                    |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如:

```
/* I2C0 generate a STOP condition to I2C bus */
```

```
i2c_stop_on_bus(I2C0);
```

### 函数 i2c\_data\_transmit

函数i2c\_data\_transmit描述见下表：

**表 3-408. 函数 i2c\_data\_transmit**

|            |   |
|------------|---|
| 函数名称       | i2c_data_transmit   |
| 函数原型       | void i2c_data_transmit(uint32_t i2c_periph, uint32_t data); |
| 功能描述       | 发送数据  |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设  |
| I2Cx       | (x=0,1)   |
| 输入参数{in}   |   |
| data       | transmit data   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如：

```
/* I2C0 transmit data */
```

```
i2c_data_transmit(I2C0, 0x80);
```

### 函数 i2c\_data\_receive

函数i2c\_data\_receive描述见下表：

**表 3-409. 函数 i2c\_data\_receive**

|            |   |
|------------|---|
| 函数名称       | i2c_data_receive                                |
| 函数原型       | uint32_t i2c_data_receive(uint32_t i2c_periph); |
| 功能描述       | 接收数据  |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设  |
| I2Cx       | (x=0,1)   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |

|          |                        |
|----------|------------------------|
| uint32_t | 0x00000000..0x000000FF |
|----------|------------------------|

例如:

```
/* I2C0 receive data */
```

```
uint32_t i2c_receiver;
```

```
i2c_receiver = i2c_data_receive(I2C0);
```

### 函数 i2c\_reload\_enable

函数i2c\_reload\_enable描述见下表:

表 3-410. 函数 i2c\_reload\_enable

|            |  |
|------------|--|
| 函数名称       | i2c_reload_enable                            |
| 函数原型       | void i2c_reload_enable(uint32_t i2c_periph); |
| 功能描述       | 使能 I2C 重载模式                                  |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设                                       |
| I2Cx       | (x=0,1)                                      |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如:

```
/* enable I2C reload mode */
```

```
i2c_reload_enable(I2C0);
```

### 函数 i2c\_reload\_disable

函数i2c\_reload\_disable描述见下表:

表 3-411. 函数 i2c\_reload\_disable

|            |   |
|------------|---|
| 函数名称       | i2c_reload_disable                            |
| 函数原型       | void i2c_reload_disable(uint32_t i2c_periph); |
| 功能描述       | 禁能 I2C 重载模式                                   |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设  |
| I2Cx       | (x=0,1)                                       |
| 输出参数{out}  |   |

|     |   |
|-----|---|
| -   | - |
| 返回值 |   |
| -   | - |

例如：

```
/* disable I2C reload mode */
```

```
i2c_reload_disable(I2C0);
```

### 函数 i2c\_transfer\_byte\_number\_config

函数i2c\_transfer\_byte\_number\_config描述见下表：

表 3-412. 函数 i2c\_transfer\_byte\_number\_config

|             |  |
|-------------|--|
| 函数名称        | i2c_transfer_byte_number_config  |
| 函数原型        | void i2c_transfer_byte_number_config(uint32_t i2c_periph, uint32_t byte_number); |
| 功能描述        | 配置待发送字节数   |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| i2c_periph  | I2C 外设   |
| I2Cx        | (x=0,1)  |
| 输入参数{in}    |  |
| byte_number | 0x0-0xFF，待传输的字节数   |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如：

```
/* configure number of bytes to be transferred */
```

```
i2c_transfer_byte_number_config(I2C0, 0xFF);
```

### 函数 i2c\_dma\_enable

函数i2c\_dma\_enable描述见下表：

表 3-413. 函数 i2c\_dma\_enable

|       |  |
|-------|--|
| 函数名称  | i2c_dma_enable   |
| 函数原型  | void i2c_dma_enable(uint32_t i2c_periph, uint8_t dma); |
| 功能描述  | 使能发送/接收模式下 DMA   |
| 先决条件  | -  |
| 被调用函数 | -  |

| 输入参数{in}                |               |
|-------------------------|---------------|
| <b>i2c_periph</b>       | I2C 外设        |
| <i>I2Cx</i>             | (x=0,1)       |
| 输入参数{in}                |               |
| <b>dma</b>              | I2C DMA       |
| <i>I2C_DMA_TRANSMIT</i> | 采用 DMA 方式发送数据 |
| <i>I2C_DMA_RECEIVE</i>  | 采用 DMA 方式接收数据 |
| 输出参数{out}               |               |
| -                       | -             |
| 返回值                     |               |
| -                       | -             |

例如：

```
/* enable I2C DMA for transmission or reception */
```

```
i2c_dma_enable(I2C0, I2C_DMA_RECEIVE);
```

### 函数 i2c\_dma\_disable

函数i2c\_dma\_disable描述见下表：

**表 3-414. 函数 i2c\_dma\_disable**

| 函数名称                    | i2c_dma_disable   |
|-------------------------|---|
| 函数原型                    | void i2c_dma_disable(uint32_t i2c_periph, uint8_t dma); |
| 功能描述                    | 禁能发送/接收模式下 DMA  |
| 先决条件                    | -   |
| 被调用函数                   | -   |
| 输入参数{in}                |   |
| <b>i2c_periph</b>       | I2C 外设  |
| <i>I2Cx</i>             | (x=0,1)   |
| 输入参数{in}                |   |
| <b>dma</b>              | I2C DMA   |
| <i>I2C_DMA_TRANSMIT</i> | 采用 DMA 方式发送数据   |
| <i>I2C_DMA_RECEIVE</i>  | 采用 DMA 方式接收数据   |
| 输出参数{out}               |   |
| -                       | -   |
| 返回值                     |   |
| -                       | -   |

例如：

```
/* disable I2C DMA for transmission or reception */
```

```
i2c_dma_disable(I2C0, I2C_DMA_RECEIVE);
```

**函数 i2c\_pec\_transfer**

函数i2c\_pec\_transfer描述见下表:

**表 3-415. 函数 i2c\_pec\_transfer**

|            |   |
|------------|---|
| 函数名称       | i2c_pec_transfer                            |
| 函数原型       | void i2c_pec_transfer(uint32_t i2c_periph); |
| 功能描述       | I2C 传输 PEC 值                                |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设                                      |
| I2Cx       | (x=0,1)                                     |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如:

```
/* I2C transfers PEC value */
```

```
i2c_pec_transfer(I2C0);
```

**函数 i2c\_pec\_enable**

函数i2c\_pec\_enable描述见下表:

**表 3-416. 函数 i2c\_pec\_enable**

|            |   |
|------------|---|
| 函数名称       | i2c_pec_enable                            |
| 函数原型       | void i2c_pec_enable(uint32_t i2c_periph); |
| 功能描述       | 使能报文错误校验                                  |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设                                    |
| I2Cx       | (x=0,1)                                   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如:

```
/* enable I2C PEC calculation */
```

```
i2c_pec_enable(I2C0);
```

## 函数 i2c\_pec\_disable

函数i2c\_pec\_disable描述见下表:

表 3-417. 函数 i2c\_pec\_disable

|            |  |
|------------|--|
| 函数名称       | i2c_pec_disable                            |
| 函数原型       | void i2c_pec_disable(uint32_t i2c_periph); |
| 功能描述       | 禁能报文错误校验                                   |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设                                     |
| I2Cx       | (x=0,1)                                    |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如:

```
/* disable I2C PEC calculation */
```

```
i2c_pec_disable(I2C0);
```

## 函数 i2c\_pec\_value\_get

函数i2c\_pec\_value\_get描述见下表:

表 3-418. 函数 i2c\_pec\_value\_get

|            |  |
|------------|--|
| 函数名称       | i2c_pec_value_get                                |
| 函数原型       | uint32_t i2c_pec_value_get(uint32_t i2c_periph); |
| 功能描述       | 获取报文错误校验值  |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设   |
| I2Cx       | (x=0,1)  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| uint32_t   | PEC 值  |

例如:

```
/* I2C0 get packet error checking value */
```

```
uint32_t pec_value;
```



```
pec_value = i2c_pec_value_get(I2C0);
```

### 函数 i2c\_smbus\_alert\_enable

函数i2c\_smbus\_alert\_enable描述见下表:

表 3-419. 函数 i2c\_smbus\_alert\_enable

|            |   |
|------------|---|
| 函数名称       | i2c_smbus_alert_enable                            |
| 函数原型       | void i2c_smbus_alert_enable(uint32_t i2c_periph); |
| 功能描述       | 使能 SMBus 报警                                       |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设  |
| I2Cx       | (x=0,1)   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如:

```
/* enable SMBus Alert */
```

```
i2c_smbus_alert_enable(I2C0);
```

### 函数 i2c\_smbus\_alert\_disable

函数i2c\_smbus\_alert\_disable描述见下表:

表 3-420. 函数 i2c\_smbus\_alert\_disable

|            |  |
|------------|--|
| 函数名称       | i2c_smbus_alert_disable                            |
| 函数原型       | void i2c_smbus_alert_disable(uint32_t i2c_periph); |
| 功能描述       | 禁能 SMBus 报警  |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设   |
| I2Cx       | (x=0,1)  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如:

```
/* disable SMBus Alert */
```

```
i2c_smbus_alert_disable(I2C0);
```

### 函数 i2c\_smbus\_default\_addr\_enable

函数i2c\_smbus\_default\_addr\_enable描述见下表：

表 3-421. 函数 i2c\_smbus\_default\_addr\_enable

|            |  |
|------------|--|
| 函数名称       | i2c_smbus_default_addr_enable                            |
| 函数原型       | void i2c_smbus_default_addr_enable(uint32_t i2c_periph); |
| 功能描述       | 使能 SMBus 设备默认地址  |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设   |
| I2Cx       | (x=0,1)  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如：

```
/* enable SMBus device default address */
```

```
i2c_smbus_default_addr_enable(I2C0);
```

### 函数 i2c\_smbus\_default\_addr\_disable

函数i2c\_smbus\_default\_addr\_disable描述见下表：

表 3-422. 函数 i2c\_smbus\_default\_addr\_disable

|            |   |
|------------|---|
| 函数名称       | i2c_smbus_default_addr_disable                            |
| 函数原型       | void i2c_smbus_default_addr_disable(uint32_t i2c_periph); |
| 功能描述       | 禁能 SMBus 设备默认地址   |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设  |
| I2Cx       | (x=0,1)   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如：

```
/* disable SMBus device default address */
```

```
i2c_smbus_default_addr_disable(I2C0);
```

### 函数 i2c\_smbus\_host\_addr\_enable

函数i2c\_smbus\_host\_addr\_enable描述见下表：

**表 3-423. 函数 i2c\_smbus\_host\_addr\_enable**

|            |   |
|------------|---|
| 函数名称       | i2c_smbus_host_addr_enable                            |
| 函数原型       | void i2c_smbus_host_addr_enable(uint32_t i2c_periph); |
| 功能描述       | 使能 SMBus 主机地址   |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设  |
| I2Cx       | (x=0,1)   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如：

```
/* enable SMBus Host address */
```

```
i2c_smbus_host_addr_enable(I2C0);
```

### 函数 i2c\_smbus\_host\_addr\_disable

函数i2c\_smbus\_host\_addr\_disable描述见下表：

**表 3-424. 函数 i2c\_smbus\_host\_addr\_disable**

|            |  |
|------------|--|
| 函数名称       | i2c_smbus_host_addr_disable                            |
| 函数原型       | void i2c_smbus_host_addr_disable(uint32_t i2c_periph); |
| 功能描述       | 禁能 SMBus 主机地址  |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设   |
| I2Cx       | (x=0,1)  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如：

```
/* disable SMBus Host address */
```

```
i2c_smbus_host_addr_disable(I2C0);
```

### 函数 i2c\_extented\_clock\_timeout\_enable

函数i2c\_extented\_clock\_timeout\_enable描述见下表：

**表 3-425. 函数 i2c\_extented\_clock\_timeout\_enable**

|            |  |
|------------|--|
| 函数名称       | i2c_extented_clock_timeout_enable                            |
| 函数原型       | void i2c_extented_clock_timeout_enable(uint32_t i2c_periph); |
| 功能描述       | 使能时钟信号延展超时检测   |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设   |
| I2Cx       | (x=0,1)  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如：

```
/* enable extended clock timeout detection */
```

```
i2c_extented_clock_timeout_enable(I2C0);
```

### 函数 i2c\_extented\_clock\_timeout\_disable

函数i2c\_extented\_clock\_timeout\_disable描述见下表：

**表 3-426. 函数 i2c\_extented\_clock\_timeout\_disable**

|            |   |
|------------|---|
| 函数名称       | i2c_extented_clock_timeout_disable                            |
| 函数原型       | void i2c_extented_clock_timeout_disable(uint32_t i2c_periph); |
| 功能描述       | 禁能扩展时钟超时检测  |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设  |
| I2Cx       | (x=0,1)   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如：

```
/* disable extended clock timeout detection */
```

i2c\_extented\_clock\_timeout\_disable(I2C0);

### 函数 i2c\_clock\_timeout\_enable

函数i2c\_clock\_timeout\_enable描述见下表:

表 3-427. 函数 i2c\_clock\_timeout\_enable

|            |   |
|------------|---|
| 函数名称       | i2c_clock_timeout_enable                            |
| 函数原型       | void i2c_clock_timeout_enable(uint32_t i2c_periph); |
| 功能描述       | 禁能时钟信号延展超时检测  |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设  |
| I2Cx       | (x=0,1)   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如:

```
/* enable clock timeout detection */
```

```
i2c_clock_timeout_enable(I2C0);
```

### 函数 i2c\_clock\_timeout\_disable

函数i2c\_clock\_timeout\_disable描述见下表:

表 3-428. 函数 i2c\_clock\_timeout\_disable

|            |  |
|------------|--|
| 函数名称       | i2c_clock_timeout_disable                            |
| 函数原型       | void i2c_clock_timeout_disable(uint32_t i2c_periph); |
| 功能描述       | 禁能时钟超时检测   |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| i2c_periph | I2C 外设   |
| I2Cx       | (x=0,1)  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如:

```
/* disable clock timeout detection */
```

```
i2c_clock_timeout_disable(I2C0);
```

### 函数 i2c\_bus\_timeout\_b\_config

函数i2c\_bus\_timeout\_b\_config描述见下表：

表 3-429. 函数 i2c\_bus\_timeout\_b\_config

|            |   |
|------------|---|
| 函数名称       | i2c_bus_timeout_b_config  |
| 函数原型       | void i2c_bus_timeout_b_config(uint32_t i2c_periph, uint32_t timeout); |
| 功能描述       | 配置总线超时 B  |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设  |
| I2Cx       | (x=0,1)   |
| 输入参数{in}   |   |
| timeout    | 0-0xffff, 总线超时 B  |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如：

```
/* configure bus timeout B */
```

```
i2c_bus_timeout_b_config(I2C0, 0xff);
```

### 函数 i2c\_bus\_timeout\_a\_config

函数i2c\_bus\_timeout\_a\_config描述见下表：

表 3-430. 函数 i2c\_bus\_timeout\_a\_config

|            |   |
|------------|---|
| 函数名称       | i2c_bus_timeout_a_config  |
| 函数原型       | void i2c_bus_timeout_a_config(uint32_t i2c_periph, uint32_t timeout); |
| 功能描述       | 配置总线超时 A  |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| i2c_periph | I2C 外设  |
| I2Cx       | (x=0,1)   |
| 输入参数{in}   |   |
| timeout    | 0-0xffff, 总线超时 A  |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |

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

例如:

```
/* configure bus timeout A */
```

```
i2c_bus_timeout_a_config(I2C0, 0xff);
```

### 函数 i2c\_idle\_clock\_timeout\_config

函数i2c\_idle\_clock\_timeout\_config描述见下表:

**表 3-431. 函数 i2c\_idle\_clock\_timeout\_config**

|                       |  |
|-----------------------|--|
| 函数名称                  | i2c_idle_clock_timeout_config  |
| 函数原型                  | void i2c_idle_clock_timeout_config(uint32_t i2c_periph, uint32_t timeout); |
| 功能描述                  | 配置空闲时钟超时检测   |
| 先决条件                  | -  |
| 被调用函数                 | -  |
| 输入参数{in}              |  |
| i2c_periph            | I2C 外设   |
| I2Cx                  | (x=0,1)  |
| 输入参数{in}              |  |
| timeout               | 总线超时 A   |
| BUSTOA_DETECT_SCL_LOW | BUSTOA 用于检测 SCL 低电平超时  |
| BUSTOA_DETECT_IDLE    | BUSTOA 用于检测总线空闲情况下 SCL 和 SDA 高电平超时   |
| 输出参数{out}             |  |
| -                     | -  |
| 返回值                   |  |
| -                     | -  |

例如:

```
/* configure idle clock timeout detection */
```

```
i2c_idle_clock_timeout_config(I2C0, BUSTOA_DETECT_SCL_LOW);
```

### 函数 i2c\_flag\_get

函数i2c\_flag\_get描述见下表:

**表 3-432. 函数 i2c\_flag\_get**

|      |  |
|------|--|
| 函数名称 | i2c_flag_get   |
| 函数原型 | FlagStatus i2c_flag_get(uint32_t i2c_periph, uint32_t flag); |
| 功能描述 | 获取 I2C 标志位   |
| 先决条件 | -  |

|                  |                        |
|------------------|------------------------|
| 被调用函数            | -                      |
| 输入参数{in}         |                        |
| i2c_periph       | I2C 外设                 |
| I2Cx             | (x=0,1)                |
| 输入参数{in}         |                        |
| flag             | I2C 标志位                |
| I2C_FLAG_TBE     | 发送期间 I2C_TDATA 寄存器空标志  |
| I2C_FLAG_TI      | 发送中断标志                 |
| I2C_FLAG_RBNE    | 接收期间 I2C_RDATA 非空标志    |
| I2C_FLAG_ADDSEND | 从机模式下，接收到的地址与自身地址匹配    |
| I2C_FLAG_NACK    | NACK 标志                |
| I2C_FLAG_STPDET  | 从机模式下检测到 STOP 信号       |
| I2C_FLAG_TC      | 主机模式下传输完成标志            |
| I2C_FLAG_TCR     | 传输完成重载标志               |
| I2C_FLAG_BERR    | 总线错误标志                 |
| I2C_FLAG_LOSTARB | 仲裁丢失标志                 |
| I2C_FLAG_OUERR   | 从机模式下，过载/欠载错误标志        |
| I2C_FLAG_PECERR  | PEC 错误标志               |
| I2C_FLAG_TIMEOUT | 超时标志                   |
| I2C_FLAG_SMBALT  | SMBus 报警标志             |
| I2C_FLAG_I2CBSY  | 忙标志                    |
| I2C_FLAG_TR      | 从机模式下，I2C 作为发送器还是接收器标志 |
| 输出参数{out}        |                        |
| -                | -                      |
| 返回值              |                        |
| FlagStatus       | SET / RESET            |

例如：

```
/* get I2C flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_flag_get(I2C0, I2C_FLAG_TBE);
```

### 函数 i2c\_flag\_clear

函数i2c\_flag\_clear描述见下表：

表 3-433. 函数 i2c\_flag\_clear

|       |  |
|-------|--|
| 函数名称  | i2c_flag_clear   |
| 函数原型  | void i2c_flag_clear(uint32_t i2c_periph, uint32_t flag); |
| 功能描述  | 清除 I2C 标志位   |
| 先决条件  | -  |
| 被调用函数 | -  |



| 输入参数{in}                |                     |
|-------------------------|---------------------|
| <b>i2c_periph</b>       | I2C 外设              |
| <i>I2Cx</i>             | (x=0,1)             |
| 输入参数{in}                |                     |
| <b>flag</b>             | I2C 标志位             |
| <i>I2C_FLAG_ADDSEND</i> | 从机模式下，接收到的地址与自身地址匹配 |
| <i>I2C_FLAG_NACK</i>    | NACK 标志             |
| <i>I2C_FLAG_STPDET</i>  | 从机模式下检测到 STOP 信号    |
| <i>I2C_FLAG_BERR</i>    | 总线错误标志              |
| <i>I2C_FLAG_LOSTARB</i> | 仲裁丢失标志              |
| <i>I2C_FLAG_OUERR</i>   | 从机模式下，过载/欠载错误标志     |
| <i>I2C_FLAG_PECERR</i>  | PEC 错误标志            |
| <i>I2C_FLAG_TIMEOUT</i> | 超时标志                |
| <i>I2C_FLAG_SMBALT</i>  | SMBus 报警标志          |
| 输出参数{out}               |                     |
| -                       | -                   |
| 返回值                     |                     |
| -                       | -                   |

例如：

```
/* clear a bus error flag*/
```

```
i2c_flag_clear(I2C0, I2C_FLAG_BERR);
```

## 函数 i2c\_interrupt\_enable

函数i2c\_interrupt\_enable描述见下表：

表 3-434. 函数 i2c\_interrupt\_enable

| 函数名称                  | i2c_interrupt_enable  |
|-----------------------|---|
| 函数原型                  | void i2c_interrupt_enable(uint32_t i2c_periph, uint32_t interrupt); |
| 功能描述                  | 中断使能  |
| 先决条件                  | -   |
| 被调用函数                 | -   |
| 输入参数{in}              |   |
| <b>i2c_periph</b>     | I2C 外设  |
| <i>I2Cx</i>           | (x=0,1)   |
| 输入参数{in}              |   |
| <b>interrupt</b>      | I2C 中断  |
| <i>I2C_INT_ERR</i>    | 错误中断  |
| <i>I2C_INT_TC</i>     | 发送完成中断  |
| <i>I2C_INT_STPDET</i> | 检测到 STOP 中断   |
| <i>I2C_INT_NACK</i>   | 接收到 NACK 中断   |
| <i>I2C_INT_ADDM</i>   | 地址匹配中断  |

|                     |      |
|---------------------|------|
| <i>I2C_INT_RBNE</i> | 接收中断 |
| <i>I2C_INT_TI</i>   | 发送中断 |
| 输出参数{out}           |      |
| -                   | -    |
| 返回值                 |      |
| -                   | -    |

例如：

```
/* enable I2C0 transmit interrupt */
```

```
i2c_interrupt_enable(I2C0, I2C_INT_TI);
```

### 函数 i2c\_interrupt\_disable

函数i2c\_interrupt\_disable描述见下表：

**表 3-435. 函数 i2c\_interrupt\_disable**

|                       |  |
|-----------------------|--|
| 函数名称                  | i2c_interrupt_disable  |
| 函数原型                  | void i2c_interrupt_disable(uint32_t i2c_periph, uint32_t interrupt); |
| 功能描述                  | 中断除能   |
| 先决条件                  | -  |
| 被调用函数                 | -  |
| 输入参数{in}              |  |
| <b>i2c_periph</b>     | I2C 外设   |
| <i>I2Cx</i>           | (x=0,1)  |
| 输入参数{in}              |  |
| <b>interrupt</b>      | I2C 中断   |
| <i>I2C_INT_ERR</i>    | 错误中断   |
| <i>I2C_INT_TC</i>     | 发送完成中断   |
| <i>I2C_INT_STPDET</i> | 检测到 STOP 中断  |
| <i>I2C_INT_NACK</i>   | 接收到 NACK 中断  |
| <i>I2C_INT_ADDM</i>   | 地址匹配中断   |
| <i>I2C_INT_RBNE</i>   | 接收中断   |
| <i>I2C_INT_TI</i>     | 发送中断   |
| 输出参数{out}             |  |
| -                     | -  |
| 返回值                   |  |
| -                     | -  |

例如：

```
/* disable I2C0 transmit interrupt */
```

```
i2c_interrupt_disable(I2C0, I2C_INT_TI);
```

## 函数 i2c\_interrupt\_flag\_get

函数i2c\_interrupt\_flag\_get描述见下表:

表 3-436. 函数 i2c\_interrupt\_flag\_get

|                          |  |
|--------------------------|--|
| 函数名称                     | i2c_interrupt_flag_get   |
| 函数原型                     | FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph,<br>i2c_interrupt_flag_enum int_flag); |
| 功能描述                     | 获取中断标志位  |
| 先决条件                     | -  |
| 被调用函数                    | -  |
| 输入参数{in}                 |  |
| i2c_periph               | I2C 外设   |
| I2Cx                     | (x=0,1)  |
| 输入参数{in}                 |  |
| int_flag                 | I2C 中断标志位, 参考 <a href="#">表 3-376. 枚举类型 i2c_interrupt_flag_enum</a> 。                        |
| I2C_INT_FLAG_TI          | 发送中断标志   |
| I2C_INT_FLAG_RBNE        | 接收期间 I2C_RDATA 非空中断标志  |
| I2C_INT_FLAG_ADDS<br>END | 从机模式下, 接收到的地址与自身地址匹配中断标志   |
| I2C_INT_FLAG_NACK        | NACK 中断标志  |
| I2C_INT_FLAG_STPD<br>ET  | 从机模式下检测到 STOP 信号中断标志   |
| I2C_INT_FLAG_TC          | 主机模式下传输完成中断标志  |
| I2C_INT_FLAG_TCR         | 传输完成重载中断标志   |
| I2C_INT_FLAG_BERR        | 总线错误中断标志   |
| I2C_INT_FLAG_LOSTA<br>RB | 仲裁丢失中断标志   |
| I2C_INT_FLAG_OUER<br>R   | 从机模式下, 过载/欠载错误中断标志   |
| I2C_INT_FLAG_PEC<br>RR   | PEC 错误中断标志   |
| I2C_INT_FLAG_TIMEO<br>UT | 超时中断标志   |
| I2C_INT_FLAG_SMBA<br>LT  | SMBus 报警中断标志   |
| 输出参数{out}                |  |
| -                        | -  |
| 返回值                      |  |
| FlagStatus               | SET / RESET  |

例如:

```
/* get I2C interrupt flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_interrupt_flag_get(I2C0, I2C_INT_FLAG_TI);
```

### 函数 i2c\_interrupt\_flag\_clear

函数i2c\_interrupt\_flag\_clear描述见下表：

**表 3-437. 函数 i2c\_interrupt\_flag\_clear**

|                          |   |
|--------------------------|---|
| 函数名称                     | i2c_interrupt_flag_clear  |
| 函数原型                     | void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag); |
| 功能描述                     | 清除中断标志位   |
| 先决条件                     | -   |
| 被调用函数                    | -   |
| 输入参数{in}                 |   |
| i2c_periph               | I2C 外设  |
| I2Cx                     | (x=0,1)   |
| 输入参数{in}                 |   |
| int_flag                 | I2C 中断标志位，参考 <a href="#">表 3-376. 枚举类型 i2c_interrupt_flag_enum</a> 。                  |
| I2C_INT_FLAG_ADDS<br>END | 从机模式下，接收到的地址与自身地址匹配中断标志   |
| I2C_INT_FLAG_NACK        | NACK 中断标志   |
| I2C_INT_FLAG_STPD<br>ET  | 从机模式下检测到 STOP 信号中断标志  |
| I2C_INT_FLAG_BERR        | 总线错误中断标志  |
| I2C_INT_FLAG_LOSTA<br>RB | 仲裁丢失中断标志  |
| I2C_INT_FLAG_OUER<br>R   | 从机模式下，过载/欠载错误中断标志   |
| I2C_INT_FLAG_PEC<br>RR   | PEC 错误中断标志  |
| I2C_INT_FLAG_TIMEO<br>UT | 超时中断标志  |
| I2C_INT_FLAG_SMBA<br>LT  | SMBus 报警中断标志  |
| 输出参数{out}                |   |
| -                        | -   |
| 返回值                      |   |
| -                        | -   |

例如：

```
/* clear a bus error flag */
```

```
i2c_interrupt_flag_clear(I2C0, I2C_INT_FLAG_BERR);
```

## 3.15. MFCOM

MFCOM模块是一个高度可配置的模块，提供了多种串行通信协议的仿真和灵活的定时器配置。章节[3.15.1](#)描述了MFCOM的寄存器列表，章节[3.15.2](#)对MFCOM库函数进行说明。

### 3.15.1. 外设寄存器说明

MFCOM寄存器列表如下表所示：

**表 3-438. MFCOM 寄存器**

| 寄存器名称          | 寄存器描述           |
|----------------|-----------------|
| MFCOM_CTL      | 控制寄存器           |
| MFCOM_PINDATA  | 引脚数据寄存器         |
| MFCOM_SSTAT    | 移位器状态寄存器        |
| MFCOM_SERR     | 移位器错误寄存器        |
| MFCOM_TMSTAT   | 定时器状态寄存器        |
| MFCOM_SSIEN    | 移位器状态中断使能寄存器    |
| MFCOM_SEIEN    | 移位器错误中断使能寄存器    |
| MFCOM_TMSIEN   | 定时器状态中断使能寄存器    |
| MFCOM_SSDMAEN  | 移位器状态 DMA 使能寄存器 |
| MFCOM_SCTLx    | 移位器控制寄存器 x      |
| MFCOM_SCFGx    | 移位器配置寄存器 x      |
| MFCOM_SBUFx    | 移位缓冲区寄存器 x      |
| MFCOM_SBUFBISx | 移位缓冲区位交换寄存器 x   |
| MFCOM_SBUFBYSx | 移位缓冲区字节交换寄存器 x  |
| MFCOM_SBUFBBSx | 移位缓冲区位字节交换寄存器 x |
| MFCOM_TMCTLx   | 定时器控制寄存器 x      |
| MFCOM_TMCFGx   | 定时器配置寄存器 x      |
| MFCOM_TMCMPx   | 定时器比较寄存器 x      |

### 3.15.2. 外设库函数说明

MFCOM库函数列表如下表所示：

**表 3-439. MFCOM 库函数**

| 库函数名称                          | 库函数描述           |
|--------------------------------|-----------------|
| mfcom_deinit                   | 复位大多数 MFCOM 寄存器 |
| mfcom_software_reset           | 软复位 MFCOM       |
| mfcom_enable                   | 使能 MFCOM 功能     |
| mfcom_disable                  | 禁能 MFCOM 功能     |
| mfcom_timer_struct_para_init   | 使用默认参数值初始化定时器   |
| mfcom_shifter_struct_para_init | 使用默认参数值初始化移位器   |
| mfcom_timer_init               | 初始化定时器参数        |

| 库函数名称                                  | 库函数描述        |
|--|--------------|
| mfcom_shifter_init                     | 初始化移位器参数     |
| mfcom_timer_pin_config                 | 配置定时器引脚模式    |
| mfcom_shifter_pin_config               | 配置移位器引脚模式    |
| mfcom_timer_enable                     | 使能定时器在特定模式运行 |
| mfcom_shifter_enable                   | 使能移位器在特定模式运行 |
| mfcom_timer_disable                    | 禁能定时器        |
| mfcom_shifter_disable                  | 禁能移位器        |
| mfcom_timer_cmpvalue_set               | 设置定时器比较值     |
| mfcom_timer_cmpvalue_get               | 获取定时器比较值     |
| mfcom_timer_dismode_set                | 设置禁能定时器条件    |
| mfcom_shifter_stopbit_set              | 设置移位器停止位     |
| mfcom_buffer_write                     | 写移位器缓存区      |
| mfcom_buffer_read                      | 读移位器缓存区      |
| mfcom_shifter_flag_get                 | 获取移位器状态标志    |
| mfcom_shifter_error_flag_get           | 获取移位器错误状态标志  |
| mfcom_timer_flag_get                   | 获取定时器状态标志    |
| mfcom_shifter_interrupt_flag_get       | 获取移位器中断标志    |
| mfcom_shifter_error_interrupt_flag_get | 获取移位器错误中断标志  |
| mfcom_timer_interrupt_flag_get         | 获取定时器中断标志    |
| mfcom_shifter_flag_clear               | 清除移位器标志      |
| mfcom_shifter_error_flag_clear         | 清除移位器错误标志    |
| mfcom_timer_flag_clear                 | 清除定时器标志      |
| mfcom_shifter_interrupt_enable         | 使能移位器中断      |
| mfcom_shifter_error_interrupt_enable   | 使能移位器错误中断    |
| mfcom_timer_interrupt_enable           | 使能定时器中断      |
| mfcom_shifter_dma_enable               | 使能移位器 DMA    |
| mfcom_shifter_interrupt_disable        | 禁能移位器中断      |
| mfcom_shifter_error_interrupt_disable  | 禁能移位器错误中断    |
| mfcom_timer_interrupt_disable          | 禁能定时器中断      |
| mfcom_shifter_dma_disable              | 禁能移位器 DMA    |

### 结构体 mfcom\_timer\_parameter\_struct

表 3-440. 结构体 rtc\_parameter\_struct

| 成员名称             | 功能描述              |
|------------------|-------------------|
| trigger_select   | 触发信号选择            |
| trigger_polarity | 触发极性              |
| pin_config       | 定时器引脚配置           |
| pin_select       | 定时器引脚选择           |
| pin_polarity     | 定时器引脚极性           |
| mode             | 定时器工作模式           |
| output           | 定时器初始输出状态及是否受复位影响 |

|           |             |
|-----------|-------------|
| decrement | 定时器递减的参考时钟源 |
| reset     | 定时器的计时器复位条件 |
| disable   | 禁能定时器的条件    |
| enable    | 使能定时器的条件    |
| stopbit   | 定时器停止位      |
| startbit  | 定时器起始位      |
| compare   | 定时器比较寄存器x的值 |

### 结构体 `mfcom_shifter_parameter_struct`

表 3-441. 结构体 `mfcom_shifter_parameter_struct`

| 成员名称           | 功能描述            |
|----------------|-----------------|
| timer_select   | 选择定时器产生时钟及控制移位器 |
| timer_polarity | 定时器极性           |
| pin_config     | 移位器引脚配置         |
| pin_select     | 移位器引脚选择         |
| pin_polarity   | 移位器引脚极性         |
| mode           | 移位器工作模式         |
| input_source   | 移位器输入源          |
| stopbit        | 移位器停止位          |
| startbit       | 移位器起始位          |

### 函数 `mfcom_deinit`

函数`mfcom_deinit`描述见下表:

表 3-442. 函数 `mfcom_deinit`

| 函数名称      | <code>mfcom_deinit</code>  |
|-----------|--|
| 函数原型      | <code>void mfcom_deinit(void);</code>  |
| 功能描述      | 复位大多数MFCOM寄存器  |
| 先决条件      | -  |
| 被调用函数     | <code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code> |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如:

```
/* reset most of the MFCOM registers */
```

```
mfcom_deinit();
```

## 函数 mfcom\_software\_reset

函数mfcom\_software\_reset描述见下表：

**表 3-443. 函数 mfcom\_software\_reset**

|           |                                  |
|-----------|----------------------------------|
| 函数名称      | mfcom_software_reset             |
| 函数原型      | void mfcom_software_reset(void); |
| 功能描述      | 软复位MFCOM                         |
| 先决条件      | -                                |
| 被调用函数     | -                                |
| 输入参数{in}  |                                  |
| -         | -                                |
| 输出参数{out} |                                  |
| -         | -                                |
| 返回值       |                                  |
| -         | -                                |

例如：

```
/* software reset */
mfcom_software_reset();
```

## 函数 mfcom\_enable

函数mfcom\_enable描述见下表：

**表 3-444. 函数 mfcom\_enable**

|           |                          |
|-----------|--------------------------|
| 函数名称      | mfcom_enable             |
| 函数原型      | void mfcom_enable(void); |
| 功能描述      | 使能MFCOM功能                |
| 先决条件      | -                        |
| 被调用函数     | -                        |
| 输入参数{in}  |                          |
| -         | -                        |
| 输出参数{out} |                          |
| -         | -                        |
| 返回值       |                          |
| -         | -                        |

例如：

```
/* enable MFCOM function */
mfcom_enable();
```



## 函数 mfcom\_disable

函数mfcom\_disable描述见下表:

表 3-445. 函数 mfcom\_disable

|           |                           |
|-----------|---------------------------|
| 函数名称      | mfcom_disable             |
| 函数原型      | void mfcom_disable(void); |
| 功能描述      | 禁能MFCOM功能                 |
| 先决条件      | -                         |
| 被调用函数     | -                         |
| 输入参数{in}  |                           |
| -         | -                         |
| 输出参数{out} |                           |
| -         | -                         |
| 返回值       |                           |
| -         | -                         |

例如:

```
/* disable MFCOM function */
```

```
mfcom_disable();
```

## 函数 mfcom\_timer\_struct\_para\_init

函数mfcom\_timer\_struct\_para\_init描述见下表:

表 3-446. 函数 mfcom\_timer\_struct\_para\_init

|             |  |
|-------------|--|
| 函数名称        | mfcom_timer_struct_para_init   |
| 函数原型        | void mfcom_timer_struct_para_init(mfcom_timer_parameter_struct*<br>init_struct); |
| 功能描述        | 使用默认参数值初始化定时器  |
| 先决条件        | -  |
| 被调用函数       | -  |
| 输入参数{in}    |  |
| init_struct | 初始化结构体, 结构体成员参考 <a href="#">结构体mfcom_timer_parameter_struct</a>                  |
| 输出参数{out}   |  |
| -           | -  |
| 返回值         |  |
| -           | -  |

例如:

```
/* initialize mfcom_timer_parameter_struct with the default values */
```

```
mfcom_timer_parameter_struct mfcom_timer_struct
```

```
mfcom_timer_struct_para_init(&mfcom_timer_struct);
```

## 函数 mfcom\_shifter\_struct\_para\_init

函数mfcom\_shifter\_struct\_para\_init描述见下表：

表 3-447. 函数 mfcom\_shifter\_struct\_para\_init

|             |   |
|-------------|---|
| 函数名称        | mfcom_shifter_struct_para_init  |
| 函数原型        | void mfcom_shifter_struct_para_init(mfcom_shifter_parameter_struct* init_struct); |
| 功能描述        | 使用默认参数值初始化移位器   |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| init_struct | 初始化结构体，结构体成员参考 <a href="#">结构体mfcom_shifter_parameter_struct</a>                  |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如：

```
/* initialize mfcom_shifter_parameter_struct with the default values */
```

```
mfcom_shifter_parameter_struct mfcom_shifter_struct
```

```
mfcom_shifter_struct_para_init(&mfcom_shifter_struct);
```

## 函数 mfcom\_timer\_init

函数mfcom\_timer\_init描述见下表：

表 3-448. 函数 mfcom\_timer\_init

|               |  |
|---------------|--|
| 函数名称          | mfcom_timer_init   |
| 函数原型          | void mfcom_timer_init(uint32_t timernum, mfcom_timer_parameter_struct* init_struct); |
| 功能描述          | 初始化定时器参数   |
| 先决条件          | -  |
| 被调用函数         | -  |
| 输入参数{in}      |  |
| timernum      | 定时器序号  |
| MFCOM_TIMER_x | x = 0...3  |
| 输入参数{in}      |  |
| init_struct   | 初始化结构体，结构体成员参考 <a href="#">结构体mfcom_timer_parameter_struct</a>                       |
| 输出参数{out}     |  |
| -             | -  |
| 返回值           |  |
| -             | -  |

例如:

```
/* initialize MFCOM timer parameter */
mfcom_timer_parameter_struct init_struct;

init_struct.trigger_select      = MFCOM_TIMER_TRGSEL_PIN0;
init_struct.trigger_polarity    = MFCOM_TIMER_TRGPOL_ACTIVE_HIGH;
init_struct.pin_config          = MFCOM_TIMER_PINCFG_INPUT;
init_struct.pin_select          = MFCOM_TIMER_PINSEL_PIN0;
init_struct.pin_polarity        = MFCOM_TIMER_PINPOL_ACTIVE_HIGH;
init_struct.mode                = MFCOM_TIMER_BAUDMODE;
init_struct.output              = MFCOM_TIMER_OUT_HIGH_EN_RESET;
init_struct.decrement           = MFCOM_TIMER_DEC_CLK_SHIFT_OUT;
init_struct.reset               = MFCOM_TIMER_RESET_TRIG_TIMEOUT;
init_struct.disable             = MFCOM_TIMER_DISMODE_PINBOTH;
init_struct.enable              = MFCOM_TIMER_ENMODE_TRIGHIGH;
init_struct.stopbit             = MFCOM_TIMER_STOPBIT_TIMDIS;
init_struct.startbit            = MFCOM_TIMER_STARTBIT_ENABLE;

mfcom_timer_init(MFCOM_TIMER_0, &init_struct);
```

### 函数 mfcom\_shifter\_init

函数mfcom\_shifter\_init描述见下表:

表 3-449. 函数 mfcom\_shifter\_init

|                 |   |
|-----------------|---|
| 函数名称            | mfcom_shifter_init  |
| 函数原型            | void mfcom_shifter_init(uint32_t shifternum,<br>mfcom_shifter_parameter_struct* init_struct); |
| 功能描述            | 初始化移位器参数  |
| 先决条件            | -   |
| 被调用函数           | -   |
| 输入参数{in}        |   |
| shifternum      | 移位器序号   |
| MFCOM_SHIFTER_x | x = 0...3   |
| 输入参数{in}        |   |
| init_struct     | 初始化结构体, 结构体成员参考 <a href="#">结构体mfcom_shifter_parameter_struct</a>                             |
| 输出参数{out}       |   |
| -               | -   |

| 返回值 |   |
|-----|---|
| -   | - |

例如：

```
/* initialize MFCOM shifter parameter */
```

```
mfcom_shifter_parameter_struct init_struct;
```

```
init_struct.timer_select          = MFCOM_SHIFTER_TIMER0;
```

```
init_struct.timer_polarity       = MFCOM_SHIFTER_TIMPOL_ACTIVE_HIGH;
```

```
init_struct.pin_config           = MFCOM_SHIFTER_PINCFG_INPUT;
```

```
init_struct.pin_select           = MFCOM_SHIFTER_PINSEL_PIN0;
```

```
init_struct.pin_polarity         = MFCOM_SHIFTER_PINPOL_ACTIVE_HIGH;
```

```
init_struct.mode                 = MFCOM_SHIFTER_TRANSMIT;
```

```
init_struct.input_source         = MFCOM_SHIFTER_INSRC_PIN;
```

```
init_struct.stopbit              = MFCOM_SHIFTER_STOPBIT_HIGH;
```

```
init_struct.startbit             = MFCOM_SHIFTER_STARTBIT_LOW;
```

```
mfcom_timer_init(MFCOM_SHIFTER_0, &init_struct);
```

### 函数 mfcom\_timer\_pin\_config

函数mfcom\_timer\_pin\_config描述见下表：

表 3-450. 函数 mfcom\_timer\_pin\_config

| 函数名称                         | mfcom_timer_pin_config   |
|------------------------------|--|
| 函数原型                         | void mfcom_timer_pin_config(uint32_t timernum, uint32_t mode); |
| 功能描述                         | 配置定时器引脚模式  |
| 先决条件                         | -  |
| 被调用函数                        | -  |
| 输入参数{in}                     |  |
| timernum                     | 定时器序号  |
| MFCOM_TIMER_x                | x = 0...3  |
| 输入参数{in}                     |  |
| mode                         | 定时器引脚模式  |
| MFCOM_TIMER_PINCFG_INPUT     | 引脚输入模式   |
| MFCOM_TIMER_PINCFG_OPENDRAIN | 引脚开漏模式   |
| MFCOM_TIMER_PINCFG_BIDI      | 级联引脚输入/输出数据模式  |

|                                       |        |
|---------------------------------------|--------|
| <i>MFCOM_TIMER_PINC<br/>FG_OUTPUT</i> | 引脚输出模式 |
| 输出参数{out}                             |        |
| -                                     | -      |
| 返回值                                   |        |
| -                                     | -      |

例如:

```
/* configure timer pin mode */
```

```
mfcom_timer_pin_config(MFCOM_TIMER_0, MFCOM_TIMER_PINCFG_OPENDRAIN);
```

### 函数 mfcom\_shifter\_pin\_config

函数mfcom\_shifter\_pin\_config描述见下表:

**表 3-451. 函数 mfcom\_shifter\_pin\_config**

|   |  |
|---|--|
| 函数名称                                      | mfcom_shifter_pin_config   |
| 函数原型                                      | void mfcom_shifter_pin_config(uint32_t shifternum, uint32_t mode); |
| 功能描述                                      | 配置移位器引脚模式  |
| 先决条件                                      | -  |
| 被调用函数                                     | -  |
| 输入参数{in}                                  |  |
| <b>shifternum</b>                         | 移位器序号  |
| <i>MFCOM_TIMER_x</i>                      | x = 0...3  |
| 输入参数{in}                                  |  |
| <b>mode</b>                               | 移位器引脚模式  |
| <i>MFCOM_SHIFTER_P<br/>NCFG_INPUT</i>     | 引脚输入模式   |
| <i>MFCOM_SHIFTER_P<br/>NCFG_OPENDRAIN</i> | 引脚开漏模式   |
| <i>MFCOM_SHIFTER_P<br/>NCFG_BIDI</i>      | 级联引脚输入/输出数据模式  |
| <i>MFCOM_SHIFTER_P<br/>NCFG_OUTPUT</i>    | 引脚输出模式   |
| 输出参数{out}                                 |  |
| -   | -  |
| 返回值                                       |  |
| -   | -  |

例如:

```
/* configure shifter pin mode */
```

```
mfcom_shifter_pin_config(MFCOM_SHIFTER_0, MFCOM_SHIFTER_PINCFG_BIDI);
```

函数 **mfcom\_timer\_enable**

函数mfcom\_timer\_enable描述见下表:

表 3-452. 函数 **mfcom\_timer\_enable**

|                          |   |
|--------------------------|---|
| 函数名称                     | mfcom_timer_enable  |
| 函数原型                     | void mfcom_timer_enable(uint32_t timernum, uint32_t timermode); |
| 功能描述                     | 使能定时器在特定模式运行  |
| 先决条件                     | -   |
| 被调用函数                    | -   |
| 输入参数{in}                 |   |
| timernum                 | 定时器序号   |
| MFCOM_TIMER_x            | x = 0...3   |
| 输入参数{in}                 |   |
| timermode                | 定时器工作模式   |
| MFCOM_TIMER_DISABLE      | 禁能定时器   |
| MFCOM_TIMER_BAUDMODE     | 双8位波特计数模式   |
| MFCOM_TIMER_PWMMODE      | 双8位PWM模式  |
| MFCOM_TIMER_16BITCOUNTER | 16位计数模式   |
| 输出参数{out}                |   |
| -                        | -   |
| 返回值                      |   |
| -                        | -   |

例如:

```
/* enable MFCOM timer in specific mode */
```

```
mfcom_timer_enable(MFCOM_TIMER_0, MFCOM_TIMER_BAUDMODE);
```

函数 **mfcom\_shifter\_enable**

函数mfcom\_shifter\_enable描述见下表:

表 3-453. 函数 **mfcom\_shifter\_enable**

|            |   |
|------------|---|
| 函数名称       | mfcom_shifter_enable  |
| 函数原型       | void mfcom_shifter_enable(uint32_t shifternum, uint32_t shiftermode); |
| 功能描述       | 使能移位器在特定模式运行  |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| shifternum | 移位器序号   |

|                                |           |
|--------------------------------|-----------|
| MFCOM_SHIFTER_x                | x = 0...3 |
| 输入参数{in}                       |           |
| timermode                      | 定时器工作模式   |
| MFCOM_SHIFTER_DISABLE          | 禁能移位器     |
| MFCOM_SHIFTER_RECEIVE          | 接收模式      |
| MFCOM_SHIFTER_TRANSMIT         | 发送模式      |
| MFCOM_SHIFTER_MATCH_STORE      | 匹配存储模式    |
| MFCOM_SHIFTER_MATCH_CONTINUOUS | 持续匹配模式    |
| 输出参数{out}                      |           |
| -                              | -         |
| 返回值                            |           |
| -                              | -         |

例如：

```
/* enable MFCOM shifter in specific mode */
```

```
mfcom_shifter_enable(MFCOM_SHIFTER_0, MFCOM_SHIFTER_RECEIVE);
```

### 函数 mfcom\_timer\_disable

函数mfcom\_timer\_disable描述见下表：

表 3-454. 函数 mfcom\_timer\_disable

|               |  |
|---------------|--|
| 函数名称          | mfcom_timer_disable                          |
| 函数原型          | void mfcom_timer_disable(uint32_t timernum); |
| 功能描述          | 禁能定时器  |
| 先决条件          | -  |
| 被调用函数         | -  |
| 输入参数{in}      |  |
| timernum      | 定时器序号  |
| MFCOM_TIMER_x | x = 0...3                                    |
| 输出参数{out}     |  |
| -             | -  |
| 返回值           |  |
| -             | -  |

例如：

```
/* disable MFCOM timer */
```

```
mfcom_timer_disable(MFCOM_TIMER_0);
```

## 函数 mfcom\_shifter\_disable

函数mfcom\_shifter\_disable描述见下表:

表 3-455. 函数 mfcom\_shifter\_disable

|                 |  |
|-----------------|--|
| 函数名称            | mfcom_shifter_disable                            |
| 函数原型            | void mfcom_shifter_disable(uint32_t shifternum); |
| 功能描述            | 禁能移位器  |
| 先决条件            | -  |
| 被调用函数           | -  |
| 输入参数{in}        |  |
| shifternum      | 移位器序号  |
| MFCOM_SHIFTER_x | x = 0...3  |
| 输出参数{out}       |  |
| -               | -  |
| 返回值             |  |
| -               | -  |

例如:

```
/* disable MFCOM shifter */
mfcom_shifter_disable(MFCOM_SHIFTER_0);
```

## 函数 mfcom\_timer\_cmpvalue\_set

函数mfcom\_timer\_cmpvalue\_set描述见下表:

表 3-456. 函数 mfcom\_timer\_cmpvalue\_set

|               |   |
|---------------|---|
| 函数名称          | mfcom_timer_cmpvalue_set  |
| 函数原型          | void mfcom_timer_cmpvalue_set(uint32_t timernum, uint32_t compare); |
| 功能描述          | 设置定时器比较值  |
| 先决条件          | -   |
| 被调用函数         | -   |
| 输入参数{in}      |   |
| timernum      | 定时器序号   |
| MFCOM_TIMER_x | x = 0...3   |
| 输入参数{in}      |   |
| compare       | 比较值   |
| 输出参数{out}     |   |
| -             | -   |
| 返回值           |   |
| -             | -   |

例如:

```
/* set MFCOM timer compare value */
```



```
mfcom_timer_cmpvalue_set(MFCOM_TIMER_0, 0x0A0A);
```

### 函数 mfcom\_timer\_cmpvalue\_get

函数mfcom\_timer\_cmpvalue\_get描述见下表：

表 3-457. 函数 mfcom\_timer\_cmpvalue\_get

|               |   |
|---------------|---|
| 函数名称          | mfcom_timer_cmpvalue_get                              |
| 函数原型          | uint32_t mfcom_timer_cmpvalue_get(uint32_t timernum); |
| 功能描述          | 获取定时器比较值  |
| 先决条件          | -   |
| 被调用函数         | -   |
| 输入参数{in}      |   |
| timernum      | 定时器序号   |
| MFCOM_TIMER_x | x = 0...3   |
| 输出参数{out}     |   |
| -             | -   |
| 返回值           |   |
| uint32_t      | cmpvalue  |

例如：

```
/* get MFCOM timer compare value */
```

```
uint32_t value = 0;
```

```
value = mfcom_timer_cmpvalue_get(MFCOM_TIMER_0);
```

### 函数 mfcom\_timer\_dismode\_set

函数mfcom\_timer\_dismode\_set描述见下表：

表 3-458. 函数 mfcom\_timer\_dismode\_set

|                               |  |
|-------------------------------|--|
| 函数名称                          | mfcom_timer_dismode_set  |
| 函数原型                          | void mfcom_timer_dismode_set(uint32_t timernum, uint32_t dismode); |
| 功能描述                          | 设置禁能定时器条件  |
| 先决条件                          | -  |
| 被调用函数                         | -  |
| 输入参数{in}                      |  |
| timernum                      | 定时器序号  |
| MFCOM_TIMER_x                 | x = 0...3  |
| 输入参数{in}                      |  |
| dismode                       | 禁能定时器的条件   |
| MFCOM_TIMER_DISM<br>ODE_NEVER | 永不禁能   |
| MFCOM_TIMER_DISM              | 前一个定时器禁能时禁能  |

|  |                  |
|--|------------------|
| ODE_PRE_TIMDIS                               |                  |
| MFCOM_TIMER_DISM<br>ODE_COMPARE              | 发生比较事件时触发        |
| MFCOM_TIMER_DISM<br>ODE_COMPARE_TRI<br>GLOW  | 发生比较事件且触发为低电平时禁能 |
| MFCOM_TIMER_DISM<br>ODE_PINBOTH              | 引脚上升下降沿时禁能       |
| MFCOM_TIMER_DISM<br>ODE_PINBOTH_TRIG<br>HIGH | 引脚上升下降沿或触发高电平时禁能 |
| MFCOM_TIMER_DISM<br>ODE_TRIGFALLING          | 触发低电平时禁能         |
| 输出参数{out}                                    |                  |
| -  | -                |
| 返回值  |                  |
| -  | -                |

例如：

```
/* set MFCOM timer disable mode */
```

```
mfcom_timer_dismode_set(MFCOM_TIMER_0, MFCOM_TIMER_DISMODE_COMPARE);
```

### 函数 mfcom\_shifter\_stopbit\_set

函数mfcom\_shifter\_stopbit\_set描述见下表：

表 3-459. 函数 mfcom\_shifter\_stopbit\_set

|                                   |  |
|-----------------------------------|--|
| 函数名称                              | mfcom_shifter_stopbit_set  |
| 函数原型                              | void mfcom_shifter_stopbit_set(uint32_t shifternum, uint32_t stopbit); |
| 功能描述                              | 设置移位器停止位   |
| 先决条件                              | -  |
| 被调用函数                             | -  |
| 输入参数{in}                          |  |
| shifternum                        | 移位器序号  |
| MFCOM_SHIFTER_x                   | x = 0...3  |
| 输入参数{in}                          |  |
| stopbit                           | 停止位  |
| MFCOM_SHIFTER_ST<br>OPBIT_DISABLE | 禁能移位器停止位   |
| MFCOM_SHIFTER_ST<br>OPBIT_LOW     | 设置停止位为0  |
| MFCOM_SHIFTER_ST<br>OPBIT_HIGH    | 设置停止位为1  |

| 输出参数{out} |   |
|-----------|---|
| -         | - |
| 返回值       |   |
| -         | - |

例如:

```
/* set MFCOM shifter stopbit */
```

```
mfcom_shifter_stopbit_set(MFCOM_SHIFTER_0, MFCOM_SHIFTER_STOPBIT_LOW);
```

### 函数 mfcom\_buffer\_write

函数mfcom\_buffer\_write描述见下表:

表 3-460. 函数 mfcom\_buffer\_write

| 函数名称                     | mfcom_buffer_write  |
|--------------------------|---|
| 函数原型                     | void mfcom_buffer_write(uint32_t shifternum, uint32_t data, uint32_t rwmode); |
| 功能描述                     | 写移位器缓存区   |
| 先决条件                     | -   |
| 被调用函数                    | -   |
| 输入参数{in}                 |   |
| shifternum               | 移位器序号   |
| MFCOM_SHIFTER_x          | x = 0...3   |
| 输入参数{in}                 |   |
| data                     | 待写入数据   |
| 输入参数{in}                 |   |
| rwmode                   | MFCOM读写模式   |
| MFCOM_RWMODE_NORMAL      | 正常读写模式  |
| MFCOM_RWMODE_BITSWAP     | 位交换模式   |
| MFCOM_RWMODE_BYTESWAP    | 字节交换模式  |
| MFCOM_RWMODE_BITBYTESWAP | 位字节交换模式   |
| 输出参数{out}                |   |
| -                        | -   |
| 返回值                      |   |
| -                        | -   |

例如:

```
/* write MFCOM shift buffer */
```

```
mfcom_buffer_write(MFCOM_SHIFTER_0, 0x6699, MFCOM_RWMODE_BITSWAP);
```

**函数 mfcom\_buffer\_read**

函数mfcom\_buffer\_read描述见下表：

**表 3-461. 函数 mfcom\_buffer\_read**

|                            |   |
|----------------------------|---|
| 函数名称                       | mfcom_buffer_read   |
| 函数原型                       | uint32_t mfcom_buffer_read(uint32_t shifternum, uint32_t rwmode); |
| 功能描述                       | 读移位器缓存区   |
| 先决条件                       | -   |
| 被调用函数                      | -   |
| 输入参数{in}                   |   |
| shifternum                 | 移位器序号   |
| MFCOM_SHIFTER_x            | x = 0...3   |
| 输入参数{in}                   |   |
| rwmode                     | MFCOM读写模式   |
| MFCOM_RWMODE_NORMAL        | 正常读写模式  |
| MFCOM_RWMODE_BITSWAP       | 位交换模式   |
| MFCOM_RWMODE_BYTE_SWAP     | 字节交换模式  |
| MFCOM_RWMODE_BIT_BYTE_SWAP | 位字节交换模式   |
| 输出参数{out}                  |   |
| -                          | -   |
| 返回值                        |   |
| uint32_t                   | data  |

例如：

```
/* read MFCOM shift buffer */
```

```
uint32_t data = 0;
```

```
data = mfcom_buffer_read(MFCOM_SHIFTER_0, MFCOM_RWMODE_NORMAL);
```

**函数 mfcom\_shifter\_flag\_get**

函数mfcom\_shifter\_flag\_get描述见下表：

**表 3-462. 函数 mfcom\_shifter\_flag\_get**

|       |  |
|-------|--|
| 函数名称  | mfcom_shifter_flag_get                               |
| 函数原型  | FlagStatus mfcom_shifter_flag_get(uint32_t shifter); |
| 功能描述  | 获取移位器状态标志  |
| 先决条件  | -  |
| 被调用函数 | -  |

| 输入参数{in}        |              |
|-----------------|--------------|
| shifter         | 移位器序号        |
| MFCOM_SHIFTER_x | x = 0...3    |
| 输出参数{out}       |              |
| -               | -            |
| 返回值             |              |
| FlagStatus      | SET or RESET |

例如：

```
/* get MFCOM shifter flag */
```

```
flag = mfcom_shifter_flag_get(MFCOM_SHIFTER_0);
```

### 函数 mfcom\_shifter\_error\_flag\_get

函数mfcom\_shifter\_error\_flag\_get描述见下表：

表 3-463. 函数 mfcom\_shifter\_error\_flag\_get

| 函数名称            | mfcom_shifter_error_flag_get                               |
|-----------------|--|
| 函数原型            | FlagStatus mfcom_shifter_error_flag_get(uint32_t shifter); |
| 功能描述            | 获取移位器错误状态标志  |
| 先决条件            | -  |
| 被调用函数           | -  |
| 输入参数{in}        |  |
| shifter         | 移位器序号  |
| MFCOM_SHIFTER_x | x = 0...3  |
| 输出参数{out}       |  |
| -               | -  |
| 返回值             |  |
| FlagStatus      | SET or RESET   |

例如：

```
/* get MFCOM shifter error flag */
```

```
flag = mfcom_shifter_error_flag_get(MFCOM_SHIFTER_0);
```

### 函数 mfcom\_timer\_flag\_get

函数mfcom\_timer\_flag\_get描述见下表：

表 3-464. 函数 mfcom\_timer\_flag\_get

|      |  |
|------|--|
| 函数名称 | mfcom_timer_flag_get                             |
| 函数原型 | FlagStatus mfcom_timer_flag_get(uint32_t timer); |
| 功能描述 | 获取定时器状态标志  |
| 先决条件 | -  |

|               |              |
|---------------|--------------|
| 被调用函数         | -            |
| 输入参数{in}      |              |
| timer         | 定时器序号        |
| MFCOM_TIMER_x | x = 0...3    |
| 输出参数{out}     |              |
| -             | -            |
| 返回值           |              |
| FlagStatus    | SET or RESET |

例如：

```
/* get MFCOM timer flag */
```

```
flag = mfcom_timer_flag_get(MFCOM_TIMER_0);
```

### 函数 mfcom\_shifter\_interrupt\_flag\_get

函数mfcom\_shifter\_interrupt\_flag\_get描述见下表：

表 3-465. 函数 mfcom\_shifter\_interrupt\_flag\_get

|                 |  |
|-----------------|--|
| 函数名称            | mfcom_shifter_interrupt_flag_get                               |
| 函数原型            | FlagStatus mfcom_shifter_interrupt_flag_get(uint32_t shifter); |
| 功能描述            | 获取移位器中断标志  |
| 先决条件            | -  |
| 被调用函数           | -  |
| 输入参数{in}        |  |
| shifter         | 移位器序号  |
| MFCOM_SHIFTER_x | x = 0...3  |
| 输出参数{out}       |  |
| -               | -  |
| 返回值             |  |
| FlagStatus      | SET or RESET   |

例如：

```
/* get MFCOM shifter interrupt flag */
```

```
flag = mfcom_shifter_interrupt_flag_get(MFCOM_SHIFTER_0);
```

### 函数 mfcom\_shifter\_error\_interrupt\_flag\_get

函数mfcom\_shifter\_error\_interrupt\_flag\_get描述见下表：

表 3-466. 函数 mfcom\_shifter\_error\_interrupt\_flag\_get

|      |  |
|------|--|
| 函数名称 | mfcom_shifter_error_interrupt_flag_get                               |
| 函数原型 | FlagStatus mfcom_shifter_error_interrupt_flag_get(uint32_t shifter); |
| 功能描述 | 获取移位器错误中断标志  |

|                 |              |
|-----------------|--------------|
| 先决条件            | -            |
| 被调用函数           | -            |
| 输入参数{in}        |              |
| shifter         | 移位器序号        |
| MFCOM_SHIFTER_x | x = 0...3    |
| 输出参数{out}       |              |
| -               | -            |
| 返回值             |              |
| FlagStatus      | SET or RESET |

例如：

```
/* get MFCOM shifter error interrupt flag */
```

```
flag = mfcom_shifter_error_interrupt_flag_get (MFCOM_SHIFTER_0);
```

#### 函数 mfcom\_timer\_interrupt\_flag\_get

函数mfcom\_timer\_interrupt\_flag\_get描述见下表：

**表 3-467. 函数 mfcom\_timer\_interrupt\_flag\_get**

|               |  |
|---------------|--|
| 函数名称          | mfcom_timer_interrupt_flag_get                             |
| 函数原型          | FlagStatus mfcom_timer_interrupt_flag_get(uint32_t timer); |
| 功能描述          | 获取定时器中断标志  |
| 先决条件          | -  |
| 被调用函数         | -  |
| 输入参数{in}      |  |
| timer         | 定时器序号  |
| MFCOM_TIMER_x | x = 0...3  |
| 输出参数{out}     |  |
| -             | -  |
| 返回值           |  |
| FlagStatus    | SET or RESET   |

例如：

```
/* get MFCOM timer interrupt flag */
```

```
flag = mfcom_timer_interrupt_flag_get (MFCOM_TIMER_0);
```

#### 函数 mfcom\_shifter\_flag\_clear

函数mfcom\_shifter\_flag\_clear描述见下表：

**表 3-468. 函数 mfcom\_shifter\_flag\_clear**

|      |  |
|------|--|
| 函数名称 | mfcom_shifter_flag_clear                         |
| 函数原型 | void mfcom_shifter_flag_clear(uint32_t shifter); |

|                 |           |
|-----------------|-----------|
| 功能描述            | 清除移位器标志   |
| 先决条件            | -         |
| 被调用函数           | -         |
| 输入参数{in}        |           |
| shifter         | 移位器序号     |
| MFCOM_SHIFTER_x | x = 0...3 |
| 输出参数{out}       |           |
| -               | -         |
| 返回值             |           |
| -               | -         |

例如：

```
/* clear MFCOM shifter flag */
```

```
flag = mfcom_shifter_flag_clear(MFCOM_SHIFTER_0);
```

### 函数 mfcom\_shifter\_error\_flag\_clear

函数mfcom\_shifter\_error\_flag\_clear描述见下表：

表 3-469. 函数 mfcom\_shifter\_error\_flag\_clear

|                 |   |
|-----------------|---|
| 函数名称            | mfcom_shifter_error_flag_clear                          |
| 函数原型            | void mfcom_shifter_error_flag_clear (uint32_t shifter); |
| 功能描述            | 清除移位器错误标志   |
| 先决条件            | -   |
| 被调用函数           | -   |
| 输入参数{in}        |   |
| shifter         | 移位器序号   |
| MFCOM_SHIFTER_x | x = 0...3   |
| 输出参数{out}       |   |
| -               | -   |
| 返回值             |   |
| -               | -   |

例如：

```
/* clear MFCOM shifter flag */
```

```
flag = mfcom_shifter_flag_clear(MFCOM_SHIFTER_0);
```

### 函数 mfcom\_timer\_flag\_clear

函数mfcom\_timer\_flag\_clear描述见下表：

表 3-470. 函数 mfcom\_timer\_flag\_clear

|      |                        |
|------|------------------------|
| 函数名称 | mfcom_timer_flag_clear |
|------|------------------------|



|               |  |
|---------------|--|
| 函数原型          | void mfcom_timer_flag_clear(uint32_t timer); |
| 功能描述          | 清除定时器标志                                      |
| 先决条件          | -  |
| 被调用函数         | -  |
| 输入参数{in}      |  |
| timer         | 定时器序号  |
| MFCOM_TIMER_x | x = 0...3                                    |
| 输出参数{out}     |  |
| -             | -  |
| 返回值           |  |
| -             | -  |

例如：

```
/* clear MFCOM timer flag */
mfcom_timer_flag_clear (MFCOM_TIMER_0);
```

### 函数 mfcom\_shifter\_interrupt\_enable

函数mfcom\_shifter\_interrupt\_enable描述见下表：

表 3-471. 函数 mfcom\_shifter\_interrupt\_enable

|                 |   |
|-----------------|---|
| 函数名称            | mfcom_shifter_interrupt_enable                          |
| 函数原型            | void mfcom_shifter_interrupt_enable (uint32_t shifter); |
| 功能描述            | 使能移位器中断   |
| 先决条件            | -   |
| 被调用函数           | -   |
| 输入参数{in}        |   |
| shifter         | 移位器序号   |
| MFCOM_SHIFTER_x | x = 0...3   |
| 输出参数{out}       |   |
| -               | -   |
| 返回值             |   |
| -               | -   |

例如：

```
/* enable MFCOM shifter interrupt */
mfcom_shifter_interrupt_enable (MFCOM_SHIFTER_0);
```

### 函数 mfcom\_shifter\_error\_interrupt\_enable

函数mfcom\_shifter\_error\_interrupt\_enable描述见下表：

表 3-472. 函数 `mfcom_shifter_error_interrupt_enable`

|                              |  |
|------------------------------|--|
| 函数名称                         | <code>mfcom_shifter_error_interrupt_enable</code>                          |
| 函数原型                         | <code>void mfcom_shifter_error_interrupt_enable (uint32_t shifter);</code> |
| 功能描述                         | 使能移位器错误中断  |
| 先决条件                         | -  |
| 被调用函数                        | -  |
| 输入参数{in}                     |  |
| <b>shifter</b>               | 移位器序号  |
| <code>MFCOM_SHIFTER_x</code> | <code>x = 0...3</code>   |
| 输出参数{out}                    |  |
| -                            | -  |
| 返回值                          |  |
| -                            | -  |

例如：

```
/* enable MFCOM shifter error interrupt */
mfcom_shifter_error_interrupt_enable (MFCOM_SHIFTER_0);
```

### 函数 `mfcom_timer_interrupt_enable`

函数`mfcom_timer_interrupt_enable`描述见下表：

表 3-473. 函数 `mfcom_timer_interrupt_enable`

|                            |  |
|----------------------------|--|
| 函数名称                       | <code>mfcom_timer_interrupt_enable</code>                        |
| 函数原型                       | <code>void mfcom_timer_interrupt_enable (uint32_t timer);</code> |
| 功能描述                       | 使能定时器中断  |
| 先决条件                       | -  |
| 被调用函数                      | -  |
| 输入参数{in}                   |  |
| <b>timer</b>               | 定时器序号  |
| <code>MFCOM_TIMER_x</code> | <code>x = 0...3</code>   |
| 输出参数{out}                  |  |
| -                          | -  |
| 返回值                        |  |
| -                          | -  |

例如：

```
/* enable MFCOM timer interrupt */
mfcom_timer_interrupt_enable (MFCOM_TIMER_0);
```

### 函数 `mfcom_shifter_dma_enable`

函数`mfcom_shifter_dma_enable`描述见下表：

表 3-474. 函数 `mfcom_shifter_dma_enable`

|                              |  |
|------------------------------|--|
| 函数名称                         | <code>mfcom_shifter_dma_enable</code>                          |
| 函数原型                         | <code>void mfcom_shifter_dma_enable (uint32_t shifter);</code> |
| 功能描述                         | 使能移位器DMA   |
| 先决条件                         | -  |
| 被调用函数                        | -  |
| 输入参数{in}                     |  |
| <b>shifter</b>               | 移位器序号  |
| <code>MFCOM_SHIFTER_x</code> | <code>x = 0...3</code>   |
| 输出参数{out}                    |  |
| -                            | -  |
| 返回值                          |  |
| -                            | -  |

例如：

```
/* enable MFCOM shifter dma */
```

```
mfcom_shifter_dma_enable (MFCOM_SHIFTER_0);
```

### 函数 `mfcom_shifter_interrupt_disable`

函数`mfcom_shifter_interrupt_disable`描述见下表：

表 3-475. 函数 `mfcom_shifter_interrupt_disable`

|                              |   |
|------------------------------|---|
| 函数名称                         | <code>mfcom_shifter_interrupt_disable</code>                          |
| 函数原型                         | <code>void mfcom_shifter_interrupt_disable (uint32_t shifter);</code> |
| 功能描述                         | 禁能移位器中断   |
| 先决条件                         | -   |
| 被调用函数                        | -   |
| 输入参数{in}                     |   |
| <b>shifter</b>               | 移位器序号   |
| <code>MFCOM_SHIFTER_x</code> | <code>x = 0...3</code>  |
| 输出参数{out}                    |   |
| -                            | -   |
| 返回值                          |   |
| -                            | -   |

例如：

```
/* disable MFCOM shifter interrupt */
```

```
mfcom_shifter_interrupt_disable (MFCOM_SHIFTER_0);
```

### 函数 `mfcom_shifter_error_interrupt_disable`

函数`mfcom_shifter_error_interrupt_disable`描述见下表：

表 3-476. 函数 `mfcom_shifter_error_interrupt_disable`

|                              |   |
|------------------------------|---|
| 函数名称                         | <code>mfcom_shifter_error_interrupt_disable</code>                          |
| 函数原型                         | <code>void mfcom_shifter_error_interrupt_disable (uint32_t shifter);</code> |
| 功能描述                         | 禁能移位器错误中断   |
| 先决条件                         | -   |
| 被调用函数                        | -   |
| 输入参数{in}                     |   |
| <b>shifter</b>               | 移位器序号   |
| <code>MFCOM_SHIFTER_x</code> | <code>x = 0...3</code>  |
| 输出参数{out}                    |   |
| -                            | -   |
| 返回值                          |   |
| -                            | -   |

例如：

```
/* disable MFCOM shifter error interrupt */
```

```
mfcom_shifter_error_interrupt_disable (MFCOM_SHIFTER_0);
```

### 函数 `mfcom_timer_interrupt_disable`

函数`mfcom_timer_interrupt_disable`描述见下表：

表 3-477. 函数 `mfcom_timer_interrupt_disable`

|                            |   |
|----------------------------|---|
| 函数名称                       | <code>mfcom_timer_interrupt_disable</code>                        |
| 函数原型                       | <code>void mfcom_timer_interrupt_disable (uint32_t timer);</code> |
| 功能描述                       | 禁能定时器中断   |
| 先决条件                       | -   |
| 被调用函数                      | -   |
| 输入参数{in}                   |   |
| <b>timer</b>               | 定时器序号   |
| <code>MFCOM_TIMER_x</code> | <code>x = 0...3</code>  |
| 输出参数{out}                  |   |
| -                          | -   |
| 返回值                        |   |
| -                          | -   |

例如：

```
/* disable MFCOM timer interrupt */
```

```
mfcom_timer_interrupt_disable (MFCOM_TIMER_0);
```

### 函数 `mfcom_shifter_dma_disable`

函数`mfcom_shifter_dma_disable`描述见下表：

表 3-478. 函数 `mfcom_shifter_dma_disable`

|                        |   |
|------------------------|---|
| 函数名称                   | <code>mfcom_shifter_dma_disable</code>                          |
| 函数原型                   | <code>void mfcom_shifter_dma_disable (uint32_t shifter);</code> |
| 功能描述                   | 禁能移位器DMA  |
| 先决条件                   | -   |
| 被调用函数                  | -   |
| 输入参数{in}               |   |
| <b>shifter</b>         | 移位器序号   |
| <i>MFCOM_SHIFTER_x</i> | <code>x = 0...3</code>  |
| 输出参数{out}              |   |
| -                      | -   |
| 返回值                    |   |
| -                      | -   |

例如：

```
/* disable MFCOM shifter dma */
```

```
mfcom_shifter_dma_disable (MFCOM_SHIFTER_0);
```

### 3.16. MISC

MISC 是对嵌套向量中断控制器（NVIC）和系统定时器（SysTick）操作的软件包。章节 [3.16.1](#) 描述了 NVIC 和 SysTick 的寄存器列表，章节 [3.16.2](#) 对 MISC 库函数进行说明。

#### 3.16.1. 外设寄存器说明

表 3-479. NVIC 寄存器

| 寄存器名称                | 寄存器描述          |
|----------------------|----------------|
| ISER <sup>(1)</sup>  | 中断使能寄存器        |
| ICER <sup>(1)</sup>  | 中断禁能寄存器        |
| ISPR <sup>(1)</sup>  | 中断挂起寄存器        |
| ICPR <sup>(1)</sup>  | 中断清除寄存器        |
| IABR <sup>(1)</sup>  | 中断活动状态寄存器      |
| ITNS <sup>(1)</sup>  | 中断不安全状态寄存器     |
| IPR <sup>(1)</sup>   | 中断优先级寄存器       |
| STIR <sup>(1)</sup>  | 软触发中断寄存器       |
| CPUID <sup>(2)</sup> | CPUID寄存器       |
| ICSR <sup>(2)</sup>  | 中断控制及状态寄存器     |
| VTOR <sup>(2)</sup>  | 向量表偏移量寄存器      |
| AIRCR <sup>(2)</sup> | 应用程序中断及复位控制寄存器 |
| SCR <sup>(2)</sup>   | 系统控制寄存器        |
| CCR <sup>(2)</sup>   | 配置与控制寄存器       |
| SHPR <sup>(2)</sup>  | 系统异常优先级寄存器     |
| SHCSR <sup>(2)</sup> | 系统异常控制及状态寄存器   |

1. 参考 core\_cm33.h 文件中定义的结构体类型 NVIC\_Type
2. 参考 core\_cm33.h 文件中定义的结构体类型 SCB\_Type

表 3-480. SysTick 寄存器

| 寄存器名称                | 寄存器描述           |
|----------------------|-----------------|
| CTRL <sup>(1)</sup>  | Systick控制和状态寄存器 |
| LOAD <sup>(1)</sup>  | Systick重载值寄存器   |
| VAL <sup>(1)</sup>   | Systick当前值寄存器   |
| CALIB <sup>(1)</sup> | Systick校准寄存器    |

1. 参考 core\_cm33.h 文件中定义的结构体类型 SysTick\_Type

#### 3.16.2. 外设库函数说明

MISC库函数列表如下表所示：

表 3-481. MISC 库函数

| 库函数名称                   | 库函数描述    |
|-------------------------|----------|
| nvic_priority_group_set | 配置中断优先级组 |

| 库函数名称                 | 库函数描述       |
|-----------------------|-------------|
| nvic_irq_enable       | 使能NVIC的中断   |
| nvic_irq_disable      | 禁能NVIC的中断   |
| nvic_system_reset     | 复位MCU       |
| nvic_vector_table_set | 设置向量表地址     |
| system_lowpower_set   | 设置系统低功耗模式状态 |
| system_lowpower_reset | 复位系统低功耗模式状态 |
| systick_clksource_set | 设置系统定时器时钟源  |

## 枚举类型 IRQn\_Type

表 3-482. 枚举类型 IRQn\_Type

| 成员名称                        | 功能描述                     |
|-----------------------------|--------------------------|
| WWDGT_IRQn                  | 窗口看门狗中断                  |
| LVD_IRQn                    | 连接到 EXTI 线的 LVD (PVD) 中断 |
| RTC_IRQn                    | RTC 全局中断                 |
| FMC_IRQn                    | FMC 全局中断                 |
| RCU_IRQn                    | RCU 全局中断                 |
| EXTI0_IRQn                  | EXTI 线 0 中断              |
| EXTI1_IRQn                  | EXTI 线 1 中断              |
| EXTI2_IRQn                  | EXTI 线 2 中断              |
| EXTI3_IRQn                  | EXTI 线 3 中断              |
| EXTI4_IRQn                  | EXTI 线 4 中断              |
| DMA0_Channel0_IRQn          | DMA0 通道 0 全局中断           |
| DMA0_Channel1_IRQn          | DMA0 通道 1 全局中断           |
| DMA0_Channel2_IRQn          | DMA0 通道 2 全局中断           |
| DMA0_Channel3_IRQn          | DMA0 通道 3 全局中断           |
| DMA0_Channel4_IRQn          | DMA0 通道 4 全局中断           |
| DMA0_Channel5_IRQn          | DMA0 通道 5 全局中断           |
| DMA0_Channel6_IRQn          | DMA0 通道 6 全局中断           |
| ADC0_1_IRQn                 | ADC0 和 ADC1 中断           |
| CAN0_Message_IRQn           | CAN0 消息缓冲区中断             |
| CAN0_Busoff_IRQn            | CAN0 总线关闭/总线关闭完成中断       |
| CAN0_Error_IRQn             | CAN0 错误中断                |
| CAN0_FastError_IRQn         | CAN0 快速传输错误中断            |
| CAN0_TEC_IRQn               | CAN0 发送警告中断              |
| CAN0_REC_IRQn               | CAN0 接收警告中断              |
| CAN0_WKUP_IRQn              | 连接到 EXTI 线的 CAN0 唤醒中断    |
| TIMER0_BRK_UP_TRG_CMT_IRQn  | TIMER0 中止, 更新, 触发和换相中断   |
| TIMER0_Channel_IRQn         | TIMER0 捕获比较中断            |
| TIMER1_IRQn                 | TIMER1 全局中断              |
| TIMER19_BRK_UP_TRG_CMT_IRQn | TIMER19 中止, 更新, 触发和换相中断  |
| TIMER19_Channel_IRQn        | TIMER19 捕获比较中断           |

|                             |                         |
|-----------------------------|-------------------------|
| I2C0_EV_IRQn                | I2C0 事件中断               |
| I2C0_ER_IRQn                | I2C0 错误中断               |
| I2C1_EV_IRQn                | I2C1 事件中断               |
| I2C1_ER_IRQn                | I2C1 错误中断               |
| SPI0_IRQn                   | SPI0 全局中断               |
| SPI1_IRQn                   | SPI1 全局中断               |
| USART0_IRQn                 | USART0 全局中断             |
| USART1_IRQn                 | USART1 全局中断             |
| USART2_IRQn                 | USART2 全局中断             |
| EXTI10_15_IRQn              | EXTI 线[15:10]中断         |
| EXTI5_9_IRQn                | EXTI 线[9:5]中断           |
| TAMPER_IRQn                 | BKP 篡改中断                |
| TIMER20_BRK_UP_TRG_CMT_IRQn | TIMER20 中止, 更新, 触发和换相中断 |
| TIMER20_Channel_IRQn        | TIMER20 捕获比较            |
| TIMER7_BRK_UP_TRG_CMT_IRQn  | TIMER7 中止, 更新, 触发和换相中断  |
| TIMER7_Channel_IRQn         | TIMER7 捕获比较             |
| DMAMUX_IRQn                 | DMA MUX 中断              |
| SRAMC_ECCSE_IRQn            | SYSCFG SRAM ECC 单比特错误中断 |
| CMP_IRQn                    | 连接到 EXTI 线的 CMP 中断      |
| OVD_IRQn                    | 连接到 EXTI 线的过压检测中断       |
| TIMER5_DAC_IRQn             | TIMER5 或 DAC0 全局中断      |
| TIMER6_IRQn                 | TIMER6 全局中断             |
| DMA1_Channel0_IRQn          | DMA1 通道 0 全局中断          |
| DMA1_Channel1_IRQn          | DMA1 通道 1 全局中断          |
| DMA1_Channel2_IRQn          | DMA1 通道 2 全局中断          |
| DMA1_Channel3_IRQn          | DMA1 通道 3 全局中断          |
| DMA1_Channel4_IRQn          | DMA1 通道 4 全局中断          |
| CAN1_WKUP_IRQn              | 连接到 EXTI 线的 CAN1 唤醒中断   |
| CAN1_Message_IRQn           | CAN1 消息缓冲区中断            |
| CAN1_Busoff_IRQn            | CAN1 总线关闭/总线关闭完成中断      |
| CAN1_Error_IRQn             | CAN1 错误中断               |
| CAN1_FastError_IRQn         | CAN1 快速传输错误中断           |
| CAN1_TEC_IRQn               | CAN1 传输警告中断             |
| CAN1_REC_IRQn               | CAN1 接收警告中断             |
| FPU_IRQn                    | FPU 全局中断                |
| MFCOM_IRQn                  | MFCOM 中断                |

### 函数 nvic\_priority\_group\_set

函数 nvic\_priority\_group\_set 描述见下表:

表 3-483. 函数 nvic\_priority\_group\_set

|      |                         |
|------|-------------------------|
| 函数名称 | nvic_priority_group_set |
|------|-------------------------|



|                         |   |
|-------------------------|---|
| 函数原型                    | void nvic_priority_group_set(uint32_t nvic_prigroup); |
| 功能描述                    | 配置优先级组的位长度  |
| 先决条件                    | -   |
| 被调用函数                   | -   |
| 输入参数{in}                |   |
| nvic_prigroup           | 优先级组  |
| NVIC_PRIGROUP_PRE0_SUB4 | 0位用于抢占优先级，4位用于次优先级                                    |
| NVIC_PRIGROUP_PRE1_SUB3 | 1位用于抢占优先级，3位用于次优先级                                    |
| NVIC_PRIGROUP_PRE2_SUB2 | 2位用于抢占优先级，2位用于次优先级                                    |
| NVIC_PRIGROUP_PRE3_SUB1 | 3位用于抢占优先级，1位用于次优先级                                    |
| NVIC_PRIGROUP_PRE4_SUB0 | 4位用于抢占优先级，0位用于次优先级                                    |
| 输出参数{out}               |   |
| -                       | -   |
| 返回值                     |   |
| -                       | -   |

例如：

```
/* priority group configuration , 0 bits for pre-emption priority 4 bits for subpriority */
```

```
nvic_priority_group_set(NVIC_PRIGROUP_PRE0_SUB4);
```

### 函数 nvic\_irq\_enable

函数nvic\_irq\_enable描述见下表：

表 3-484. 函数 nvic\_irq\_enable

|                       |   |
|-----------------------|---|
| 函数名称                  | nvic_irq_enable   |
| 函数原型                  | void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority); |
| 功能描述                  | 使能NVIC中断  |
| 先决条件                  | -   |
| 被调用函数                 | nvic_priority_group_set   |
| 输入参数{in}              |   |
| nvic_irq              | NVIC中断，参考枚举类型 <a href="#">表3-482. 枚举类型IRQn_Type</a>   |
| 输入参数{in}              |   |
| nvic_irq_pre_priority | 抢占优先级   |
| nvic_irq_sub_priority | 次优先级  |
| 输出参数{out}             |   |
| -                     | -   |

| 返回值 |   |
|-----|---|
| -   | - |

例如：

```
/* enable window watchDog timer interrupt, pre-priority is 1, sub-priority is 1 */
nvic_irq_enable(WWDGT_IRQn, 1, 1);
```

### 函数 nvic\_irq\_disable

函数nvic\_irq\_disable描述见下表：

**表 3-485. 函数 nvic\_irq\_disable**

|           |   |
|-----------|---|
| 函数名称      | nvic_irq_disable                                    |
| 函数原型      | void nvic_irq_disable(uint8_t nvic_irq);            |
| 功能描述      | 禁能NVIC中断  |
| 先决条件      | -   |
| 被调用函数     | NVIC_DisableIRQ                                     |
| 输入参数{in}  |   |
| nvic_irq  | NVIC中断，参考枚举类型 <a href="#">表3-482. 枚举类型IRQn_Type</a> |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

```
/* disable window watchDog timer interrupt */
nvic_irq_disable(WWDGT_IRQn);
```

**表 3-486. 函数 nvic\_system\_reset**

|           |                               |
|-----------|-------------------------------|
| 函数名称      | nvic_system_reset             |
| 函数原型      | void nvic_system_reset(void); |
| 功能描述      | 复位MCU                         |
| 先决条件      | -                             |
| 被调用函数     | NVIC_SystemReset              |
| 输入参数{in}  |                               |
| -         | -                             |
| 输出参数{out} |                               |
| -         | -                             |
| 返回值       |                               |
| -         | -                             |

例如：

```
/* reset the MCU */
```

```
nvic_system_reset();
```

### 函数 nvic\_vector\_table\_set

函数nvic\_vector\_table\_set描述见下表：

表 3-487. 函数 nvic\_vector\_table\_set

|                    |  |
|--------------------|--|
| 函数名称               | nvic_vector_table_set  |
| 函数原型               | void nvic_vector_table_set(uint32_t nvic_vect_tab, uint32_t offset); |
| 功能描述               | 设置向量表地址  |
| 先决条件               | -  |
| 被调用函数              | -  |
| 输入参数{in}           |  |
| nvic_vect_tab      | RAM或者FLASH基地址  |
| NVIC_VECTTAB_RAM   | RAM基地址   |
| NVIC_VECTTAB_FLASH | FLASH基地址   |
| 输入参数{in}           |  |
| offset             | 向量表偏移量（向量表地址=基地址+偏移量）  |
| 输出参数{out}          |  |
| -                  | -  |
| 返回值                |  |
| -                  | -  |

例如：

```
/* set vector table address = NVIC_VECTTAB_FLASH + 0x200 */
```

```
nvic_vector_table_set(NVIC_VECTTAB_FLASH, 0x200);
```

### 函数 system\_lowpower\_set

函数system\_lowpower\_set描述见下表：

表 3-488. 函数 system\_lowpower\_set

|                        |  |
|------------------------|--|
| 函数名称                   | system_lowpower_set                              |
| 函数原型                   | void system_lowpower_set(uint8_t lowpower_mode); |
| 功能描述                   | 设置系统低功耗模式状态                                      |
| 先决条件                   | -  |
| 被调用函数                  | -  |
| 输入参数{in}               |  |
| lowpower_mode          | 系统低功耗模式的状态                                       |
| SCB_LPM_SLEEP_EXIT_ISR | 该位为1时，退出ISR时一直处于低功耗模式                            |
| SCB_LPM_DEEPSL         | 该位为1时，系统处于deep sleep模式                           |

|                                |                                 |
|--------------------------------|---------------------------------|
| <i>EEP</i>                     |                                 |
| <i>SCB_LPM_WAKE_BY_ALL_INT</i> | 该位为1时，低功耗模式可以被所有中断唤醒（无论中断是否被使能） |
| 输出参数{out}                      |                                 |
| -                              | -                               |
| 返回值                            |                                 |
| -                              | -                               |

例如：

```
/* the system always enter low power mode by exiting from ISR */
```

```
system_lowpower_set(SCB_LPM_SLEEP_EXIT_ISR);
```

### 函数 **system\_lowpower\_reset**

函数system\_lowpower\_reset描述见下表：

**表 3-489. 函数 system\_lowpower\_reset**

|                                |  |
|--------------------------------|--|
| 函数名称                           | system_lowpower_reset                              |
| 函数原型                           | void system_lowpower_reset(uint8_t lowpower_mode); |
| 功能描述                           | 复位系统低功耗模式状态  |
| 先决条件                           | -  |
| 被调用函数                          | -  |
| 输入参数{in}                       |  |
| <b>lowpower_mode</b>           | 系统低功耗模式的状态   |
| <i>SCB_LPM_SLEEP_EXIT_ISR</i>  | 该位为0时，系统将通过退出ISR退出低功耗模式                            |
| <i>SCB_LPM_DEEPSLEEP</i>       | 该位为0时，系统进入sleep模式                                  |
| <i>SCB_LPM_WAKE_BY_ALL_INT</i> | 该位为0时，系统只能被使能的中断唤醒                                 |
| 输出参数{out}                      |  |
| -                              | -  |
| 返回值                            |  |
| -                              | -  |

例如：

```
/* the system will exit low power mode by exiting from ISR */
```

```
system_lowpower_reset(SCB_LPM_SLEEP_EXIT_ISR);
```

### 函数 **systick\_clksource\_set**

函数systick\_clksource\_set描述见下表：

表 3-490. 函数 `systick_clksource_set`

|  |  |
|--|--|
| 函数名称                                     | <code>systick_clksource_set</code>                                   |
| 函数原型                                     | <code>void systick_clksource_set(uint32_t systick_clksource);</code> |
| 功能描述                                     | 设置SysTick时钟源   |
| 先决条件                                     | -  |
| 被调用函数                                    | -  |
| 输入参数{in}                                 |  |
| <code>systick_clksource</code>           | SysTick时钟源   |
| <code>SYSTICK_CLKSOURCE_HCLK</code>      | SysTick时钟源为HCLK时钟  |
| <code>SYSTICK_CLKSOURCE_HCLK_DIV8</code> | SysTick时钟源为HCLK时钟的8分频  |
| 输出参数{out}                                |  |
| -  | -  |
| 返回值                                      |  |
| -  | -  |

例如：

```
/* systick clock source is HCLK/8 */
```

```
systick_clksource_set(SYSTICK_CLKSOURCE_HCLK_DIV8);
```

## 3.17. PMU

电源管理单元提供了三种省电模式，包括睡眠模式，深度睡眠模式和待机模式。章节 [3.17.1](#) 描述了 PMU 的寄存器列表，章节 [3.17.2](#) 对 PMU 库函数进行说明。

### 3.17.1. 外设寄存器说明

PMU 寄存器列表如下表所示：

表 3-491. PMU 寄存器

| 寄存器名称   | 寄存器描述      |
|---------|------------|
| PMU_CTL | 控制寄存器      |
| PMU_CS  | 电源控制和状态寄存器 |

### 3.17.2. 外设库函数说明

PMU 库函数列表如下表所示：

表 3-492. PMU 库函数

| 库函数名称                       | 库函数描述    |
|-----------------------------|----------|
| <code>pmu_deinit</code>     | 复位外设PMU  |
| <code>pmu_lvd_select</code> | 选择低压检测阈值 |

| 库函数名称                           | 库函数描述              |
|---------------------------------|--------------------|
| pmu_lvd_disable                 | 关闭低压检测器            |
| pmu_ovd_select                  | 选择过压检测阈值           |
| pmu_ovd_disable                 | 关闭过压检测器            |
| pmu_lowdriver_mode_enable       | 打开深度睡眠模式下低驱动模式     |
| pmu_lowdriver_mode_disable      | 关闭深度睡眠模式下低驱动模式     |
| pmu_sram1_poweroff_mode_enable  | 关闭深度睡眠模式下SRAM1电源开关 |
| pmu_sram1_poweroff_mode_disable | 打开深度睡眠模式下SRAM1电源开关 |
| pmu_sram2_poweroff_mode_enable  | 关闭深度睡眠模式下SRAM2电源开关 |
| pmu_sram2_poweroff_mode_disable | 打开深度睡眠模式下SRAM2电源开关 |
| pmu_to_sleepmode                | 进入睡眠模式             |
| pmu_to_deepsleepmode            | 进入深度睡眠模式           |
| pmu_to_standbymode              | 进入待机模式             |
| pmu_wakeup_pin_enable           | WKUP引脚唤醒使能         |
| pmu_wakeup_pin_disable          | WKUP引脚唤醒失能         |
| pmu_backup_write_enable         | 备份域写使能             |
| pmu_backup_write_disable        | 备份域写失能             |
| pmu_flag_get                    | 获取标志位              |
| pmu_flag_clear                  | 清除标志位              |

## 函数 pmu\_deinit

函数pmu\_deinit描述见下表：

**表 3-493. 函数 pmu\_deinit**

|           |  |
|-----------|--|
| 函数名称      | pmu_deinit   |
| 函数原型      | void pmu_deinit(void);                             |
| 功能描述      | 复位外设PMU  |
| 先决条件      | -  |
| 被调用函数     | rcu_periph_reset_enable / rcu_periph_reset_disable |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如：

```
/* reset PMU */
```

```
pmu_deinit ();
```

函数 **pmu\_lvd\_select**

函数 **pmu\_lvd\_select** 描述见下表:

表 3-494. 函数 **pmu\_lvd\_select**

|            |  |
|------------|--|
| 函数名称       | pmu_lvd_select                         |
| 函数原型       | void pmu_lvd_select(uint32_t lvd_t_n); |
| 功能描述       | 选择低压检测阈值                               |
| 先决条件       | -                                      |
| 被调用函数      | -                                      |
| 输入参数{in}   |  |
| lvd_t_n    | 电压阈值                                   |
| PMU_LVDT_0 | 电压阈值为2.9V                              |
| PMU_LVDT_1 | 电压阈值为3.1V                              |
| PMU_LVDT_2 | 电压阈值为3.3V                              |
| PMU_LVDT_3 | 电压阈值为3.5V                              |
| PMU_LVDT_4 | 电压阈值为4.0V                              |
| PMU_LVDT_5 | 电压阈值为4.2V                              |
| PMU_LVDT_6 | 电压阈值为4.4V                              |
| PMU_LVDT_7 | 电压阈值为4.6V                              |
| 输出参数{out}  |  |
| -          | -                                      |
| 返回值        |  |
| -          | -                                      |

例如:

```
/* select low voltage detector threshold as 4.6V */
```

```
pmu_lvd_select (PMU_LVDT_7);
```

函数 **pmu\_lvd\_disable**

函数 **pmu\_lvd\_disable** 描述见下表:

表 3-495. 函数 **pmu\_lvd\_disable**

|           |                              |
|-----------|------------------------------|
| 函数名称      | pmu_lvd_disable              |
| 函数原型      | void pmu_lvd_disable (void); |
| 功能描述      | 关闭低压检测器                      |
| 先决条件      | -                            |
| 被调用函数     | -                            |
| 输入参数{in}  |                              |
| -         | -                            |
| 输出参数{out} |                              |
| -         | -                            |
| 返回值       |                              |

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

例如:

```
/* disable PMU lvd */
```

```
pmu_lvd_disable ();
```

### 函数 pmu\_ovd\_select

函数pmu\_ovd\_select描述见下表:

**表 3-496. 函数 pmu\_ovd\_select**

|            |                                     |
|------------|-------------------------------------|
| 函数名称       | pmu_ovd_select                      |
| 函数原型       | void pmu_ovd_select(uint32_t ovdn); |
| 功能描述       | 选择过压检测阈值                            |
| 先决条件       | -                                   |
| 被调用函数      | -                                   |
| 输入参数{in}   |                                     |
| ovdn       | 电压阈值                                |
| PMU_OVDN_0 | 电压阈值为5.0V                           |
| PMU_OVDN_1 | 电压阈值为5.5V                           |
| 输出参数{out}  |                                     |
| -          | -                                   |
| 返回值        |                                     |
| -          | -                                   |

例如:

```
/* select over voltage detector threshold as 5.5V */
```

```
pmu_ovd_select (PMU_OVDN_1);
```

### 函数 pmu\_ovd\_disable

函数pmu\_ovd\_disable描述见下表:

**表 3-497. 函数 pmu\_ovd\_disable**

|           |                              |
|-----------|------------------------------|
| 函数名称      | pmu_ovd_disable              |
| 函数原型      | void pmu_ovd_disable (void); |
| 功能描述      | 关闭过压检测器                      |
| 先决条件      | -                            |
| 被调用函数     | -                            |
| 输入参数{in}  |                              |
| -         | -                            |
| 输出参数{out} |                              |
| -         | -                            |



| 返回值 |   |
|-----|---|
| -   | - |

例如:

```
/* disable PMU ovd */
```

```
pmu_ovd_disable ();
```

### 函数 pmu\_lowdriver\_mode\_enable

函数pmu\_lowdriver\_mode\_enable描述见下表:

表 3-498. 函数 pmu\_lowdriver\_mode\_enable

|           |  |
|-----------|--|
| 函数名称      | pmu_lowdriver_mode_enable              |
| 函数原型      | void pmu_lowdriver_mode_enable (void); |
| 功能描述      | 打开深度睡眠模式下低驱动模式                         |
| 先决条件      | -                                      |
| 被调用函数     | -                                      |
| 输入参数{in}  |  |
| -         | -                                      |
| 输出参数{out} |  |
| -         | -                                      |
| 返回值       |  |
| -         | -                                      |

例如:

```
/* enable low-driver mode in deep-sleep mode */
```

```
pmu_lowdriver_mode_enable ();
```

### 函数 pmu\_lowdriver\_mode\_disable

函数pmu\_lowdriver\_mode\_disable描述见下表:

表 3-499. 函数 pmu\_lowdriver\_mode\_disable

|           |   |
|-----------|---|
| 函数名称      | pmu_lowdriver_mode_disable              |
| 函数原型      | void pmu_lowdriver_mode_disable (void); |
| 功能描述      | 关闭深度睡眠模式下低驱动模式                          |
| 先决条件      | -                                       |
| 被调用函数     | -                                       |
| 输入参数{in}  |   |
| -         | -                                       |
| 输出参数{out} |   |
| -         | -                                       |
| 返回值       |   |

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

例如：

```
/* disable low-driver mode in deep-sleep mode */
```

```
pmu_lowdriver_mode_disable ();
```

### 函数 pmu\_sram1\_poweroff\_mode\_enable

函数pmu\_sram1\_poweroff\_mode\_enable描述见下表：

**表 3-500. 函数 pmu\_sram1\_poweroff\_mode\_enable**

|           |   |
|-----------|---|
| 函数名称      | pmu_sram1_poweroff_mode_enable              |
| 函数原型      | void pmu_sram1_poweroff_mode_enable (void); |
| 功能描述      | 关闭深度睡眠模式下SRAM1电源开关                          |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| -         | -   |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

```
/* SRAM1 power off in deep-sleep mode */
```

```
pmu_sram1_poweroff_mode_enable ();
```

### 函数 pmu\_sram1\_poweroff\_mode\_disable

函数pmu\_sram1\_poweroff\_mode\_disable描述见下表：

**表 3-501. 函数 pmu\_sram1\_poweroff\_mode\_disable**

|           |  |
|-----------|--|
| 函数名称      | pmu_sram1_poweroff_mode_disable              |
| 函数原型      | void pmu_sram1_poweroff_mode_disable (void); |
| 功能描述      | 打开深度睡眠模式下SRAM1电源开关                           |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如：

```
/* SRAM1 power on in deep-sleep mode */
```

```
pmu_sram1_poweroff_mode_disable ();
```

### 函数 pmu\_sram2\_poweroff\_mode\_enable

函数pmu\_sram2\_poweroff\_mode\_enable描述见下表：

表 3-502. 函数 pmu\_sram2\_poweroff\_mode\_enable

|           |   |
|-----------|---|
| 函数名称      | pmu_sram2_poweroff_mode_enable              |
| 函数原型      | void pmu_sram2_poweroff_mode_enable (void); |
| 功能描述      | 关闭深度睡眠模式下SRAM2电源开关                          |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| -         | -   |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

```
/* SRAM2 power off in deep-sleep mode */
```

```
pmu_sram2_poweroff_mode_enable ();
```

### 函数 pmu\_sram2\_poweroff\_mode\_disable

函数pmu\_sram2\_poweroff\_mode\_disable描述见下表：

表 3-503. 函数 pmu\_sram2\_poweroff\_mode\_disable

|           |  |
|-----------|--|
| 函数名称      | pmu_sram2_poweroff_mode_disable              |
| 函数原型      | void pmu_sram2_poweroff_mode_disable (void); |
| 功能描述      | 打开深度睡眠模式下SRAM2电源开关                           |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如：

```
/* SRAM2 power on in deep-sleep mode */
```

```
pmu_sram2_poweroff_mode_disable ();
```

### 函数 pmu\_to\_sleepmode

函数pmu\_to\_sleepmode描述见下表：

**表 3-504. 函数 pmu\_to\_sleepmode**

|              |  |
|--------------|--|
| 函数名称         | pmu_to_sleepmode                             |
| 函数原型         | void pmu_to_sleepmode(uint8_t sleepmodecmd); |
| 功能描述         | 进入睡眠模式                                       |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| sleepmodecmd | 进入睡眠模式命令                                     |
| WFI_CMD      | WFI命令  |
| WFE_CMD      | WFE命令  |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| -            | -  |

例如：

```
/* PMU work at sleep mode */
```

```
pmu_to_sleepmode (WFI_CMD);
```

### 函数 pmu\_to\_deepsleepmode

函数pmu\_to\_deepsleepmode描述见下表：

**表 3-505. 函数 pmu\_to\_deepsleepmode**

|                  |   |
|------------------|---|
| 函数名称             | pmu_to_deepsleepmode  |
| 函数原型             | void pmu_to_deepsleepmode(uint32_t ldo, uint32_t lowdrive, uint8_t deepsleepmodecmd); |
| 功能描述             | 进入深度睡眠模式  |
| 先决条件             | -   |
| 被调用函数            | -   |
| 输入参数{in}         |   |
| ldo              | LDO工作模式   |
| PMU_LDO_NORMAL   | 当系统进入深度睡眠模式时，LDO仍正常工作   |
| PMU_LDO_LOWPOWER | 当系统进入深度睡眠模式时，LDO进入低功耗模式   |

| 输入参数{in}                     |                          |
|------------------------------|--------------------------|
| <b>lowdrive</b>              | 低驱动模式                    |
| <i>PMU_LOWDRIVER_ENABLE</i>  | 当系统进入深度睡眠模式时，LDO进入低驱动模式  |
| <i>PMU_LOWDRIVER_DISABLE</i> | 当系统进入深度睡眠模式时，LDO进入正常驱动模式 |
| 输入参数{in}                     |                          |
| <b>deepsleepmodecmd</b>      | 进入深度睡眠模式命令               |
| <i>WFI_CMD</i>               | WFI命令                    |
| <i>WFE_CMD</i>               | WFE命令                    |
| 输出参数{out}                    |                          |
| -                            | -                        |
| 返回值                          |                          |
| -                            | -                        |

例如：

```
/* PMU work at deepsleep mode */
```

```
pmu_to_deepsleepmode (PMU_LDO_NORMAL, PMU_LOWDRIVER_DISABLE, WFI_CMD);
```

### 函数 pmu\_to\_standbymode

函数pmu\_to\_standbymode描述见下表：

表 3-506. 函数 pmu\_to\_standbymode

| 函数名称      | pmu_to_standbymode             |
|-----------|--------------------------------|
| 函数原型      | void pmu_to_standbymode(void); |
| 功能描述      | 进入待机模式                         |
| 先决条件      | -                              |
| 被调用函数     | -                              |
| 输入参数{in}  |                                |
| -         | -                              |
| 输出参数{out} |                                |
| -         | -                              |
| 返回值       |                                |
| -         | -                              |

例如：

```
/* PMU work at standby mode */
```

```
pmu_to_standby ();
```

**函数 pmu\_wakeup\_pin\_enable**

函数pmu\_wakeup\_pin\_enable描述见下表:

**表 3-507. 函数 pmu\_wakeup\_pin\_enable**

|                 |  |
|-----------------|--|
| 函数名称            | pmu_wakeup_pin_enable                            |
| 函数原型            | void pmu_wakeup_pin_enable(uint32_t wakeup_pin); |
| 功能描述            | WKUP引脚唤醒使能                                       |
| 先决条件            | -  |
| 被调用函数           | -  |
| 输入参数{in}        |  |
| wakeup_pin      | 唤醒引脚   |
| PMU_WAKEUP_PIN0 | 唤醒引脚0 (PA0) 使能                                   |
| PMU_WAKEUP_PIN1 | 唤醒引脚1 (PC13) 使能                                  |
| 输出参数{out}       |  |
| -               | -  |
| 返回值             |  |
| -               | -  |

例如:

```
/* enable wakeup pin */
```

```
pmu_wakeup_pin_enable (PMU_WAKEUP_PIN0);
```

**函数 pmu\_wakeup\_pin\_disable**

函数pmu\_wakeup\_pin\_disable描述见下表:

**表 3-508. 函数 pmu\_wakeup\_pin\_disable**

|                 |   |
|-----------------|---|
| 函数名称            | pmu_wakeup_pin_disable                            |
| 函数原型            | void pmu_wakeup_pin_disable(uint32_t wakeup_pin); |
| 功能描述            | WKUP引脚唤醒失能  |
| 先决条件            | -   |
| 被调用函数           | -   |
| 输入参数{in}        |   |
| wakeup_pin      | 唤醒引脚  |
| PMU_WAKEUP_PIN0 | 唤醒引脚0 (PA0) 失能                                    |
| PMU_WAKEUP_PIN1 | 唤醒引脚1 (PC13) 失能                                   |
| 输出参数{out}       |   |
| -               | -   |
| 返回值             |   |

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

例如:

```
/* disable wakeup pin */
```

```
pmu_wakeup_pin_disable (PMU_WAKEUP_PIN0);
```

### 函数 pmu\_backup\_write\_enable

函数pmu\_backup\_write\_enable描述见下表:

**表 3-509. 函数 pmu\_backup\_write\_enable**

|           |                                      |
|-----------|--------------------------------------|
| 函数名称      | pmu_backup_write_enable              |
| 函数原型      | void pmu_backup_write_enable (void); |
| 功能描述      | 备份域写使能                               |
| 先决条件      | -                                    |
| 被调用函数     | -                                    |
| 输入参数{in}  |                                      |
| -         | -                                    |
| 输出参数{out} |                                      |
| -         | -                                    |
| 返回值       |                                      |
| -         | -                                    |

例如:

```
/* enable backup domain write */
```

```
pmu_backup_write_enable ();
```

### 函数 pmu\_backup\_write\_disable

函数pmu\_backup\_write\_disable描述见下表:

**表 3-510. 函数 pmu\_backup\_write\_disable**

|           |                                       |
|-----------|---------------------------------------|
| 函数名称      | pmu_backup_write_disable              |
| 函数原型      | void pmu_backup_write_disable (void); |
| 功能描述      | 备份域写失能                                |
| 先决条件      | -                                     |
| 被调用函数     | -                                     |
| 输入参数{in}  |                                       |
| -         | -                                     |
| 输出参数{out} |                                       |
| -         | -                                     |
| 返回值       |                                       |
| -         | -                                     |

例如：

```
/* disable backup domain write */  
  
pmu_backup_write_disable ();
```

### 函数 pmu\_flag\_get

函数pmu\_flag\_get描述见下表：

表 3-511. 函数 pmu\_flag\_get

|                  |   |
|------------------|---|
| 函数名称             | pmu_flag_get                            |
| 函数原型             | FlagStatus pmu_flag_get(uint32_t flag); |
| 功能描述             | 获取标志位                                   |
| 先决条件             | -                                       |
| 被调用函数            | -                                       |
| 输入参数{in}         |   |
| flag             | 标志位                                     |
| PMU_FLAG_WAKEUP  | 唤醒标志                                    |
| PMU_FLAG_STANDBY | 待机标志                                    |
| PMU_FLAG_LVD     | 低电压状态标志                                 |
| PMU_FLAG_OVD     | 过电压状态标志                                 |
| 输出参数{out}        |   |
| -                | -                                       |
| 返回值              |   |
| FlagStatus       | SET或RESET                               |

例如：

```
/* get flag state */  
  
FlagStatus status;  
  
status = pmu_flag_get (PMU_FLAG_WAKEUP);
```

### 函数 pmu\_flag\_clear

函数pmu\_flag\_clear描述见下表：

表 3-512. 函数 pmu\_flag\_clear

|          |   |
|----------|---|
| 函数名称     | pmu_flag_clear                            |
| 函数原型     | void pmu_flag_clear(uint32_t flag_reset); |
| 功能描述     | 清除标志位                                     |
| 先决条件     | -   |
| 被调用函数    | -   |
| 输入参数{in} |   |



|  |        |
|--|--------|
| <b>flag_reset</b>                        | 标志位    |
| <i>PMU_FLAG_RESE</i><br><i>T_WAKEUP</i>  | 清除唤醒标志 |
| <i>PMU_FLAG_RESE</i><br><i>T_STANDBY</i> | 清除待机标志 |
| 输出参数{out}                                |        |
| -  | -      |
| 返回值                                      |        |
| -  | -      |

例如：

```
/* clear flag bit */
```

```
pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

## 3.18. RCU

RCU 是复位和时钟单元，复位控制包括三种控制方式：电源复位、系统复位和备份域复位。时钟控制单元提供了一系列频率的时钟功能。章节 [3.18.1](#) 描述了 RCU 的寄存器列表，章节 [3.18.2](#) 对 RCU 库函数进行说明。

### 3.18.1. 外设寄存器说明

RCU寄存器列表如下表所示：

表 3-513. RCU 寄存器

| 寄存器名称       | 寄存器描述       |
|-------------|-------------|
| RCU_CTL     | 控制寄存器       |
| RCU_CFG0    | 配置寄存器0      |
| RCU_INT     | 中断寄存器       |
| RCU_APB2RST | APB2复位寄存器   |
| RCU_APB1RST | APB1复位寄存器   |
| RCU_AHBEN   | AHB使能寄存器    |
| RCU_APB2EN  | APB2使能寄存器   |
| RCU_APB1EN  | APB1使能寄存器   |
| RCU_BDCTL   | 备份域控制寄存器    |
| RCU_RSTSCK  | 复位源/时钟寄存器   |
| RCU_AHBRST  | AHB复位寄存器    |
| RCU_CFG1    | 配置寄存器1      |
| RCU_CFG2    | 配置寄存器2      |
| RCU_VKEY    | 电源解锁寄存器     |
| RCU_DSV     | 深度睡眠模式电压寄存器 |

### 3.18.2. 外设库函数说明

RCU库函数列表如下表所示：

**表 3-514. RCU 库函数**

| 库函数名称                             | 库函数描述                |
|-----------------------------------|----------------------|
| rcu_deinit                        | 复位RCU，将RCU寄存器复位为初始值  |
| rcu_periph_clock_enable           | 使能外设时钟               |
| rcu_periph_clock_disable          | 禁能外设时钟               |
| rcu_periph_reset_enable           | 外设时钟复位使能             |
| rcu_periph_reset_disable          | 外设时钟复位禁能             |
| rcu_periph_clock_sleep_enable     | 在睡眠模式下，使能外设时钟        |
| rcu_periph_clock_sleep_disable    | 在睡眠模式下，禁能外设时钟        |
| rcu_bkp_reset_enable              | 备份域时钟复位使能            |
| rcu_bkp_reset_disable             | 备份域时钟复位禁能            |
| rcu_system_clock_source_config    | 配置选择系统时钟源            |
| rcu_system_clock_source_get       | 获取系统时钟源选择状态          |
| rcu_ahb_clock_config              | 配置AHB时钟预分频选择         |
| rcu_apb1_clock_config             | 配置APB1时钟预分频选择        |
| rcu_apb2_clock_config             | 配置APB2时钟预分频选择        |
| rcu_ckout_config                  | 配置CKOUT时钟源选择及分频系数    |
| rcu_pll_config                    | 配置主PLL时钟             |
| rcu_double_pll_enable             | 两倍PLL功能使能            |
| rcu_double_pll_disable            | 两倍PLL功能禁能            |
| rcu_system_reset_enable           | 系统复位源使能              |
| rcu_system_reset_disable          | 系统复位源禁能              |
| rcu_adc_clock_config              | 配置ADC时钟预分频选择         |
| rcu_rtc_clock_config              | 配置RTC时钟源选择           |
| rcu_usart_clock_config            | 配置USART时钟源时钟         |
| rcu_can_clock_config              | 配置CAN时钟源时钟           |
| rcu_lxtal_drive_capability_config | 配置LXTAL的驱动力          |
| rcu_osci_stab_wait                | 等待振荡器稳定标志位置位或振荡器起振超时 |
| rcu_osci_on                       | 打开振荡器                |
| rcu_osci_off                      | 关闭振荡器                |
| rcu_osci_bypass_mode_enable       | 使能时钟旁路模式             |
| rcu_osci_bypass_mode_disable      | 禁能时钟旁路模式             |
| rcu_hxtal_frequency_scale_select  | 外部晶振频率范围选择           |
| rcu_hxtal_prediv_config           | 配置PLL输入源分频因子         |
| rcu_irc8m_adjust_value_set        | 设置内部8MHz RC振荡器时钟调整值  |
| rcu_hxtal_clock_monitor_enable    | HXTAL时钟监视器使能         |
| rcu_hxtal_clock_monitor_disable   | HXTAL时钟监视器禁能         |
| rcu_lxtal_clock_monitor_enable    | LXTAL时钟监视器使能         |

| 库函数名称                           | 库函数描述           |
|---------------------------------|-----------------|
| rcu_lxtal_clock_monitor_disable | LXTAL时钟监视器禁能    |
| rcu_pll_clock_monitor_enable    | PLL时钟监视器使能      |
| rcu_pll_clock_monitor_disable   | PLL时钟监视器禁能      |
| rcu_voltage_key_unlock          | 解锁电压锁定          |
| rcu_deepsleep_voltage_set       | 设置深度睡眠模式电压      |
| rcu_clock_freq_get              | 获取系统、总线或外设时钟频率  |
| rcu_flag_get                    | 获取时钟稳定状态和外设复位标志 |
| rcu_all_reset_flag_clear        | 清除复位标志          |
| rcu_interrupt_flag_get          | 获取时钟中断和CKM中断标志  |
| rcu_interrupt_flag_clear        | 清除中断标志          |
| rcu_interrupt_enable            | 时钟稳定中断使能        |
| rcu_interrupt_disable           | 时钟稳定中断禁能        |

### 枚举类型 rcu\_periph\_enum

表 3-515. 枚举类型 rcu\_periph\_enum

| 成员名称        | 功能描述      |
|-------------|-----------|
| RCU_DMA0    | DMA0时钟    |
| RCU_DMA1    | DMA1时钟    |
| RCU_DMAMUX  | DMAMUX时钟  |
| RCU_CRC     | CRC时钟     |
| RCU_MFCOM   | MFCOM时钟   |
| RCU_GPIOA   | GPIOA时钟   |
| RCU_GPIOB   | GPIOB时钟   |
| RCU_GPIOC   | GPIOC时钟   |
| RCU_GPIOD   | GPIOD时钟   |
| RCU_GPIOE   | GPIOE时钟   |
| RCU_GPIOF   | GPIOF时钟   |
| RCU_SYSCFG  | SYSCFG时钟  |
| RCU_CMP     | CMP时钟     |
| RCU_ADC0    | ADC0时钟    |
| RCU_ADC1    | ADC1时钟    |
| RCU_TIMER0  | TIMER0时钟  |
| RCU_SPI0    | SPI0时钟    |
| RCU_TIMER7  | TIMER7时钟  |
| RCU_USART0  | USART0时钟  |
| RCU_TIMER19 | TIMER19时钟 |
| RCU_TIMER20 | TIMER20时钟 |
| RCU_TRIGSEL | TRIGSEL时钟 |
| RCU_CAN0    | CAN0时钟    |
| RCU_CAN1    | CAN1时钟    |
| RCU_TIMER1  | TIMER1时钟  |

| 成员名称       | 功能描述     |
|------------|----------|
| RCU_TIMER5 | TIMER5时钟 |
| RCU_TIMER6 | TIMER6时钟 |
| RCU_WWDGT  | WWDGT时钟  |
| RCU_SPI1   | SPI1时钟   |
| RCU_USART1 | USART1时钟 |
| RCU_USART2 | USART2时钟 |
| RCU_I2C0   | I2C0时钟   |
| RCU_I2C1   | I2C1时钟   |
| RCU_BKP    | BKP时钟    |
| RCU_PMU    | PMU时钟    |
| RCU_DAC    | DAC时钟    |
| RCU_RTC    | RTC时钟    |

#### 枚举类型 `rcu_periph_sleep_enum`

表 3-516. 枚举类型 `rcu_periph_sleep_enum`

| 成员名称         | 功能描述   |
|--------------|--------|
| RCU_SRAM_SLP | SRAM时钟 |
| RCU_FMC_SLP  | FMC时钟  |

#### 枚举类型 `rcu_periph_reset_enum`

表 3-517. 枚举类型 `rcu_periph_reset_enum`

| 成员名称          | 功能描述       |
|---------------|------------|
| RCU_DMA0RST   | 复位DMA0时钟   |
| RCU_DMA1RST   | 复位DMA1时钟   |
| RCU_DMAMUXRST | 复位DMAMUX时钟 |
| RCU_CRCRST    | 复位CRC时钟    |
| RCU_MFCOMRST  | 复位MFCOM时钟  |
| RCU_GPIOARST  | 复位GPIOA时钟  |
| RCU_GPIOBRST  | 复位GPIOB时钟  |
| RCU_GPIOCRST  | 复位GPIOC时钟  |
| RCU_GPIODRST  | 复位GPIOD时钟  |
| RCU_GPIOERST  | 复位GPIOE时钟  |
| RCU_GPIOFRST  | 复位GPIOF时钟  |
| RCU_SYSCFGRST | 复位SYSCFG时钟 |
| RCU_CMPRST    | 复位CMP时钟    |
| RCU_ADC0RST   | 复位ADC0时钟   |
| RCU_ADC1RST   | 复位ADC1时钟   |
| RCU_TIMER0RST | 复位TIMER0时钟 |
| RCU_SPI0RST   | 复位SPI0时钟   |
| RCU_TIMER7RST | 复位TIMER7时钟 |

| 成员名称           | 功能描述        |
|----------------|-------------|
| RCU_USART0RST  | 复位USART0时钟  |
| RCU_TIMER19RST | 复位TIMER19时钟 |
| RCU_TIMER20RST | 复位TIMER20时钟 |
| RCU_CAN0RST    | 复位CAN0时钟    |
| RCU_CAN1RST    | 复位CAN1时钟    |
| RCU_TIMER1RST  | 复位TIMER1时钟  |
| RCU_TIMER5RST  | 复位TIMER5时钟  |
| RCU_TIMER6RST  | 复位TIMER6时钟  |
| RCU_WWDGTRST   | 复位WWDGT时钟   |
| RCU_SPI1RST    | 复位SPI1时钟    |
| RCU_USART1RST  | 复位USART1时钟  |
| RCU_USART2RST  | 复位USART2时钟  |
| RCU_I2C0RST    | 复位I2C0时钟    |
| RCU_I2C1RST    | 复位I2C1时钟    |
| RCU_PMURST     | 复位PMU时钟     |
| RCU_DACRST     | 复位DAC时钟     |

### 枚举类型 rcu\_flag\_enum

表 3-518. 枚举类型 rcu\_flag\_enum

| 成员名称               | 功能描述        |
|--------------------|-------------|
| RCU_FLAG_IRC8MSTB  | IRC8M稳定标志   |
| RCU_FLAG_HXTALSTB  | HXTAL稳定标志   |
| RCU_FLAG_PLLSTB    | PLL稳定标志     |
| RCU_FLAG_LXTALSTB  | LXTAL稳定标志   |
| RCU_FLAG_IRC40KSTB | IRC40K稳定标志  |
| RCU_FLAG_BORRST    | 欠压复位标志      |
| RCU_FLAG_LOCKUPRST | CPU锁死复位标志   |
| RCU_FLAG_LVDRST    | 低电压检测错误标志   |
| RCU_FLAG_ECCRST    | 2位ECC错误复位标志 |
| RCU_FLAG_LOHRST    | HXTAL丢失复位标志 |
| RCU_FLAG_LOPRST    | PLL丢失复位标志   |
| RCU_FLAG_V11RST    | 1.1V电压域复位标志 |
| RCU_FLAG_OBLRST    | 选项字节复位标志    |
| RCU_FLAG_EPRST     | 外部引脚复位标志    |
| RCU_FLAG_PORRST    | 电源复位标志      |
| RCU_FLAG_SWRST     | 软件复位标志      |
| RCU_FLAG_FWDGTRST  | 独立看门狗复位标志   |
| RCU_FLAG_WWDGTRST  | 窗口看门狗复位标志   |
| RCU_FLAG_LPRST     | 低功耗复位标志     |

## 枚举类型 `rcu_int_flag_enum`

表 3-519. 枚举类型 `rcu_int_flag_enum`

| 成员名称                                | 功能描述            |
|-------------------------------------|-----------------|
| <code>RCU_INT_FLAG_IRC40KSTB</code> | IRC40K时钟稳定中断标志  |
| <code>RCU_INT_FLAG_LXTALSTB</code>  | LXTAL时钟稳定中断标志   |
| <code>RCU_INT_FLAG_IRC8MSTB</code>  | IRC8M时钟稳定中断标志   |
| <code>RCU_INT_FLAG_HXTALSTB</code>  | HXTAL时钟稳定中断标志   |
| <code>RCU_INT_FLAG_PLLSTB</code>    | PLL时钟稳定中断标志     |
| <code>RCU_INT_FLAG_LCKM</code>      | LXTAL时钟稳定中断标志   |
| <code>RCU_INT_FLAG_PLLM</code>      | PLL时钟监视器标志      |
| <code>RCU_INT_FLAG_CKM</code>       | 外部高速晶振时钟监视器中断标志 |

## 枚举类型 `rcu_int_flag_clear_enum`

表 3-520. 枚举类型 `rcu_int_flag_clear_enum`

| 成员名称                                    | 功能描述              |
|---|-------------------|
| <code>RCU_INT_FLAG_IRC40KSTB_CLR</code> | IRC40K时钟稳定中断清除标志  |
| <code>RCU_INT_FLAG_LXTALSTB_CLR</code>  | LXTAL时钟稳定中断清除标志   |
| <code>RCU_INT_FLAG_IRC8MSTB_CLR</code>  | IRC8M时钟稳定中断清除标志   |
| <code>RCU_INT_FLAG_HXTALSTB_CLR</code>  | HXTAL时钟稳定中断清除标志   |
| <code>RCU_INT_FLAG_PLLSTB_CLR</code>    | PLL时钟稳定中断清除标志     |
| <code>RCU_INT_FLAG_LXTALCKM_CLR</code>  | LXTAL时钟稳定中断清除标志   |
| <code>RCU_INT_FLAG_PLLM_CLR</code>      | PLL时钟监视器清除标志      |
| <code>RCU_INT_FLAG_CKM_CLR</code>       | 外部高速晶振时钟监视器中断清除标志 |

## 枚举类型 `rcu_int_enum`

表 3-521. 枚举类型 `rcu_int_enum`

| 成员名称                           | 功能描述         |
|--------------------------------|--------------|
| <code>RCU_INT_IRC40KSTB</code> | IRC40K时钟稳定中断 |
| <code>RCU_INT_LXTALSTB</code>  | LXTAL时钟稳定中断  |
| <code>RCU_INT_IRC8MSTB</code>  | IRC8M时钟稳定中断  |
| <code>RCU_INT_HXTALSTB</code>  | HXTAL时钟稳定中断  |
| <code>RCU_INT_PLLSTB</code>    | PLL时钟稳定中断    |
| <code>RCU_INT_LCKM</code>      | LXTAL时钟监视器中断 |
| <code>RCU_INT_PLLM</code>      | PLL时钟监视器中断   |

枚举类型 `rcu_osc_type_enum`表 3-522. 枚举类型 `rcu_osc_type_enum`

| 成员名称       | 功能描述      |
|------------|-----------|
| RCU_HXTAL  | 外部高速振荡器   |
| RCU_LXTAL  | 外部低速振荡器   |
| RCU_IRC8M  | IRC8M振荡器  |
| RCU_IRC40K | IRC40K振荡器 |
| RCU_PLL_CK | 锁相环时钟     |

枚举类型 `rcu_clock_freq_enum`表 3-523. 枚举类型 `rcu_clock_freq_enum`

| 成员名称      | 功能描述     |
|-----------|----------|
| CK_SYS    | 系统时钟     |
| CK_AHB    | AHB时钟    |
| CK_APB1   | APB1时钟   |
| CK_APB2   | APB2时钟   |
| CK_USART0 | USART0时钟 |
| CK_USART1 | USART1时钟 |
| CK_USART2 | USART2时钟 |

函数 `rcu_deinit`

函数`rcu_deinit`描述见下表：

表 3-524. 函数 `rcu_deinit`

| 函数名称      | <code>rcu_deinit</code>             |
|-----------|-------------------------------------|
| 函数原形      | <code>void rcu_deinit(void);</code> |
| 功能描述      | 复位RCU，将RCU所有寄存器的值复位成初始值             |
| 先决条件      | -                                   |
| 被调用函数     | -                                   |
| 输入参数{in}  |                                     |
| -         | -                                   |
| 输出参数{out} |                                     |
| -         | -                                   |
| 返回值       |                                     |
| -         | -                                   |

例如：

```
/* deinitialize the RCU */
```

```
rcu_deinit();
```

函数 `rcu_periph_clock_enable`

函数 `rcu_periph_clock_enable` 描述见下表:

表 3-525. 函数 `rcu_periph_clock_enable`

|                          |   |
|--------------------------|---|
| 函数名称                     | <code>rcu_periph_clock_enable</code>                                  |
| 函数原形                     | <code>void rcu_periph_clock_enable(rcu_periph_enum periph);</code>    |
| 功能描述                     | 使能外设时钟  |
| 先决条件                     | -   |
| 被调用函数                    | -   |
| 输入参数{in}                 |   |
| <b>periph</b>            | RCU外设, 具体参考 <a href="#">表3-515. 枚举类型 <code>rcu_periph_enum</code></a> |
| <code>RCU_GPIOx</code>   | GPIOx时钟(x = A,B,C,D,E,F)  |
| <code>RCU_DMAx</code>    | DMAx时钟(x = 0, 1)  |
| <code>RCU_CRC</code>     | CRC时钟   |
| <code>RCU_SYSCFG</code>  | SYSCFG时钟  |
| <code>RCU_CMP</code>     | CMP时钟   |
| <code>RCU_ADCx</code>    | ADCx时钟(x = 0, 1)  |
| <code>RCU_TIMERx</code>  | TIMERx时钟(x = 0,1,5,6,7,19,20)   |
| <code>RCU_SPIx</code>    | SPIx时钟(x = 0,1)   |
| <code>RCU_USARTx</code>  | USARTx时钟(x = 0,1,2)   |
| <code>RCU_MFCOM</code>   | MFCOM时钟   |
| <code>RCU_TRIGSEL</code> | TRIGSEL时钟   |
| <code>RCU_CANx</code>    | CANx时钟(x = 0,1)   |
| <code>RCU_I2Cx</code>    | I2Cx时钟(x = 0,1)   |
| <code>RCU_WWDGT</code>   | WWDGT时钟   |
| <code>RCU_BKP</code>     | BKP时钟   |
| <code>RCU_PMU</code>     | PMU时钟   |
| <code>RCU_DAC</code>     | DAC时钟   |
| <code>RCU_RTC</code>     | RTC时钟   |
| 输出参数{out}                |   |
| -                        | -   |
| 返回值                      |   |
| -                        | -   |

例如:

```
/* enable the USART0 clock */
```

```
rcu_periph_clock_enable(RCU_USART0);
```

函数 `rcu_periph_clock_disable`

函数 `rcu_periph_clock_disable` 描述见下表:



表 3-526. 函数 rcu\_periph\_clock\_disable

|             |   |
|-------------|---|
| 函数名称        | rcu_periph_clock_disable                                |
| 函数原形        | void rcu_periph_clock_disable(rcu_periph_enum periph);  |
| 功能描述        | 禁能外设时钟  |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| periph      | RCU外设, 具体参考 <a href="#">表3-515. 枚举类型rcu_periph_enum</a> |
| RCU_GPIOx   | GPIOx时钟(x = A,B,C,D,E,F)                                |
| RCU_DMAX    | DMAx时钟(x = 0, 1)  |
| RCU_CRC     | CRC时钟   |
| RCU_SYSCFG  | SYSCFG时钟  |
| RCU_CMP     | CMP时钟   |
| RCU_ADCx    | ADCx时钟(x = 0, 1)  |
| RCU_TIMERx  | TIMERx时钟(x = 0,1,5,6,7,19,20)                           |
| RCU_SPIx    | SPIx时钟(x = 0,1)   |
| RCU_USARTx  | USARTx时钟(x = 0,1,2)                                     |
| RCU_MFCCOM  | MFCCOM时钟  |
| RCU_TRIGSEL | TRIGSEL时钟   |
| RCU_CANx    | CANx时钟(x = 0,1)   |
| RCU_I2Cx    | I2Cx时钟(x = 0,1)   |
| RCU_WWDGT   | WWDGT时钟   |
| RCU_BKP     | BKP时钟   |
| RCU_PMU     | PMU时钟   |
| RCU_DAC     | DAC时钟   |
| RCU_RTC     | RTC时钟   |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如:

```
/* disable the USART0 clock */
```

```
rcu_periph_clock_disable(RCU_USART0);
```

### 函数 rcu\_periph\_reset\_enable

函数rcu\_periph\_reset\_enable描述见下表:

表 3-527. 函数 rcu\_periph\_reset\_enable

|      |   |
|------|---|
| 函数名称 | rcu_periph_reset_enable   |
| 函数原形 | void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset); |
| 功能描述 | 使能外设复位  |

|                      |   |
|----------------------|---|
| 先决条件                 | -   |
| 被调用函数                | -   |
| 输入参数{in}             |   |
| <b>periph_reset</b>  | RCU外设复位, 参考 <a href="#">表3-527. 函数rcu_periph_reset_enable</a> |
| <i>RCU_GPIOxRST</i>  | 复位GPIO时钟(x = A,B,C,D,E,F)                                     |
| <i>RCU_DMAxRST</i>   | 复位DMAx时钟(x = 0,1)   |
| <i>RCU_DMAMUXRST</i> | 复位DMAMUX时钟  |
| <i>RCU_MFCOMRST</i>  | 复位MFCOM时钟   |
| <i>RCU_CRCRST</i>    | 复位CRC时钟   |
| <i>RCU_SYSCFGRST</i> | 复位SYSCFG时钟  |
| <i>RCU_CMPRST</i>    | 复位CMP时钟   |
| <i>RCU_ADCxRST</i>   | 复位ADCx时钟(x = 0,1)   |
| <i>RCU_TIMERxRST</i> | 复位TIMERx时钟(x = 0,1,5,6,7,19,20)                               |
| <i>RCU_SPIxRST</i>   | 复位SPIx时钟(x = 0,1)   |
| <i>RCU_USARTxRST</i> | 复位USARTx时钟(x = 0,1,2)   |
| <i>RCU_CANxRST</i>   | 复位CANx时钟(x = 0,1)   |
| <i>RCU_I2CxRST</i>   | 复位I2Cx时钟(x = 0,1)   |
| <i>RCU_WWDGTRST</i>  | 复位WWDGT时钟   |
| <i>RCU_PMURST</i>    | 复位PMU时钟   |
| <i>RCU_DACRST</i>    | 复位DAC时钟   |
| <i>RCU_PMURST</i>    | 复位PMU时钟   |
| <i>RCU_DACRST</i>    | 复位DAC时钟   |
| 输出参数{out}            |   |
| -                    | -   |
| 返回值                  |   |
| -                    | -   |

例如:

```
/* enable SPI0 reset */
```

```
rcu_periph_reset_enable(RCU_SPI0RST);
```

### 函数 rcu\_periph\_reset\_disable

函数rcu\_periph\_reset\_disable描述见下表:

表 3-528. 函数 rcu\_periph\_reset\_disable

|                     |  |
|---------------------|--|
| 函数名称                | rcu_periph_reset_disable   |
| 函数原形                | void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset); |
| 功能描述                | 禁能外设复位   |
| 先决条件                | -  |
| 被调用函数               | -  |
| 输入参数{in}            |  |
| <b>periph_reset</b> | RCU外设复位, 参考 <a href="#">表3-527. 函数rcu_periph_reset_enable</a>      |

|                      |                                 |
|----------------------|---------------------------------|
| <i>RCU_GPIOxRST</i>  | 复位GPIO时钟(x = A,B,C,D,E,F)       |
| <i>RCU_DMAxRST</i>   | 复位DMAx时钟(x = 0,1)               |
| <i>RCU_DMAMUXRST</i> | 复位DMAMUX时钟                      |
| <i>RCU_MFCOMRST</i>  | 复位MFCOM时钟                       |
| <i>RCU_CRCRST</i>    | 复位CRC时钟                         |
| <i>RCU_SYSCFGRST</i> | 复位SYSCFG时钟                      |
| <i>RCU_CMPRST</i>    | 复位CMP时钟                         |
| <i>RCU_ADCxRST</i>   | 复位ADCx时钟(x = 0,1)               |
| <i>RCU_TIMERxRST</i> | 复位TIMERx时钟(x = 0,1,5,6,7,19,20) |
| <i>RCU_SPIxRST</i>   | 复位SPIx时钟(x = 0,1)               |
| <i>RCU_USARTxRST</i> | 复位USARTx时钟(x = 0,1,2)           |
| <i>RCU_CANxRST</i>   | 复位CANx时钟(x = 0,1)               |
| <i>RCU_I2CxRST</i>   | 复位I2Cx时钟(x = 0,1)               |
| <i>RCU_WWDGTRST</i>  | 复位WWDGT时钟                       |
| <i>RCU_PMURST</i>    | 复位PMU时钟                         |
| <i>RCU_DACRST</i>    | 复位DAC时钟                         |
| <i>RCU_PMURST</i>    | 复位PMU时钟                         |
| <i>RCU_DACRST</i>    | 复位DAC时钟                         |
| 输出参数{out}            |                                 |
| -                    | -                               |
| 返回值                  |                                 |
| -                    | -                               |

例如:

```
/* disable SPI0 reset */
```

```
rcu_periph_reset_disable(RCU_SPI0RST);
```

### 函数 `rcu_periph_clock_sleep_enable`

函数`rcu_periph_clock_sleep_enable`描述见下表:

**表 3-529. 函数 `rcu_periph_clock_sleep_enable`**

|                      |  |
|----------------------|--|
| 函数名称                 | <code>rcu_periph_clock_sleep_enable</code>                                     |
| 函数原形                 | <code>void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);</code> |
| 功能描述                 | 在睡眠模式下, 使能外设时钟   |
| 先决条件                 | -  |
| 被调用函数                | -  |
| 输入参数{in}             |  |
| <b>periph</b>        | RCU外设, 参考 <a href="#">表3-516. 枚举类型 <code>rcu_periph_sleep_enum</code></a>      |
| <i>RCU_FMC_SLP</i>   | FMC时钟  |
| <i>RCU_SRAM0_SLP</i> | SRAM时钟   |
| 输出参数{out}            |  |
| -                    | -  |

| 返回值 |   |
|-----|---|
| -   | - |

例如：

```
/* enable the FMC clock when in sleep mode */
```

```
rcu_periph_clock_sleep_enable(RCU_FMC_SLP);
```

### 函数 rcu\_periph\_clock\_sleep\_disable

函数rcu\_periph\_clock\_sleep\_disable描述见下表：

表 3-530. 函数 rcu\_periph\_clock\_sleep\_disable

| 函数名称          | rcu_periph_clock_sleep_disable                                     |
|---------------|--|
| 函数原形          | void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph); |
| 功能描述          | 在睡眠模式下，禁能外设时钟  |
| 先决条件          | -  |
| 被调用函数         | -  |
| 输入参数{in}      |  |
| periph        | RCU外设，参考 <a href="#">表3-516. 枚举类型rcu_periph_sleep_enum</a>         |
| RCU_FMC_SLP   | FMC时钟  |
| RCU_SRAM0_SLP | SRAM时钟   |
| 输出参数{out}     |  |
| -             | -  |
| 返回值           |  |
| -             | -  |

例如：

```
/* disable the FMC clock when in sleep mode */
```

```
rcu_periph_clock_sleep_disable(RCU_FMC_SLP);
```

### 函数 rcu\_bkp\_reset\_enable

函数rcu\_bkp\_reset\_enable描述见下表：

表 3-531. 函数 rcu\_bkp\_reset\_enable

| 函数名称      | rcu_bkp_reset_enable             |
|-----------|----------------------------------|
| 函数原形      | void rcu_bkp_reset_enable(void); |
| 功能描述      | 使能BKP复位                          |
| 先决条件      | -                                |
| 被调用函数     | -                                |
| 输入参数{in}  |                                  |
| -         | -                                |
| 输出参数{out} |                                  |

|     |   |
|-----|---|
| -   | - |
| 返回值 |   |
| -   | - |

例如：

```
/* reset the BKP domain */
```

```
rcu_bkp_reset_enable();
```

### 函数 rcu\_bkp\_reset\_disable

函数rcu\_bkp\_reset\_disable描述见下表：

**表 3-532. 函数 rcu\_bkp\_reset\_disable**

|           |                                   |
|-----------|-----------------------------------|
| 函数名称      | rcu_bkp_reset_disable             |
| 函数原形      | void rcu_bkp_reset_disable(void); |
| 功能描述      | 禁能BKP复位                           |
| 先决条件      | -                                 |
| 被调用函数     | -                                 |
| 输入参数{in}  |                                   |
| -         | -                                 |
| 输出参数{out} |                                   |
| -         | -                                 |
| 返回值       |                                   |
| -         | -                                 |

例如：

```
/* disable the BKP domain reset */
```

```
rcu_bkp_reset_disable();
```

### 函数 rcu\_system\_clock\_source\_config

函数rcu\_system\_clock\_source\_config描述见下表：

**表 3-533. 函数 rcu\_system\_clock\_source\_config**

|                        |   |
|------------------------|---|
| 函数名称                   | rcu_system_clock_source_config                        |
| 函数原形                   | void rcu_system_clock_source_config(uint32_t ck_sys); |
| 功能描述                   | 配置选择系统时钟源   |
| 先决条件                   | -   |
| 被调用函数                  | -   |
| 输入参数{in}               |   |
| ck_sys                 | 系统时钟源选择   |
| RCU_CKSYSSRC_I<br>RC8M | 选择CK_IRC8M时钟作为CK_SYS时钟源                               |

|                                      |                         |
|--------------------------------------|-------------------------|
| <i>RCU_CKSYSSRC_</i><br><i>HXTAL</i> | 选择CK_HXTAL时钟作为CK_SYS时钟源 |
| <i>RCU_CKSYSSRC_</i><br><i>PLL</i>   | 选择CK_PLL时钟作为CK_SYS时钟源   |
| 输出参数{out}                            |                         |
| -                                    | -                       |
| 返回值                                  |                         |
| -                                    | -                       |

例如:

```
/* configure the CK_HXTAL as the CK_SYS source */
rcu_system_clock_source_config(RCU_CKSYSSRC_HXTAL);
```

### 函数 rcu\_system\_clock\_source\_get

函数rcu\_system\_clock\_source\_get描述见下表:

表 3-534. 函数 rcu\_system\_clock\_source\_get

|           |  |
|-----------|--|
| 函数名称      | rcu_system_clock_source_get                    |
| 函数原形      | uint32_t rcu_system_clock_source_get(void);    |
| 功能描述      | 获取系统时钟源选择状态                                    |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| uint32_t  | RCU_SCSS_IRC8M / RCU_SCSS_HXTAL / RCU_SCSS_PLL |

例如:

```
uint32_t temp_cksys_status;
/* get the CK_SYS source */
temp_cksys_status = rcu_system_clock_source_get();
```

### 函数 rcu\_ahb\_clock\_config

函数rcu\_ahb\_clock\_config描述见下表:

表 3-535. 函数 rcu\_ahb\_clock\_config

|      |   |
|------|---|
| 函数名称 | rcu_ahb_clock_config                        |
| 函数原形 | void rcu_ahb_clock_config(uint32_t ck_ahb); |
| 功能描述 | 配置AHB时钟预分频选择                                |

|                        |   |
|------------------------|---|
| 先决条件                   | -   |
| 被调用函数                  | -   |
| 输入参数{in}               |   |
| ck_ahb                 | AHB预分频选择  |
| RCU_AHB_CKSYS<br>_DIVx | 选择CK_SYS时钟x分频 (x = 1, 2, 4, 8, 16, 64, 128, 256, 512) |
| 输出参数{out}              |   |
| -                      | -   |
| 返回值                    |   |
| -                      | -   |

例如:

```
/* configure CK_SYS/128 */
```

```
rcu_ahb_clock_config(RCU_AHB_CKSYS_DIV128);
```

### 函数 rcu\_apb1\_clock\_config

函数rcu\_apb1\_clock\_config描述见下表:

表 3-536. 函数 rcu\_apb1\_clock\_config

|                          |   |
|--------------------------|---|
| 函数名称                     | rcu_apb1_clock_config                         |
| 函数原形                     | void rcu_apb1_clock_config(uint32_t ck_apb1); |
| 功能描述                     | 配置APB1时钟预分频选择                                 |
| 先决条件                     | -   |
| 被调用函数                    | -   |
| 输入参数{in}                 |   |
| ck_apb1                  | APB1预分频选择                                     |
| RCU_APB1_CK_AHB<br>_DIVx | 选择CK_AHB时钟x分频作为CK_APB1时钟 (x = 1,2,4,8,16)     |
| 输出参数{out}                |   |
| -                        | -   |
| 返回值                      |   |
| -                        | -   |

例如:

```
/* configure CK_AHB/16 as CK_APB1 */
```

```
rcu_apb1_clock_config(RCU_APB1_CK_AHB_DIV16);
```

### 函数 rcu\_apb2\_clock\_config

函数rcu\_apb2\_clock\_config描述见下表:

表 3-537. 函数 rcu\_apb2\_clock\_config

|                           |   |
|---------------------------|---|
| 函数名称                      | rcu_apb2_clock_config                         |
| 函数原形                      | void rcu_apb2_clock_config(uint32_t ck_apb2); |
| 功能描述                      | 配置APB2时钟预分频选择                                 |
| 先决条件                      | -   |
| 被调用函数                     | -   |
| 输入参数{in}                  |   |
| ck_apb2                   | APB2预分频选择                                     |
| RCU_APB2_CK_AHB<br>B_DIVx | 选择CK_AHB时钟x分频作为CK_APB2时钟 (x = 1,2,4,8,16)     |
| 输出参数{out}                 |   |
| -                         | -   |
| 返回值                       |   |
| -                         | -   |

例如:

```
/* configure CK_AHB/8 as CK_APB2 */
```

```
rcu_apb2_clock_config(RCU_APB2_CK_AHB_DIV8);
```

### 函数 rcu\_ckout\_config

函数rcu\_ckout\_config描述见下表:

表 3-538. 函数 rcu\_ckout\_config

|                        |  |
|------------------------|--|
| 函数名称                   | rcu_ckout_config   |
| 函数原形                   | void rcu_ckout_config(uint32_t ckout_src, uint32_t ckout_div); |
| 功能描述                   | 配置CKOUT时钟源选择及分频系数  |
| 先决条件                   | -  |
| 被调用函数                  | -  |
| 输入参数{in}               |  |
| ckout_src              | CKOUT时钟源选择   |
| RCU_CKOUTSRC_<br>NONE  | 无时钟输出  |
| RCU_CKOUTSRC_<br>RC40K | 选择内部40K RC振荡器时钟  |
| RCU_CKOUTSRC_<br>LXTAL | 选择外部低速晶体振荡器时钟 (LXTAL)  |
| RCU_CKOUTSRC_<br>CKSYS | 选择系统时钟CK_SYS   |
| RCU_CKOUTSRC_<br>RC8M  | 选择内部8M RC振荡器时钟   |
| RCU_CKOUTSRC_<br>HXTAL | 选择外部高速晶体振荡器时钟 (HXTAL)  |



|   |  |
|---|--|
| <code>RCU_CKOUTSRC_</code><br><code>CKPLL_DIV1</code> | 选择CK_PLL时钟                               |
| <code>RCU_CKOUTSRC_</code><br><code>CKPLL_DIV2</code> | 选择 (CK_PLL / 2) 时钟                       |
| 输入参数{in}  |  |
| <code>ckout_div</code>                                | CKOUT分频系数                                |
| <code>RCU_CKOUT_DIVx</code>                           | 将CKOUT所选时钟x分频 (x = 1,2,4,8,16,32,64,128) |
| 输出参数{out}   |  |
| -   | -  |
| 返回值   |  |
| -   | -  |

例如:

```
/* configure the HXTAL as CK_OUT clock source */
```

```
rcu_ckout_config(RCU_CKOUTSRC_HXTAL, RCU_CKOUT_DIV1);
```

### 函数 `rcu_pll_config`

函数`rcu_pll_config`描述见下表:

表 3-539. 函数 `rcu_pll_config`

|   |   |
|---|---|
| 函数名称  | <code>rcu_pll_config</code>   |
| 函数原形  | <code>void rcu_pll_config(uint32_t pll_src, uint32_t pll_mul);</code> |
| 功能描述  | 配置主PLL时钟  |
| 先决条件  | -   |
| 被调用函数   | -   |
| 输入参数{in}  |   |
| <code>pll_src</code>                                | PLL时钟源选择  |
| <code>RCU_PLLSRC_IRC</code><br><code>8M_DIV2</code> | IRC8M/2被选择为PLL时钟的时钟源  |
| <code>RCU_PLLSRC_HXTAL</code><br><code>AL</code>    | HXTAL时钟被选择为PLL时钟的时钟源  |
| 输入参数{in}  |   |
| <code>pll_mul</code>                                | PLL时钟倍频因子   |
| <code>RCU_PLL_MULx</code>                           | PLL源时钟 * x (x = 2..32)  |
| 输出参数{out}   |   |
| -   | -   |
| 返回值   |   |
| -   | -   |

例如:

```
/* configure the PLL */
```

```
rcu_pll_config(RCU_PLLSRC_HXTAL, RCU_PLL_MUL10);
```

## 函数 rcu\_double\_pll\_enable

函数rcu\_double\_pll\_enable描述见下表：

**表 3-540. 函数 rcu\_double\_pll\_enable**

|           |                                   |
|-----------|-----------------------------------|
| 函数名称      | rcu_double_pll_enable             |
| 函数原形      | void rcu_double_pll_enable(void); |
| 功能描述      | 两倍PLL功能使能                         |
| 先决条件      | -                                 |
| 被调用函数     | -                                 |
| 输入参数{in}  |                                   |
| -         | -                                 |
| 输出参数{out} |                                   |
| -         | -                                 |
| 返回值       |                                   |
| -         | -                                 |

例如：

```
/* enable double PLL clock */
```

```
rcu_double_pll_enable();
```

## 函数 rcu\_double\_pll\_disable

函数rcu\_double\_pll\_disable描述见下表：

**表 3-541. 函数 rcu\_double\_pll\_disable**

|           |                                    |
|-----------|------------------------------------|
| 函数名称      | rcu_double_pll_disable             |
| 函数原形      | void rcu_double_pll_disable(void); |
| 功能描述      | 两倍PLL功能禁能                          |
| 先决条件      | -                                  |
| 被调用函数     | -                                  |
| 输入参数{in}  |                                    |
| -         | -                                  |
| 输出参数{out} |                                    |
| -         | -                                  |
| 返回值       |                                    |
| -         | -                                  |

例如：

```
/* disable double PLL clock */
```

```
rcu_double_pll_disable();
```

**函数 rcu\_system\_reset\_enable**

函数rcu\_system\_reset\_enable描述见下表:

**表 3-542. 函数 rcu\_system\_reset\_enable**

|                       |  |
|-----------------------|--|
| 函数名称                  | rcu_system_reset_enable                              |
| 函数原形                  | void rcu_system_reset_enable(uint32_t reset_source); |
| 功能描述                  | 系统复位源使能  |
| 先决条件                  | -  |
| 被调用函数                 | -  |
| 输入参数{in}              |  |
| reset_source          | 复位源  |
| RCU_SYSRST_LO<br>CKUP | CPU锁死复位  |
| RCU_SYSRST_LV<br>D    | 低电压检测复位  |
| RCU_SYSRST_EC<br>C    | 2位ECC错误复位  |
| RCU_SYSRST_LO<br>H    | HXTAL丢失复位  |
| RCU_SYSRST_LO<br>P    | PLL丢失复位  |
| 输出参数{out}             |  |
| -                     | -  |
| 返回值                   |  |
| -                     | -  |

例如:

```
/* enable RCU CPU Lock-Up reset */
```

```
rcu_system_reset_enable(RCU_SYSRST_LOCKUP);
```

**函数 rcu\_system\_reset\_disable**

函数rcu\_system\_reset\_disable描述见下表:

**表 3-543. 函数 rcu\_system\_reset\_disable**

|               |   |
|---------------|---|
| 函数名称          | rcu_system_reset_disable                              |
| 函数原形          | void rcu_system_reset_disable(uint32_t reset_source); |
| 功能描述          | 系统复位源禁能   |
| 先决条件          | -   |
| 被调用函数         | -   |
| 输入参数{in}      |   |
| reset_source  | 复位源   |
| RCU_SYSRST_LO | CPU锁死复位   |

|                                  |           |
|----------------------------------|-----------|
| <i>CKUP</i>                      |           |
| <i>RCU_SYSRST_LV</i><br><i>D</i> | 低电压检测复位   |
| <i>RCU_SYSRST_EC</i><br><i>C</i> | 2位ECC错误复位 |
| <i>RCU_SYSRST_LO</i><br><i>H</i> | HXTAL丢失复位 |
| <i>RCU_SYSRST_LO</i><br><i>P</i> | PLL丢失复位   |
| 输出参数{out}                        |           |
| -                                | -         |
| 返回值                              |           |
| -                                | -         |

例如：

```
/* disable RCU CPU Lock-Up reset */
```

```
rcu_system_reset_disable(RCU_SYSRST_LOCKUP);
```

### 函数 rcu\_adc\_clock\_config

函数rcu\_adc\_clock\_config描述见下表：

表 3-544. 函数 rcu\_adc\_clock\_config

|                                       |  |
|---------------------------------------|--|
| 函数名称                                  | rcu_adc_clock_config                         |
| 函数原形                                  | void rcu_adc_clock_config(uint32_t adc_psc); |
| 功能描述                                  | 配置adc时钟预分频选择                                 |
| 先决条件                                  | -  |
| 被调用函数                                 | -  |
| 输入参数{in}                              |  |
| <b>adc_psc</b>                        | ADC预分频选择                                     |
| <i>RCU_CKADC_CK</i><br><i>HB_DIVx</i> | ADC预分频选择CK_AHB/(x) (x = 2,3,...,32)          |
| 输出参数{out}                             |  |
| -                                     | -  |
| 返回值                                   |  |
| -                                     | -  |

例如：

```
/* configure the ADC prescaler factor */
```

```
rcu_adc_clock_config(RCU_CKADC_CKAHB_DIV2);
```

**函数 rcu\_rtc\_clock\_config**

函数rcu\_rtc\_clock\_config描述见下表：

**表 3-545. 函数 rcu\_rtc\_clock\_config**

|                          |   |
|--------------------------|---|
| 函数名称                     | rcu_rtc_clock_config                                  |
| 函数原形                     | void rcu_rtc_clock_config(uint32_t rtc_clock_source); |
| 功能描述                     | 配置RTC时钟   |
| 先决条件                     | -   |
| 被调用函数                    | -   |
| 输入参数{in}                 |   |
| rtc_clock_source         | RTC时钟源选择  |
| RCU_RTCSRC_NONE          | 未选择时钟   |
| RCU_RTCSRC_LXTAL         | 选择CK_LXTAL作为RTC时钟源                                    |
| RCU_RTCSRC_IRC40K        | 选择内部40K RC振荡器时钟作为RTC时钟源                               |
| RCU_RTCSRC_HXTAL_DIV_128 | 选择外部高速晶振128分频作为RTC时钟源                                 |
| 输出参数{out}                |   |
| -                        | -   |
| 返回值                      |   |
| -                        | -   |

例如：

```
/* configure the RTC clock source selection */
```

```
rcu_rtc_clock_config(RCU_RTCSRC_IRC40K);
```

**函数 rcu\_usart\_clock\_config**

函数rcu\_usart\_clock\_config描述见下表：

**表 3-546. 函数 rcu\_usart\_clock\_config**

|           |  |
|-----------|--|
| 函数名称      | rcu_usart_clock_config   |
| 函数原形      | void rcu_usart_clock_config(uint32_t usart_periph, uint32_t usart_clock_source); |
| 功能描述      | 配置串口时钟   |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| usart_idx | USART外设  |
| USARTx    | USARTx(x = 0,1,2)  |

| 输入参数{in}                  |                          |
|---------------------------|--------------------------|
| <b>usart_clock_source</b> | USARTx时钟源                |
| <i>RCU_USARTSRC_HXTAL</i> | 选择CK_HXTAL时钟作为CK_USART时钟 |
| <i>RCU_USARTSRC_CKSYS</i> | 选择CK_SYS时钟作为CK_USART时钟   |
| <i>RCU_USARTSRC_LXTAL</i> | 选择CK_LXTAL时钟作为CK_USART时钟 |
| <i>RCU_USARTSRC_IRC8M</i> | 选择CK_IRC8M时钟作为CK_USART时钟 |
| 输出参数{out}                 |                          |
| -                         | -                        |
| 返回值                       |                          |
| -                         | -                        |

例如：

```
/* configure the LXTAL as USART0 clock */
```

```
rcu_usart_clock_config(USART0, RCU_USARTSRC_LXTAL);
```

### 函数 rcu\_can\_clock\_config

函数rcu\_can\_clock\_config描述见下表：

**表 3-547. 函数 rcu\_can\_clock\_config**

| <b>函数名称</b>                   | rcu_can_clock_config   |
|-------------------------------|--|
| <b>函数原形</b>                   | void rcu_can_clock_config(uint32_t can_periph, uint32_t can_clock_source); |
| <b>功能描述</b>                   | 配置CAN时钟  |
| <b>先决条件</b>                   | -  |
| <b>被调用函数</b>                  | -  |
| 输入参数{in}                      |  |
| <b>can_periph</b>             | CAN外设  |
| <i>CANx</i>                   | CANx(x = 0,1)  |
| 输入参数{in}                      |  |
| <b>can_clock_source</b>       | I2Cx输入时钟源  |
| <i>RCU_CANSRC_HXTAL</i>       | 选择HXTAL时钟作为CAN时钟   |
| <i>RCU_CANSRC_PCLK2</i>       | 选择PCLK2时钟作为CAN时钟   |
| <i>RCU_CANSRC_PCLK2_DIV_2</i> | 选择PCLK2/2时钟作为CAN时钟   |
| <i>RCU_CANSRC_IRC8M</i>       | 选择IRC8M时钟作为CAN时钟   |

| 输出参数{out} |   |
|-----------|---|
| -         | - |
| 返回值       |   |
| -         | - |

例如：

```
/* configure the CAN clock source selection */
```

```
rcu_can_clock_config(CAN0, RCU_CANSRC_IRC8M);
```

### 函数 rcu\_lxtal\_drive\_capability\_config

函数rcu\_lxtal\_drive\_capability\_config描述见下表：

**表 3-548. 函数 rcu\_lxtal\_drive\_capability\_config**

| 函数名称                  | rcu_lxtal_drive_capability_config                              |
|-----------------------|--|
| 函数原形                  | void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap); |
| 功能描述                  | 配置LXTAL驱动能力  |
| 先决条件                  | -  |
| 被调用函数                 | -  |
| 输入参数{in}              |  |
| lxtal_dricap          | LXTAL驱动能力  |
| RCU_LXTAL_LOW_DRI     | 低驱动力   |
| RCU_LXTAL_MED_LOWDRI  | 中低驱动力  |
| RCU_LXTAL_MED_HIGHDRI | 中高驱动力  |
| RCU_LXTAL_HIGH_DRI    | 高驱动力   |
| 输出参数{out}             |  |
| -                     | -  |
| 返回值                   |  |
| -                     | -  |

例如：

```
/* configure the LXTAL drive capability */
```

```
rcu_lxtal_drive_capability_config(RCU_LAXTAL_LOWDRI);
```

### 函数 rcu\_osci\_stab\_wait

函数rcu\_osci\_stab\_wait描述见下表：

表 3-549. 函数 rcu\_osci\_stab\_wait

|            |  |
|------------|--|
| 函数名称       | rcu_osci_stab_wait                                       |
| 函数原形       | ErrStatus rcu_osci_stab_wait(rcu_osci_type_enum osci);   |
| 功能描述       | 等待振荡器稳定标志位置位或振荡器起振超时                                     |
| 先决条件       | -  |
| 被调用函数      | rcu_flag_get   |
| 输入参数{in}   |  |
| osci       | 振荡器类型, 参考 <a href="#">表3-522. 枚举类型rcu_osci_type_enum</a> |
| RCU_HXTAL  | 高速晶体振荡器  |
| RCU_LXTAL  | 低速晶体振荡器  |
| RCU_IRC16M | 内部8M RC振荡器   |
| RCU_IRC40K | 内部40K RC振荡器  |
| RCU_PLL_CK | 锁相环  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| ErrStatus  | SUCCESS 或 ERROR  |

例如:

```
/* wait for oscillator stabilization flag */
if(SUCCESS == rcu_osci_stab_wait(RCU_HXTAL)){
}
```

### 函数 rcu\_osci\_on

函数rcu\_osci\_on描述见下表:

表 3-550. 函数 rcu\_osci\_on

|            |  |
|------------|--|
| 函数名称       | rcu_osci_on  |
| 函数原形       | void rcu_osci_on(rcu_osci_type_enum osci);               |
| 功能描述       | 打开振荡器  |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| osci       | 振荡器类型, 参考 <a href="#">表3-522. 枚举类型rcu_osci_type_enum</a> |
| RCU_HXTAL  | 高速晶体振荡器  |
| RCU_LXTAL  | 低速晶体振荡器  |
| RCU_IRC8M  | 内部8M RC振荡器   |
| RCU_IRC40K | 内部40K RC振荡器  |
| RCU_PLL_CK | 锁相环  |
| 输出参数{out}  |  |
| -          | -  |



| 返回值 |   |
|-----|---|
| -   | - |

例如：

```
/* turn on the high speed crystal oscillator */
```

```
rcu_osc_i_on(RCU_HXTAL);
```

### 函数 rcu\_osc\_i\_off

函数rcu\_osc\_i\_off描述见下表：

表 3-551. 函数 rcu\_osc\_i\_off

| 函数名称       | rcu_osc_i_off  |
|------------|--|
| 函数原形       | void rcu_osc_i_off(rcu_osc_i_type_enum osci);            |
| 功能描述       | 关闭振荡器  |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| osci       | 振荡器类型，参考 <a href="#">表3-522. 枚举类型rcu_osc_i_type_enum</a> |
| RCU_HXTAL  | 高速晶体振荡器  |
| RCU_LXTAL  | 低速晶体振荡器  |
| RCU_IRC8M  | 内部8M RC振荡器   |
| RCU_IRC40K | 内部40K RC振荡器  |
| RCU_PLL_CK | 锁相环  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如：

```
/* turn off the high speed crystal oscillator */
```

```
rcu_osc_i_off(RCU_HXTAL);
```

### 函数 rcu\_osc\_i\_bypass\_mode\_enable

函数rcu\_osc\_i\_bypass\_mode\_enable描述见下表：

表 3-552. 函数 rcu\_osc\_i\_bypass\_mode\_enable

| 函数名称  | rcu_osc_i_bypass_mode_enable                                 |
|-------|--|
| 函数原形  | void rcu_osc_i_bypass_mode_enable(rcu_osc_i_type_enum osci); |
| 功能描述  | 使能振荡器时钟旁路模式  |
| 先决条件  | HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位                             |
| 被调用函数 | -  |

| 输入参数{in}         |   |
|------------------|---|
| <b>osci</b>      | 振荡器类型，参考 <a href="#">表3-522. 枚举类型rcu_osci_type_enum</a> |
| <i>RCU_HXTAL</i> | 高速晶体振荡器   |
| <i>RCU_LXTAL</i> | 低速晶体振荡器   |
| 输出参数{out}        |   |
| -                | -   |
| 返回值              |   |
| -                | -   |

例如：

```
/* enable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_enable(RCU_HXTAL);
```

### 函数 rcu\_osci\_bypass\_mode\_disable

函数rcu\_osci\_bypass\_mode\_disable描述见下表：

**表 3-553. 函数 rcu\_osci\_bypass\_mode\_disable**

| <b>函数名称</b>      | rcu_osci_bypass_mode_disable                                |
|------------------|---|
| <b>函数原形</b>      | void rcu_osci_bypass_mode_disable(rcu_osci_type_enum osci); |
| <b>功能描述</b>      | 禁能振荡器时钟旁路模式   |
| <b>先决条件</b>      | HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位                            |
| <b>被调用函数</b>     | -   |
| 输入参数{in}         |   |
| <b>osci</b>      | 振荡器类型，参考 <a href="#">表3-522. 枚举类型rcu_osci_type_enum</a>     |
| <i>RCU_HXTAL</i> | 高速晶体振荡器   |
| <i>RCU_LXTAL</i> | 低速晶体振荡器   |
| 输出参数{out}        |   |
| -                | -   |
| 返回值              |   |
| -                | -   |

例如：

```
/* disable the HXTAL clock monitor */
```

```
rcu_hxtal_clock_monitor_disable();
```

### 函数 rcu\_hxtal\_frequency\_scale\_select

函数rcu\_hxtal\_frequency\_scale\_select描述见下表：

**表 3-554. 函数 rcu\_hxtal\_frequency\_scale\_select**

|             |   |
|-------------|---|
| <b>函数名称</b> | rcu_hxtal_frequency_scale_select                            |
| <b>函数原形</b> | void rcu_hxtal_frequency_scale_select(uint32_t hxtal_scal); |

|                       |                 |
|-----------------------|-----------------|
| 功能描述                  | 外部晶振频率范围选择      |
| 先决条件                  | -               |
| 被调用函数                 | -               |
| 输入参数{in}              |                 |
| hxtal_scal            | HXTAL频率范围       |
| HXTAL_SCALE_2M_TO_8M  | HXTAL范围是2-8MHz  |
| HXTAL_SCALE_8M_TO_40M | HXTAL范围是8-40MHz |
| 输出参数{out}             |                 |
| -                     | -               |
| 返回值                   |                 |
| -                     | -               |

例如:

```
/* HXTAL frequency scale select */
```

```
rcu_hxtal_frequency_scale_select(HXTAL_SCALE_2M_TO_8M);
```

### 函数 rcu\_hxtal\_prediv\_config

函数rcu\_hxtal\_prediv\_config描述见下表:

表 3-555. 函数 rcu\_hxtal\_prediv\_config

|                |  |
|----------------|--|
| 函数名称           | rcu_hxtal_prediv_config                              |
| 函数原形           | void rcu_hxtal_prediv_config(uint32_t hxtal_prediv); |
| 功能描述           | 配置PLL输入源分频因子   |
| 先决条件           | -  |
| 被调用函数          | -  |
| 输入参数{in}       |  |
| hxtal_prediv   | HXTAL输入源分频因子   |
| RCU_PREDV_DIVx | HXTAL作为PLL输入源分频因子(x = 1..16)                         |
| 输出参数{out}      |  |
| -              | -  |
| 返回值            |  |
| -              | -  |

例如:

```
/* configure the HXTAL divider used as input of PLL */
```

```
rcu_hxtal_prediv_config(RCU_PREDV_DIV1);
```

### 函数 rcu\_irc8m\_adjust\_value\_set

函数rcu\_irc8m\_adjust\_value\_set描述见下表:

表 3-556. 函数 rcu\_irc8m\_adjust\_value\_set

|              |   |
|--------------|---|
| 函数名称         | rcu_irc8m_adjust_value_set                              |
| 函数原形         | void rcu_irc8m_adjust_value_set(uint32_t irc8m_adjval); |
| 功能描述         | 设置内部8MHz RC振荡器时钟调整值                                     |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| irc8m_adjval | IRC8M调整值（0到0x1F之间）                                      |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如：

```
/* set the IRC8M adjust value */
rcu_irc8m_adjust_value_set(0x10);
```

### 函数 rcu\_hxtal\_clock\_monitor\_enable

函数rcu\_hxtal\_clock\_monitor\_enable描述见下表：

表 3-557. 函数 rcu\_hxtal\_clock\_monitor\_enable

|           |  |
|-----------|--|
| 函数名称      | rcu_hxtal_clock_monitor_enable             |
| 函数原形      | void rcu_hxtal_clock_monitor_enable(void); |
| 功能描述      | 使能HXTAL时钟监视器                               |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如：

```
/* enable the HXTAL clock monitor */
rcu_hxtal_clock_monitor_enable();
```

### 函数 rcu\_hxtal\_clock\_monitor\_disable

函数rcu\_hxtal\_clock\_monitor\_disable描述见下表：

表 3-558. 函数 rcu\_hxtal\_clock\_monitor\_disable

|           |   |
|-----------|---|
| 函数名称      | rcu_hxtal_clock_monitor_disable             |
| 函数原形      | void rcu_hxtal_clock_monitor_disable(void); |
| 功能描述      | 禁能HXTAL时钟监视器                                |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| -         | -   |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

```
/* disable the HXTAL clock monitor */
```

```
rcu_hxtal_clock_monitor_disable();
```

### 函数 rcu\_lxtal\_clock\_monitor\_enable

函数rcu\_lxtal\_clock\_monitor\_enable描述见下表：

表 3-559. 函数 rcu\_lxtal\_clock\_monitor\_enable

|           |  |
|-----------|--|
| 函数名称      | rcu_lxtal_clock_monitor_enable             |
| 函数原形      | void rcu_lxtal_clock_monitor_enable(void); |
| 功能描述      | 使能LXTAL时钟监视器                               |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如：

```
/* enable the LXTAL clock monitor */
```

```
rcu_lxtal_clock_monitor_enable();
```

### 函数 rcu\_lxtal\_clock\_monitor\_disable

函数rcu\_lxtal\_clock\_monitor\_disable描述见下表：

表 3-560. 函数 rcu\_lxtal\_clock\_monitor\_disable

|           |   |
|-----------|---|
| 函数名称      | rcu_lxtal_clock_monitor_disable             |
| 函数原形      | void rcu_lxtal_clock_monitor_disable(void); |
| 功能描述      | 禁能LXTAL时钟监视器                                |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| -         | -   |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如:

```
/* disable the LXTAL clock monitor */
```

```
rcu_lxtal_clock_monitor_disable();
```

### 函数 rcu\_voltage\_key\_unlock

函数rcu\_voltage\_key\_unlock描述见下表:

表 3-561. 函数 rcu\_voltage\_key\_unlock

|           |                                    |
|-----------|------------------------------------|
| 函数名称      | rcu_voltage_key_unlock             |
| 函数原形      | void rcu_voltage_key_unlock(void); |
| 功能描述      | 解锁电压寄存器                            |
| 先决条件      | -                                  |
| 被调用函数     | -                                  |
| 输入参数{in}  |                                    |
| -         | -                                  |
| 输出参数{out} |                                    |
| -         | -                                  |
| 返回值       |                                    |
| -         | -                                  |

例如:

```
/* unlock the voltage key*/
```

```
rcu_voltage_key_unlock();
```

### 函数 rcu\_deepsleep\_voltage\_set

函数rcu\_deepsleep\_voltage\_set描述见下表:

表 3-562. 函数 rcu\_deepsleep\_voltage\_set

|                     |   |
|---------------------|---|
| 函数名称                | rcu_deepsleep_voltage_set                       |
| 函数原形                | void rcu_deepsleep_voltage_set(uint32_t dsvol); |
| 功能描述                | 设置深度睡眠模式电压                                      |
| 先决条件                | -   |
| 被调用函数               | -   |
| 输入参数{in}            |   |
| dsvol               | 深度睡眠模式电压  |
| RCU_DEEPSLEEP_V_0_8 | 在深度睡眠模式下内核电压为0.8V                               |
| RCU_DEEPSLEEP_V_0_9 | 在深度睡眠模式下内核电压为0.9V                               |
| RCU_DEEPSLEEP_V_1_0 | 在深度睡眠模式下内核电压为1.0V                               |
| RCU_DEEPSLEEP_V_1_1 | 在深度睡眠模式下内核电压为1.1V                               |
| 输出参数{out}           |   |
| -                   | -   |
| 返回值                 |   |
| -                   | -   |

例如：

```
/* select deep-sleep mode voltage */
```

```
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1_0);
```

### 函数 rcu\_clock\_freq\_get

函数rcu\_clock\_freq\_get描述见下表：

表 3-563. 函数 rcu\_clock\_freq\_get

|           |   |
|-----------|---|
| 函数名称      | rcu_clock_freq_get  |
| 函数原形      | uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);       |
| 功能描述      | 获取系统、总线以及外设时钟频率   |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| clock     | 要获取的时钟频率，具体参考 <a href="#">表3-523. 枚举类型rcu_clock_freq_enum</a> |
| CK_SYS    | 系统时钟频率  |
| CK_AHB    | AHB时钟频率   |
| CK_APB1   | APB1时钟频率  |
| CK_APB2   | APB2时钟频率  |
| CK_USART0 | USART0时钟频率  |
| CK_USART1 | USART1时钟频率  |

|           |  |
|-----------|--|
| CK_USART2 | USART2时钟频率                               |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| uint32_t  | 系统时钟/AHB时钟/APB1时钟/APB2时钟/ADC时钟/USART时钟频率 |

例如:

```
uint32_t temp_freq;
```

```
/* get the system clock frequency */
```

```
temp_freq = rcu_clock_freq_get(CK_SYS);
```

### 函数 rcu\_flag\_get

函数rcu\_flag\_get描述见下表:

表 3-564. 函数 rcu\_flag\_get

| 函数名称               | rcu_flag_get  |
|--------------------|---|
| 函数原形               | FlagStatus rcu_flag_get(rcu_flag_enum flag);              |
| 功能描述               | 获取时钟稳定和外设复位标志   |
| 先决条件               | -   |
| 被调用函数              | -   |
| 输入参数{in}           |   |
| flag               | 时钟稳定和外设复位标志, 参考 <a href="#">表3-518. 枚举类型rcu_flag_enum</a> |
| RCU_FLAG_IRC8MSTB  | IRC8M稳定标志   |
| RCU_FLAG_HXTALSTB  | HXTAL稳定标志   |
| RCU_FLAG_PLLSTB    | PLL时钟稳定标志   |
| RCU_FLAG_LXTALSTB  | LXTAL稳定标志   |
| RCU_FLAG_IRC40KSTB | IRC40K稳定标志  |
| RCU_FLAG_BORRST    | BOR欠压复位标志   |
| RCU_FLAG_LOCKUPRST | CPU锁死错误复位标志   |
| RCU_FLAG_LVDRST    | 低电压检测错误标志   |
| RCU_FLAG_ECCRST    | 2位ECC错误复位标志   |
| RCU_FLAG_LOHRST    | HXTAL丢失错误复位标志   |



|                          |               |
|--------------------------|---------------|
| <i>RCU_FLAG_LOPRST</i>   | LXTAL丢失错误复位标志 |
| <i>RCU_FLAG_V11RST</i>   | 1.1V域电压复位标志   |
| <i>RCU_FLAG_OBLRST</i>   | 选项字节复位标志      |
| <i>RCU_FLAG_EPRS</i>     | 外部引脚复位标志      |
| <i>RCU_FLAG_PORRST</i>   | 电源复位标志        |
| <i>RCU_FLAG_SWRST</i>    | 软件复位标志        |
| <i>RCU_FLAG_FWDGTRST</i> | 独立看门狗复位标志     |
| <i>RCU_FLAG_WWDGTRST</i> | 窗口看门狗复位标志     |
| <i>RCU_FLAG_LPRST</i>    | 低功耗复位标志       |
| 输出参数{out}                |               |
| -                        | -             |
| 返回值                      |               |
| -                        | SET或RESET     |

例如:

```
/* get the clock stabilization flag */
if(RESET != rcu_flag_get(RCU_FLAG_LXTALSTB)){
}
```

### 函数 rcu\_all\_reset\_flag\_clear

函数rcu\_all\_reset\_flag\_clear描述见下表:

表 3-565. 函数 rcu\_all\_reset\_flag\_clear

|           |                                      |
|-----------|--------------------------------------|
| 函数名称      | rcu_all_reset_flag_clear             |
| 函数原形      | void rcu_all_reset_flag_clear(void); |
| 功能描述      | 清除所有复位标志位                            |
| 先决条件      | -                                    |
| 被调用函数     | -                                    |
| 输入参数{in}  |                                      |
| -         | -                                    |
| 输出参数{out} |                                      |
| -         | -                                    |
| 返回值       |                                      |

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

例如：

```
/* clear all the reset flag */
```

```
rcu_all_reset_flag_clear();
```

### 函数 rcu\_interrupt\_flag\_get

函数rcu\_interrupt\_flag\_get描述见下表：

**表 3-566. 函数 rcu\_interrupt\_flag\_get**

| 函数名称                   | rcu_interrupt_flag_get   |
|------------------------|--|
| 函数原形                   | FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag); |
| 功能描述                   | 获取时钟稳定中断和时钟阻塞中断标志  |
| 先决条件                   | -  |
| 被调用函数                  | -  |
| 输入参数{in}               |  |
| int_flag               | 中断以及CKM标志，参考 <a href="#">表3-521. 枚举类型rcu_int_enum</a>          |
| RCU_INT_FLAG_IRC40KSTB | IRC40K稳定中断标志   |
| RCU_INT_FLAG_LXTALSTB  | LXTAL稳定中断标志  |
| RCU_INT_FLAG_IRC8MSTB  | IRC8M稳定中断标志  |
| RCU_INT_FLAG_HXTALSTB  | HXTAL稳定中断标志  |
| RCU_INT_FLAG_PLLSTB    | PLL稳定中断标志  |
| RCU_INT_FLAG_LCKM      | LXTAL时钟监视器中断标志   |
| RCU_INT_FLAG_PLCKM     | PLL时钟监视器中断标志   |
| RCU_INT_FLAG_HCKM      | HXTAL时钟监视器中断标志   |
| 输出参数{out}              |  |
| -                      | -  |
| 返回值                    |  |
| FlagStatus             | SET或RESET  |

例如：

```
/* get the clock stabilization interrupt flag */
```

```
if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){
```

}

**函数 rcu\_interrupt\_flag\_clear**

函数rcu\_interrupt\_flag\_clear描述见下表:

**表 3-567. 函数 rcu\_interrupt\_flag\_clear**

| 函数名称                       | rcu_interrupt_flag_clear   |
|----------------------------|--|
| 函数原形                       | void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag_clear); |
| 功能描述                       | 清除中断标志和时钟阻塞中断标志  |
| 先决条件                       | -  |
| 被调用函数                      | -  |
| 输入参数{in}                   |  |
| int_flag_clear             | 时钟稳定和阻塞中断标志清除, 参考 <a href="#">表3-520. 枚举类型 rcu_int_flag_clear_enum</a> |
| RCU_INT_FLAG_IRC40KSTB_CLR | IRC40K稳定中断清除标志   |
| RCU_INT_FLAG_LXTALSTB_CLR  | LXTAL稳定中断清除标志  |
| RCU_INT_FLAG_IRC8MSTB_CLR  | IRC8M稳定中断清除标志  |
| RCU_INT_FLAG_HXTALSTB_CLR  | HXTAL稳定中断清除标志  |
| RCU_INT_FLAG_PLLSTB_CLR    | PLL稳定中断清除标志  |
| RCU_INT_FLAG_LCKM_CLR      | LXTAL时钟监视器中断清除标志   |
| RCU_INT_FLAG_PLLM_CLR      | PLL时钟监视器中断清除标志   |
| RCU_INT_FLAG_CKMKM_CLR     | HXTAL时钟监视器中断清除标志   |
| 输出参数{out}                  |  |
| -                          | -  |
| 返回值                        |  |
| -                          | -  |

例如:

```
/* clear the interrupt HXTAL stabilization interrupt flag */
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

**函数 rcu\_interrupt\_enable**

函数rcu\_interrupt\_enable描述见下表:

表 3-568. 函数 rcu\_interrupt\_enable

|                       |  |
|-----------------------|--|
| 函数名称                  | rcu_interrupt_enable                                 |
| 函数原形                  | void rcu_interrupt_enable(rcu_int_enum stab_int);    |
| 功能描述                  | 使能时钟稳定中断   |
| 先决条件                  | -  |
| 被调用函数                 | -  |
| 输入参数{in}              |  |
| stb_int               | 时钟稳定中断，具体参考 <a href="#">表3-521. 枚举类型rcu_int_enum</a> |
| RCU_INT_IRC40KS<br>TB | IRC40K时钟稳定中断   |
| RCU_INT_LXTALS<br>TB  | LXTAL时钟稳定中断  |
| RCU_INT_IRC8MS<br>TB  | IRC8M时钟稳定中断  |
| RCU_INT_HXTALS<br>TB  | HXTAL时钟稳定中断  |
| RCU_INT_PLLSTB        | PLL时钟稳定中断  |
| RCU_INT_LCKM          | LXTAL时钟监视器中断   |
| RCU_INT_PLLM          | PLL时钟监视器中断   |
| 输出参数{out}             |  |
| -                     | -  |
| 返回值                   |  |
| -                     | -  |

例如：

```
/* enable the HXTAL stabilization interrupt */
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

### 函数 rcu\_interrupt\_disable

函数rcu\_interrupt\_disable描述见下表：

表 3-569. 函数 rcu\_interrupt\_disable

|                       |  |
|-----------------------|--|
| 函数名称                  | rcu_interrupt_disable                                |
| 函数原形                  | void rcu_interrupt_disable(rcu_int_enum stab_int);   |
| 功能描述                  | 禁能时钟稳定中断   |
| 先决条件                  | -  |
| 被调用函数                 | -  |
| 输入参数{in}              |  |
| stb_int               | 时钟稳定中断，具体参考 <a href="#">表3-521. 枚举类型rcu_int_enum</a> |
| RCU_INT_IRC40KS<br>TB | IRC40K时钟稳定中断   |
| RCU_INT_LXTALS        | LXTAL时钟稳定中断  |

|                                    |              |
|------------------------------------|--------------|
| <i>TB</i>                          |              |
| <i>RCU_INT_IRC8MS</i><br><i>TB</i> | IRC8M时钟稳定中断  |
| <i>RCU_INT_HXTALS</i><br><i>TB</i> | HXTAL时钟稳定中断  |
| <i>RCU_INT_PLLSTB</i>              | PLL时钟稳定中断    |
| <i>RCU_INT_LCKM</i>                | LXTAL时钟监视器中断 |
| <i>RCU_INT_PLLM</i>                | PLL时钟监视器中断   |
| 输出参数{out}                          |              |
| -                                  | -            |
| 返回值                                |              |
| -                                  | -            |

例如：

```
/* disable the HXTAL stabilization interrupt */
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

## 3.19. RTC

实时时钟RTC通常被用作时钟日历。位于备份域中的RTC电路，包含一个32位的累加计数器、一个闹钟、一个预分频器、一个分频器以及RTC时钟配置寄存器。章节[3.19.1](#)描述了RTC的寄存器列表，章节[3.19.2](#)对RTC库函数进行说明。

### 3.19.1. 外设寄存器描述

RTC寄存器列表如下表所示：

表 3-570. RTC 寄存器

| 寄存器名称     | 寄存器描述    |
|-----------|----------|
| RTC_INTEN | 中断使能寄存器  |
| RTC_CTL   | 控制寄存器    |
| RTC_PSCH  | 预分频寄存器高位 |
| RTC_PSCL  | 预分频寄存器低位 |
| RTC_DIVH  | 分频寄存器高位  |
| RTC_DIVL  | 分频寄存器低位  |
| RTC_CNTH  | 计数寄存器高位  |
| RTC_CNTL  | 计数寄存器低位  |
| RTC_ALRMH | 闹钟寄存器高位  |
| RTC_ALRML | 闹钟寄存器低位  |

### 3.19.2. 外设库函数描述

RTC库函数列表如下表所示：

表 3-571. RTC 库函数

| 库函数名称                        | 库函数描述                |
|------------------------------|----------------------|
| rtc_configuration_mode_enter | 进入RTC配置模式            |
| rtc_configuration_mode_exit  | 退出RTC配置模式            |
| rtc_lwoff_wait               | 等待最近一次对RTC寄存器的写操作完成  |
| rtc_register_sync_wait       | 等待RTC寄存器与RTC的APB时钟同步 |
| rtc_counter_get              | 获取RTC计数器的值           |
| rtc_counter_set              | 设置RTC计数器的值           |
| rtc_prescaler_set            | 设置RTC预分频值            |
| rtc_alarm_config             | 设置RTC闹钟值             |
| rtc_divider_get              | 获取RTC分频值             |
| rtc_interrupt_enable         | 使能RTC中断              |
| rtc_interrupt_disable        | 失能RTC中断              |
| rtc_flag_get                 | 获取RTC标志位状态           |
| rtc_flag_clear               | 清除RTC标志位状态           |
| rtc_interrupt_flag_get       | 获取RTC中断标志位状态         |
| rtc_interrupt_flag_clear     | 清除RTC中断标志位状态         |

#### 函数 rtc\_configuration\_mode\_enter

函数rtc\_configuration\_mode\_enter描述见下表：

表 3-572. 函数 rtc\_configuration\_mode\_enter

|           |  |
|-----------|--|
| 函数名称      | rtc_configuration_mode_enter             |
| 函数原型      | void rtc_configuration_mode_enter(void); |
| 功能描述      | 进入RTC配置模式                                |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如：

```
/* enter RTC configuration mode */
rtc_configuration_mode_enter( );
```

**函数 rtc\_configuration\_mode\_exit**

函数rtc\_configuration\_mode\_exit描述见下表：

**表 3-573. 函数 rtc\_configuration\_mode\_exit**

|           |   |
|-----------|---|
| 函数名称      | rtc_configuration_mode_exit             |
| 函数原型      | void rtc_configuration_mode_exit(void); |
| 功能描述      | 退出RTC配置模式                               |
| 先决条件      | -                                       |
| 被调用函数     | -                                       |
| 输入参数{in}  |   |
| -         | -                                       |
| 输出参数{out} |   |
| -         | -                                       |
| 返回值       |   |
| -         | -                                       |

例如：

```
/* exit RTC configuration mode */  
rtc_configuration_mode_exit ( );
```

**函数 rtc\_lwoff\_wait**

函数rtc\_lwoff\_wait描述见下表：

**表 3-574. 函数 rtc\_lwoff\_wait**

|           |                            |
|-----------|----------------------------|
| 函数名称      | rtc_lwoff_wait             |
| 函数原型      | void rtc_lwoff_wait(void); |
| 功能描述      | 等待最近一次对RTC寄存器的写操作完成        |
| 先决条件      | -                          |
| 被调用函数     | -                          |
| 输入参数{in}  |                            |
| -         | -                          |
| 输出参数{out} |                            |
| -         | -                          |
| 返回值       |                            |
| -         | -                          |

例如：

```
/* wait until last write operation on RTC registers has finished */  
rtc_lwoff_wait( );  
/* enable the RTC second interrupt */
```

```
rtc_interrupt_enable(RTC_INT_SECOND);
```

### 函数 rtc\_register\_sync\_wait

函数rtc\_register\_sync\_wait描述见下表:

**表 3-575. 函数 rtc\_register\_sync\_wait**

|           |                                    |
|-----------|------------------------------------|
| 函数名称      | rtc_register_sync_wait             |
| 函数原型      | void rtc_register_sync_wait(void); |
| 功能描述      | 等待RTC寄存器与RTC的APB时钟同步               |
| 先决条件      | -                                  |
| 被调用函数     | -                                  |
| 输入参数{in}  |                                    |
| -         | -                                  |
| 输出参数{out} |                                    |
| -         | -                                  |
| 返回值       |                                    |
| -         | -                                  |

例如:

```
/* wait for RTC registers synchronization */
```

```
rtc_register_sync_wait( );
```

### 函数 rtc\_counter\_get

函数rtc\_counter\_get描述见下表:

**表 3-576. 函数 rtc\_counter\_get**

|           |                                 |
|-----------|---------------------------------|
| 函数名称      | rtc_counter_get                 |
| 函数原型      | uint32_t rtc_counter_get(void); |
| 功能描述      | 获取RTC计时器的值                      |
| 先决条件      | -                               |
| 被调用函数     | -                               |
| 输入参数{in}  |                                 |
| -         | -                               |
| 输出参数{out} |                                 |
| -         | -                               |
| 返回值       |                                 |
| uint32_t  | RTC计时器的值                        |

例如:

```
/* get the counter value */
```

```
uint32_t rtc_counter_value;
```



```
rtc_counter_value = rtc_counter_get ( );
```

### 函数 rtc\_counter\_set

函数rtc\_counter\_set描述见下表:

**表 3-577. Function rtc\_counter\_set**

|           |                                     |
|-----------|-------------------------------------|
| 函数名称      | rtc_counter_set                     |
| 函数原型      | void rtc_counter_set(uint32_t cnt); |
| 功能描述      | 设置RTC计数器的值                          |
| 先决条件      | -                                   |
| 被调用函数     | -                                   |
| 输入参数{in}  |                                     |
| cnt       | RTC计数器的值 (0-0xFFFF FFFF)            |
| 输出参数{out} |                                     |
| -         | -                                   |
| 返回值       |                                     |
| -         | -                                   |

例如:

```
/* wait until last write operation on RTC registers has finished */
```

```
rtc_lwoff_wait( );
```

```
/* set counter value to 0xFFFF */
```

```
rtc_counter_set(0xFFFF);
```

### 函数 rtc\_prescaler\_set

函数rtc\_prescaler\_set描述见下表:

**表 3-578. 函数 rtc\_prescaler\_set**

|           |  |
|-----------|--|
| 函数名称      | rtc_prescaler_set  |
| 函数原型      | void rtc_prescaler_set(uint32_t psc);                      |
| 功能描述      | 设置RTC预分频值  |
| 先决条件      | 调用此函数之前, 必须调用函数rtc_lwoff_wait ( ) (等待标志位LWOFF置位)           |
| 被调用函数     | rtc_configuration_mode_enter / rtc_configuration_mode_exit |
| 输入参数{in}  |  |
| psc       | RTC预分频值 (0-0x000F FFFF)                                    |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如:

```
/* wait until last write operation on RTC registers has finished */
```

```
rtc_lwoff_wait( );
```

```
/* set RTC prescaler value to 0x7FFFF */
```

```
rtc_prescaler_set (0x7FFFF);
```

### 函数 rtc\_alarm\_config

函数rtc\_alarm\_config描述见下表：

表 3-579. 函数 rtc\_alarm\_config

|           |  |
|-----------|--|
| 函数名称      | rtc_alarm_config   |
| 函数原型      | void rtc_alarm_config(uint32_t alarm);                     |
| 功能描述      | 设置RTC闹钟值   |
| 先决条件      | 调用此函数之前，必须调用函数rtc_lwoff_wait ( )（等待标志位LWOFF置位）             |
| 被调用函数     | rtc_configuration_mode_enter / rtc_configuration_mode_exit |
| 输入参数{in}  |  |
| alarm     | RTC闹钟值（0-0xFFFF FFFF）                                      |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如：

```
/* wait until last write operation on RTC registers has finished */
```

```
rtc_lwoff_wait( );
```

```
/* set alarm value to 0xFFFF */
```

```
rtc_alarm_config (0xFFFF);
```

### 函数 rtc\_divider\_get

函数rtc\_divider\_get描述见下表：

表 3-580. 函数 rtc\_divider\_get

|           |                                 |
|-----------|---------------------------------|
| 函数名称      | rtc_divider_get                 |
| 函数原型      | uint32_t rtc_divider_get(void); |
| 功能描述      | 获取RTC分频值                        |
| 先决条件      | -                               |
| 被调用函数     | -                               |
| 输入参数{in}  |                                 |
| -         | -                               |
| 输出参数{out} |                                 |
| -         | -                               |

| 返回值      |        |
|----------|--------|
| uint32_t | RTC分频值 |

例如：

```
/* get the current RTC divider value */

uint32_t rtc_divider_value;

rtc_divider_value = rtc_divider_get( );
```

### 函数 rtc\_interrupt\_enable

函数rtc\_interrupt\_enable描述见下表：

表 3-581. 函数 rtc\_interrupt\_enable

|                  |  |
|------------------|--|
| 函数名称             | rtc_interrupt_enable                           |
| 函数原型             | void rtc_interrupt_enable(uint32_t interrupt); |
| 功能描述             | 使能RTC中断  |
| 先决条件             | 调用此函数之前，必须调用函数rtc_lwoff_wait ( )（等待标志位LWOFF置位） |
| 被调用函数            | -  |
| 输入参数{in}         |  |
| interrupt        | 待使能的RTC中断源                                     |
| RTC_INT_SECOND   | 秒中断  |
| RTC_INT_ALARM    | 闹钟中断   |
| RTC_INT_OVERFLOW | 溢出中断   |
| 输出参数{out}        |  |
| -                | -  |
| 返回值              |  |
| -                | -  |

例如：

```
/* wait until last write operation on RTC registers has finished */

rtc_lwoff_wait( );

/* enable the RTC second interrupt */

rtc_interrupt_enable(RTC_INT_SECOND);
```

### 函数 rtc\_interrupt\_disable

函数rtc\_interrupt\_disable描述见下表：

表 3-582. 函数 rtc\_interrupt\_disable

|      |   |
|------|---|
| 函数名称 | rtc_interrupt_disable                           |
| 函数原型 | void rtc_interrupt_disable(uint32_t interrupt); |

|                  |  |
|------------------|--|
| 功能描述             | 失能RTC中断                                      |
| 先决条件             | 调用此函数之前，必须调用函数rtc_lwoff_wait()（等待标志位LWOFF置位） |
| 被调用函数            | -  |
| 输入参数{in}         |  |
| interrupt        | 待失能的RTC中断源                                   |
| RTC_INT_SECOND   | 秒中断  |
| RTC_INT_ALARM    | 闹钟中断   |
| RTC_INT_OVERFLOW | 溢出中断   |
| 输出参数{out}        |  |
| -                | -  |
| 返回值              |  |
| -                | -  |

例如：

```
/* wait until last write operation on RTC registers has finished */
```

```
rtc_lwoff_wait( );
```

```
/* disable the RTC second interrupt */
```

```
rtc_interrupt_disable(RTC_INT_SECOND);
```

### 函数 rtc\_flag\_get

函数rtc\_flag\_get描述见下表：

表 3-583. 函数 rtc\_flag\_get

|                          |   |
|--------------------------|---|
| 函数名称                     | rtc_flag_get                            |
| 函数原型                     | FlagStatus rtc_flag_get(uint32_t flag); |
| 功能描述                     | 获取RTC标志位状态                              |
| 先决条件                     | -                                       |
| 被调用函数                    | -                                       |
| 输入参数{in}                 |   |
| flag                     | 需要获取的RTC标志位                             |
| RTC_FLAG_SECON<br>D      | 秒中断标志位                                  |
| RTC_FLAG_ALARM           | 闹钟中断标志位                                 |
| RTC_FLAG_OVERFLOW<br>LOW | 溢出中断标志位                                 |
| RTC_FLAG_RSYN            | 寄存器同步标志位                                |
| RTC_FLAG_LWOF            | 最近一次对RTC寄存器的写操作完成标志位                    |
| 输出参数{out}                |   |
| -                        | -                                       |
| 返回值                      |   |

|            |           |
|------------|-----------|
| FlagStatus | SET或RESET |
|------------|-----------|

例如:

```
/* get the RTC overflow interrupt status */
```

```
FlagStatus alarm_status;
```

```
alarm_status = rtc_flag_get (RTC_FLAG_ALARM);
```

### 函数 rtc\_flag\_clear

函数rtc\_flag\_clear描述见下表:

表 3-584. 函数 rtc\_flag\_clear

|                       |                                     |
|-----------------------|-------------------------------------|
| 函数名称                  | rtc_flag_clear                      |
| 函数原型                  | void rtc_flag_clear(uint32_t flag); |
| 功能描述                  | 清除RTC标志位状态                          |
| 先决条件                  | -                                   |
| 被调用函数                 | -                                   |
| 输入参数{in}              |                                     |
| flag                  | 待清除的RTC标志位                          |
| RTC_FLAG_SECON<br>D   | 秒中断标志位                              |
| RTC_FLAG_ALARM        | 闹钟中断标志位                             |
| RTC_FLAG_OVERF<br>LOW | 溢出中断标志位                             |
| RTC_FLAG_RSYN         | 寄存器同步标志位                            |
| 输出参数{out}             |                                     |
| -                     | -                                   |
| 返回值                   |                                     |
| -                     | -                                   |

例如:

```
/* clear the RTC alarm flag */
```

```
rtc_flag_clear (RTC_FLAG_ALARM);
```

### 函数 rtc\_interrupt\_flag\_get

函数rtc\_interrupt\_flag\_get描述见下表:

表 3-585. 函数 rtc\_interrupt\_flag\_get

|      |   |
|------|---|
| 函数名称 | rtc_interrupt_flag_get                            |
| 函数原型 | FlagStatus rtc_interrupt_flag_get(uint32_t flag); |
| 功能描述 | 获取RTC中断标志位状态                                      |
| 先决条件 | -   |

|                           |               |
|---------------------------|---------------|
| 被调用函数                     | -             |
| 输入参数{in}                  |               |
| flag                      | 需要获取的RTC中断标志位 |
| RTC_INT_FLAG_SE<br>COND   | 秒中断标志位        |
| RTC_INT_FLAG_AL<br>ARM    | 闹钟中断标志位       |
| RTC_INT_FLAG_OV<br>ERFLOW | 溢出中断标志位       |
| 输出参数{out}                 |               |
| -                         | -             |
| 返回值                       |               |
| FlagStatus                | SET或RESET     |

例如:

```
/* get the RTC overflow interrupt status */
```

```
FlagStatus alarm_status;
```

```
alarm_status = rtc_interrupt_flag_get (RTC_INT_FLAG_ALARM);
```

### 函数 rtc\_interrupt\_flag\_clear

函数rtc\_interrupt\_flag\_clear描述见下表:

表 3-586. 函数 rtc\_interrupt\_flag\_clear

|                           |   |
|---------------------------|---|
| 函数名称                      | rtc_interrupt_flag_clear                      |
| 函数原型                      | void rtc_interrupt_flag_clear(uint32_t flag); |
| 功能描述                      | 清除RTC中断标志位状态                                  |
| 先决条件                      | -   |
| 被调用函数                     | -   |
| 输入参数{in}                  |   |
| flag                      | 待清除的RTC中断标志位                                  |
| RTC_INT_FLAG_SE<br>COND   | 秒中断标志位  |
| RTC_INT_FLAG_AL<br>ARM    | 闹钟中断标志位                                       |
| RTC_INT_FLAG_OV<br>ERFLOW | 溢出中断标志位                                       |
| 输出参数{out}                 |   |
| -                         | -   |
| 返回值                       |   |
| -                         | -   |

例如:

```
/* clear the RTC alarm interrupt flag */
```

```
rtc_interrupt_flag_clear (RTC_INT_FLAG_ALARM);
```

## 3.20. SPI

SPI/I2S模块可以通过SPI协议或I2S音频协议与外部设备进行通信。章节[3.20.1](#)描述了SPI/I2S的寄存器列表，章节[3.20.2](#)对SPI/I2S库函数进行说明。

### 3.20.1. 外设寄存器说明

SPI/I2S寄存器列表如下表所示：

**表 3-587. SPI/I2S 寄存器**

| 寄存器名称       | 寄存器描述       |
|-------------|-------------|
| SPI_CTL0    | 控制寄存器0      |
| SPI_CTL1    | 控制寄存器1      |
| SPI_STAT    | 状态寄存器       |
| SPI_DATA    | 数据寄存器       |
| SPI_CRCPOLY | CRC多项式寄存器   |
| SPI_RCRC    | 接收CRC寄存器    |
| SPI_TCRC    | 发送CRC寄存器    |
| SPI_I2SCTL  | I2S控制寄存器    |
| SPI_I2SPSC  | I2S时钟预分频寄存器 |
| SPI_QCTL    | 四线SPI控制寄存器  |

### 3.20.2. 外设库函数说明

SPI/I2S库函数列表如下表所示：

**表 3-588. SPI/I2S 库函数**

| 库函数名称                  | 库函数描述              |
|------------------------|--------------------|
| spi_i2s_deinit         | 复位外设SPI/I2S        |
| spi_struct_para_init   | 初始化SPI结构体中所有参数为默认值 |
| spi_init               | 初始化外设SPI           |
| spi_enable             | 使能外设SPI            |
| spi_disable            | 失能外设SPI            |
| i2s_init               | 初始化外设I2S           |
| i2s_psc_config         | 配置I2S预分频器          |
| i2s_enable             | 使能外设I2S            |
| i2s_disable            | 失能外设I2S            |
| spi_nss_output_enable  | 使能外设SPI NSS输出      |
| spi_nss_output_disable | 失能外设SPI NSS输出      |

| 库函数名称                             | 库函数描述                |
|-----------------------------------|----------------------|
| spi_nss_internal_high             | NSS软件模式下NSS引脚拉高      |
| spi_nss_internal_low              | NSS软件模式下NSS引脚拉低      |
| spi_dma_enable                    | 使能外设SPI的DMA功能        |
| spi_dma_disable                   | 失能外设SPI的DMA功能        |
| spi_i2s_data_frame_format_config  | 配置外设SPI/I2S数据帧格式     |
| spi_i2s_data_transmit             | 发送数据                 |
| spi_i2s_data_receive              | 接收数据                 |
| spi_bidirectional_transfer_config | 配置外设SPI的数据传输方向       |
| spi_i2s_format_error_clear        | 清除SPI/I2S帧格式错误标志     |
| spi_crc_polynomial_set            | 设置外设SPI的CRC多项式值      |
| spi_crc_polynomial_get            | 获取外设SPI的CRC多项式值      |
| spi_crc_on                        | 打开外设SPI的CRC功能        |
| spi_crc_off                       | 关闭外设SPI的CRC功能        |
| spi_crc_next                      | 设置外设SPIx下一次传输数据为CRC值 |
| spi_crc_get                       | 外设SPI获取CRC值          |
| spi_crc_error_clear               | 清除SPI CRC错误标志状态      |
| spi_ti_mode_enable                | 使能SPI TI模式           |
| spi_ti_mode_disable               | 禁能SPI TI模式           |
| spi_nssp_mode_enable              | 使能SPI NSS脉冲模式        |
| spi_nssp_mode_disable             | 禁能SPI NSS脉冲模式        |
| spi_quad_enable                   | 使能四线SPI模式            |
| spi_quad_disable                  | 禁能四线SPI模式            |
| spi_quad_write_enable             | 使能四线SPI写             |
| spi_quad_read_enable              | 使能四线SPI读             |
| spi_quad_io23_output_enable       | 使能SPI_IO2和SPI_IO3输出  |
| spi_quad_io23_output_disable      | 禁能SPI_IO2和SPI_IO3输出  |
| spi_i2s_interrupt_enable          | 使能外设SPI/I2S中断        |
| spi_i2s_interrupt_disable         | 失能外设SPI/I2S中断        |
| spi_i2s_interrupt_flag_get        | 获取外设SPI/I2S中断状态      |
| spi_i2s_flag_get                  | 获取外设SPI/I2S标志状态      |

### 结构体 spi\_parameter\_struct

表 3-589. 结构体 spi\_parameter\_struct

| 成员名称        | 功能描述  |
|-------------|---|
| device_mode | 主机或设备模式配置<br>(SPI_MASTER, SPI_SLAVE)  |
| trans_mode  | 传输模式<br>(SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRCEIVE, SPI_TRANSMODE_BDTRANSMIT) |
| frame_size  | 数据帧格式配置<br>(SPI_FRAME_SIZE_xBIT, x=4,5,..16)  |



|                      |  |
|----------------------|--|
| nss                  | NSS由软件或硬件控制配置<br>(SPI_NSS_SOFT, SPI_NSS_HARD)  |
| endian               | 大端或小端模式配置<br>(SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)  |
| clock_polarity_phase | 相位和极性配置<br>(SPI_CK_PL_LOW_PH_1EDGE, SPI_CK_PL_HIGH_PH_1EDGE,<br>SPI_CK_PL_LOW_PH_2EDGE, SPI_CK_PL_HIGH_PH_2EDGE) |
| prescale             | 预分频器配置<br>(SPI_PSC_n (n=2,4,8,16,32,64,128,256))   |

### 函数 spi\_i2s\_deinit

函数spi\_i2s\_deinit描述见下表:

**表 3-590. 函数 spi\_i2s\_deinit**

|            |  |
|------------|--|
| 函数名称       | spi_i2s_deinit                                     |
| 函数原形       | void spi_i2s_deinit(uint32_t spi_periph);          |
| 功能描述       | 复位外设SPI/I2S  |
| 先决条件       | -  |
| 被调用函数      | rcu_periph_reset_enable / rcu_periph_reset_disable |
| 输入参数{in}   |  |
| spi_periph | 外设SPIx   |
| SPIx       | x=0,1  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如:

```
/* reset SPI0 */
```

```
spi_i2s_deinit(SPI0);
```

### 函数 spi\_struct\_para\_init

函数spi\_struct\_para\_init描述见下表:

**表 3-591. 函数 spi\_struct\_para\_init**

|          |  |
|----------|--|
| 函数名称     | spi_struct_para_init   |
| 函数原形     | void spi_struct_para_init(spi_parameter_struct* spi_struct); |
| 功能描述     | 初始化SPI结构体中所有参数为默认值   |
| 先决条件     | -  |
| 被调用函数    | -  |
| 输入参数{in} |  |
| -        | -  |

| 输出参数{out}         |   |
|-------------------|---|
| <b>spi_struct</b> | SPI初始化结构体，结构体成员参考 <a href="#">表3-589. 结构体spi_parameter_struct</a> |
| 返回值               |   |
| -                 | -   |

例如：

```
/* initialize the parameters of SPI */
spi_parameter_struct spi_init_struct;
spi_struct_para_init(&spi_init_struct);
```

## 函数 spi\_init

函数spi\_init描述见下表：

**表 3-592. 函数 spi\_init**

| 函数名称              | spi_init  |
|-------------------|---|
| 函数原形              | void spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct); |
| 功能描述              | 初始化外设SPI  |
| 先决条件              | -   |
| 被调用函数             | -   |
| 输入参数{in}          |   |
| <b>spi_periph</b> | 外设SPIx  |
| <i>SPIx</i>       | x=0,1   |
| 输入参数{in}          |   |
| <b>spi_struct</b> | 初始化结构体，结构体成员参考 <a href="#">表3-589. 结构体spi_parameter_struct</a>        |
| 输出参数{out}         |   |
| -                 | -   |
| 返回值               |   |
| <b>ErrStatus</b>  | -   |

例如：

```
/* initialize SPI0 */
spi_parameter_struct spi_init_struct;

ErrStatus errstatus = ERROR;

spi_init_struct.trans_mode      = SPI_TRANSMODE_BDTRANSMIT;
spi_init_struct.device_mode     = SPI_MASTER;
spi_init_struct.frame_size      = SPI_FRAME_SIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_CLOCK_POLARITY_HIGH_PHASE_2EDGE;
spi_init_struct.nss              = SPI_NSS_SOFT;
```

```
spi_init_struct.prescale      = SPI_PSC_8;

spi_init_struct.endian        = SPI_ENDIAN_MSB;

errorstatus = spi_init(SPI0, &spi_init_struct);
```

### 函数 spi\_enable

函数spi\_enable描述见下表：

表 3-593. 函数 spi\_enable

|            |                                       |
|------------|---------------------------------------|
| 函数名称       | spi_enable                            |
| 函数原形       | void spi_enable(uint32_t spi_periph); |
| 功能描述       | 使能外设SPI                               |
| 先决条件       | -                                     |
| 被调用函数      | -                                     |
| 输入参数{in}   |                                       |
| spi_periph | 外设SPIx                                |
| SPIx       | x=0,1                                 |
| 输出参数{out}  |                                       |
| -          | -                                     |
| 返回值        |                                       |
| -          | -                                     |

例如：

```
/* enable SPI0 */

spi_enable(SPI0);
```

### 函数 spi\_disable

函数spi\_disable描述见下表：

表 3-594. 函数 spi\_disable

|            |  |
|------------|--|
| 函数名称       | spi_disable                            |
| 函数原形       | void spi_disable(uint32_t spi_periph); |
| 功能描述       | 失能外设SPI                                |
| 先决条件       | -                                      |
| 被调用函数      | -                                      |
| 输入参数{in}   |  |
| spi_periph | 外设SPIx                                 |
| SPIx       | x=0,1                                  |
| 输出参数{out}  |  |
| -          | -                                      |
| 返回值        |  |
| -          | -                                      |

例如：

```
/* disable SPI0 */  
  
spi_disable(SPI0);
```

### 函数 i2s\_init

函数i2s\_init描述见下表：

表 3-595. 函数 i2s\_init

|                    |  |
|--------------------|--|
| 函数名称               | i2s_init   |
| 函数原形               | void i2s_init(uint32_t spi_periph, uint32_t mode, uint32_t standard, uint32_t ckpl); |
| 功能描述               | 初始化外设I2S   |
| 先决条件               | -  |
| 被调用函数              | -  |
| 输入参数{in}           |  |
| spi_periph         | 外设I2Sx   |
| SPIx               | x=1  |
| 输入参数{in}           |  |
| mode               | I2S运行模式  |
| I2S_MODE_SLAVE_TX  | I2S从机发送模式  |
| I2S_MODE_SLAVE_RX  | I2S从机接收模式  |
| I2S_MODE_MASTE_RTX | I2S主机发送模式  |
| I2S_MODE_MASTE_RRX | I2S主机接收模式  |
| 输入参数{in}           |  |
| standard           | I2S标准选择  |
| I2S_STD_PHILLIPS   | I2S 飞利浦标准  |
| I2S_STD_MSB        | I2S MSB对齐标准  |
| I2S_STD_LSB        | I2S LSB对齐标准  |
| I2S_STD_PCMSHO_RT  | I2S PCM短帧标准  |
| I2S_STD_PCMLONG    | I2S PCM长帧标准  |
| 输入参数{in}           |  |
| ckpl               | I2S空闲状态时钟极性  |
| I2S_CKPL_LOW       | I2S_CK空闲状态为低电平   |
| I2S_CKPL_HIGH      | I2S_CK空闲状态为高电平   |
| 输出参数{out}          |  |

|     |   |
|-----|---|
| -   | - |
| 返回值 |   |
| -   | - |

例如:

```
/* initialize I2S1 */
```

```
i2s_init(SPI1, I2S_MODE_MASTERTX, I2S_STD_PHILLIPS, I2S_CKPL_LOW);
```

### 函数 i2s\_psc\_config

函数i2s\_psc\_config描述见下表:

表 3-596. 函数 i2s\_psc\_config

|                          |  |
|--------------------------|--|
| 函数名称                     | i2s_psc_config   |
| 函数原形                     | void i2s_psc_config(uint32_t spi_periph, uint32_t audiosample, uint32_t frameformat, uint32_t mckout); |
| 功能描述                     | 配置I2S预分频器  |
| 先决条件                     | -  |
| 被调用函数                    | rcu_clock_freq_get   |
| 输入参数{in}                 |  |
| spi_periph               | 外设I2Sx   |
| SPIx                     | x=1  |
| 输入参数{in}                 |  |
| audiosample              | I2S音频采样频率  |
| I2S_AUDIOSAMPL<br>E_8K   | 音频采样频率为8KHz  |
| I2S_AUDIOSAMPL<br>E_11K  | 音频采样频率为11KHz   |
| I2S_AUDIOSAMPL<br>E_16K  | 音频采样频率为16KHz   |
| I2S_AUDIOSAMPL<br>E_22K  | 音频采样频率为22KHz   |
| I2S_AUDIOSAMPL<br>E_32K  | 音频采样频率为32KHz   |
| I2S_AUDIOSAMPL<br>E_44K  | 音频采样频率为44KHz   |
| I2S_AUDIOSAMPL<br>E_48K  | 音频采样频率为48KHz   |
| I2S_AUDIOSAMPL<br>E_96K  | 音频采样频率为96KHz   |
| I2S_AUDIOSAMPL<br>E_192K | 音频采样频率为192KHz  |
| 输入参数{in}                 |  |

|                                    |                      |
|------------------------------------|----------------------|
| <b>frameformat</b>                 | I2S数据长度和通道长度         |
| <i>I2S_FRAMEFORMAT_DT16B_CH16B</i> | I2S数据长度为16位，通道长度为16位 |
| <i>I2S_FRAMEFORMAT_DT16B_CH32B</i> | I2S数据长度为16位，通道长度为32位 |
| <i>I2S_FRAMEFORMAT_DT24B_CH32B</i> | I2S数据长度为24位，通道长度为32位 |
| <i>I2S_FRAMEFORMAT_DT32B_CH32B</i> | I2S数据长度为32位，通道长度为32位 |
| <b>输入参数{in}</b>                    |                      |
| <b>mckout</b>                      | I2S_MCK输出            |
| <i>I2S_MCKOUT_ENABLE</i>           | I2S_MCK输出使能          |
| <i>I2S_MCKOUT_DISABLE</i>          | I2S_MCK输出禁止          |
| <b>输出参数{out}</b>                   |                      |
| -                                  | -                    |
| <b>返回值</b>                         |                      |
| -                                  | -                    |

例如：

```
/* configure I2S1 prescaler */
```

```
i2s_psc_config(SPI1, I2S_AUDIOSAMPLE_44K, I2S_FRAMEFORMAT_DT16B_CH16B,  
I2S_MCKOUT_DISABLE);
```

### 函数 i2s\_enable

函数i2s\_enable描述见下表：

表 3-597. 函数 i2s\_enable

|                   |                                       |
|-------------------|---------------------------------------|
| <b>函数名称</b>       | i2s_enable                            |
| <b>函数原形</b>       | void i2s_enable(uint32_t spi_periph); |
| <b>功能描述</b>       | 使能外设I2S                               |
| <b>先决条件</b>       | -                                     |
| <b>被调用函数</b>      | -                                     |
| <b>输入参数{in}</b>   |                                       |
| <b>spi_periph</b> | 外设I2Sx                                |
| <i>SPIx</i>       | x=1                                   |
| <b>输出参数{out}</b>  |                                       |
| -                 | -                                     |
| <b>返回值</b>        |                                       |
| -                 | -                                     |

例如：

```
/* enable I2S1 */
```

```
i2s_enable(SPI1);
```

### 函数 i2s\_disable

函数i2s\_disable描述见下表:

表 3-598. 函数 i2s\_disable

|            |  |
|------------|--|
| 函数名称       | i2s_disable                            |
| 函数原形       | void i2s_disable(uint32_t spi_periph); |
| 功能描述       | 失能外设I2S                                |
| 先决条件       | -                                      |
| 被调用函数      | -                                      |
| 输入参数{in}   |  |
| spi_periph | 外设I2Sx                                 |
| SPIx       | x=1                                    |
| 输出参数{out}  |  |
| -          | -                                      |
| 返回值        |  |
| -          | -                                      |

例如:

```
/* disable I2S1 */
```

```
i2s_disable(SPI1);
```

### 函数 spi\_nss\_output\_enable

函数spi\_nss\_output\_enable描述见下表:

表 3-599. 函数 spi\_nss\_output\_enable

|            |  |
|------------|--|
| 函数名称       | spi_nss_output_enable                            |
| 函数原形       | void spi_nss_output_enable(uint32_t spi_periph); |
| 功能描述       | 使能外设SPI NSS输出                                    |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| spi_periph | 外设SPIx   |
| SPIx       | x=0,1  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如:

```
/* enable SPI0 NSS output */
```

```
spi_nss_output_enable(SPI0);
```

### 函数 spi\_nss\_output\_disable

函数spi\_nss\_output\_disable描述见下表:

表 3-600. 函数 spi\_nss\_output\_disable

|            |   |
|------------|---|
| 函数名称       | spi_nss_output_disable                            |
| 函数原形       | void spi_nss_output_disable(uint32_t spi_periph); |
| 功能描述       | 失能外设SPI NSS输出                                     |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| spi_periph | 外设SPIx  |
| SPIx       | x=0,1   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如:

```
/* disable SPI0 NSS output */
```

```
spi_nss_output_disable(SPI0);
```

### 函数 spi\_nss\_internal\_high

函数spi\_nss\_internal\_high描述见下表:

表 3-601. 函数 spi\_nss\_internal\_high

|            |  |
|------------|--|
| 函数名称       | spi_nss_internal_high                            |
| 函数原形       | void spi_nss_internal_high(uint32_t spi_periph); |
| 功能描述       | NSS软件模式下NSS引脚拉高                                  |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| spi_periph | 外设SPIx   |
| SPIx       | x=0,1  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如:



```
/* SPI0 NSS pin is pulled high level in software mode */
```

```
spi_nss_internal_high(SPI0);
```

### 函数 spi\_nss\_internal\_low

函数spi\_nss\_internal\_low描述见下表:

**表 3-602. 函数 spi\_nss\_internal\_low**

|            |   |
|------------|---|
| 函数名称       | spi_nss_internal_low                            |
| 函数原形       | void spi_nss_internal_low(uint32_t spi_periph); |
| 功能描述       | NSS软件模式下NSS引脚拉低                                 |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| spi_periph | 外设SPIx  |
| SPIx       | x=0,1   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如:

```
/* SPI0 NSS pin is pulled low level in software mode */
```

```
spi_nss_internal_low(SPI0);
```

### 函数 spi\_dma\_enable

函数spi\_dma\_enable描述见下表:

**表 3-603. 函数 spi\_dma\_enable**

|                  |  |
|------------------|--|
| 函数名称             | spi_dma_enable   |
| 函数原形             | void spi_dma_enable(uint32_t spi_periph, uint8_t dma); |
| 功能描述             | 使能外设SPI的DMA功能  |
| 先决条件             | -  |
| 被调用函数            | -  |
| 输入参数{in}         |  |
| spi_periph       | 外设SPIx   |
| SPIx             | x=0,1  |
| 输入参数{in}         |  |
| dma              | SPI DMA模式  |
| SPI_DMA_TRANSMIT | SPI发送缓冲区DMA使能  |
| SPI_DMA_RECEIVE  | SPI接收缓冲区DMA使能  |

|           |   |
|-----------|---|
| <i>E</i>  |   |
| 输出参数{out} |   |
| -         | - |
| 返回值       |   |
| -         | - |

例如:

```
/* enable SPI0 transmit data DMA function */
spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);
```

### 函数 spi\_dma\_disable

函数spi\_dma\_disable描述见下表:

**表 3-604. 函数 spi\_dma\_disable**

|                  |   |
|------------------|---|
| 函数名称             | spi_dma_disable   |
| 函数原形             | void spi_dma_disable(uint32_t spi_periph, uint8_t dma); |
| 功能描述             | 失能外设SPI的DMA功能   |
| 先决条件             | -   |
| 被调用函数            | -   |
| 输入参数{in}         |   |
| spi_periph       | 外设SPIx  |
| SPIx             | x=0,1   |
| 输入参数{in}         |   |
| dma              | SPI DMA模式   |
| SPI_DMA_TRANSMIT | SPI发送缓冲区DMA失能   |
| SPI_DMA_RECEIVE  | SPI接收缓冲区DMA失能   |
| 输出参数{out}        |   |
| -                | -   |
| 返回值              |   |
| -                | -   |

例如:

```
/* disable SPI0 transmit data DMA function */
spi_dma_disable(SPI0, SPI_DMA_TRANSMIT);
```

### 函数 spi\_i2s\_data\_frame\_format\_config

函数spi\_i2s\_data\_frame\_format\_config描述见下表:

表 3-605. 函数 spi\_i2s\_data\_frame\_format\_config

|                         |   |
|-------------------------|---|
| 函数名称                    | spi_i2s_data_frame_format_config  |
| 函数原形                    | ErrStatus spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format); |
| 功能描述                    | 配置外设SPI/I2S数据帧格式  |
| 先决条件                    | -   |
| 被调用函数                   | -   |
| 输入参数{in}                |   |
| spi_periph              | 外设SPIx  |
| SPIx                    | x=0,1   |
| 输入参数{in}                |   |
| frame_format            | SPI帧大小  |
| SPI_FRAME_SIZE_x<br>BIT | SPI x位数据帧格式, x=4,5,..16   |
| 输出参数{out}               |   |
| -                       | -   |
| 返回值                     |   |
| ErrStatus               | ERROR或者SUCCESS-   |

例如:

```
/* configure SPI0/I2S0 data frame format size is 16 bits */
```

```
spi_i2s_data_frame_format_config(SPI1, SPI_FRAME_SIZE_16BIT);
```

### 函数 spi\_i2s\_data\_transmit

函数spi\_i2s\_data\_transmit描述见下表:

表 3-606. 函数 spi\_i2s\_data\_transmit

|            |   |
|------------|---|
| 函数名称       | spi_i2s_data_transmit   |
| 函数原形       | void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data); |
| 功能描述       | SPI发送数据   |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| spi_periph | 外设SPIx  |
| SPIx       | x=0,1   |
| 输入参数{in}   |   |
| data       | 16位数据   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如:

```
/* SPI0 transmit data */
```

```
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n]);
```

### 函数 spi\_i2s\_data\_receive

函数spi\_i2s\_data\_receive描述见下表:

表 3-607. 函数 spi\_i2s\_data\_receive

|            |   |
|------------|---|
| 函数名称       | spi_i2s_data_receive                                |
| 函数原形       | uint16_t spi_i2s_data_receive(uint32_t spi_periph); |
| 功能描述       | SPI接收数据   |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| spi_periph | 外设SPIx  |
| SPIx       | x=0,1   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| uint16_t   | 16位数据   |

例如:

```
/* SPI0 receive data */
```

```
spi0_receive_array[receive_n] = spi_i2s_data_receive(SPI0);
```

### 函数 spi\_bidirectional\_transfer\_config

函数spi\_bidirectional\_transfer\_config描述见下表:

表 3-608. 函数 spi\_bidirectional\_transfer\_config

|                    |   |
|--------------------|---|
| 函数名称               | spi_bidirectional_transfer_config   |
| 函数原形               | void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction); |
| 功能描述               | 配置外设SPI的数据传输方向  |
| 先决条件               | -   |
| 被调用函数              | -   |
| 输入参数{in}           |   |
| spi_periph         | 外设SPIx  |
| SPIx               | x=0,1   |
| 输入参数{in}           |   |
| transfer_direction | SPI双向传输输出使能   |
| SPI_BIDIRECTION    | SPI工作在只发送模式   |

|                               |             |
|-------------------------------|-------------|
| AL_TRANSMIT                   |             |
| SPI_BIDIRECTION<br>AL_RECEIVE | SPI工作在只接收模式 |
| 输出参数{out}                     |             |
| -                             | -           |
| 返回值                           |             |
| -                             | -           |

例如:

```
/* SPI0 works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

### 函数 spi\_i2s\_format\_error\_clear

函数spi\_i2s\_format\_error\_clear描述见下表:

表 3-609. 函数 spi\_crc\_error\_clear

|            |   |
|------------|---|
| 函数名称       | spi_i2s_format_error_clear  |
| 函数原形       | void spi_i2s_format_error_clear (uint32_t spi_periph, uint32_t flag); |
| 功能描述       | 清除SPI/I2S帧格式错误标志  |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| spi_periph | 外设SPIx  |
| SPIx       | x=0,1   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如:

```
/* clear SPI/I2S format error flag status */
```

```
spi_i2s_format_error_clear (SPI0, SPI_FLAG_FERR);
```

### 函数 spi\_crc\_polynomial\_set

函数spi\_crc\_polynomial\_set描述见下表:

表 3-610. 函数 spi\_crc\_polynomial\_set

|      |  |
|------|--|
| 函数名称 | spi_crc_polynomial_set   |
| 函数原形 | void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly); |
| 功能描述 | 设置外设SPI的CRC多项式值  |
| 先决条件 | -  |

|                   |         |
|-------------------|---------|
| 被调用函数             | -       |
| 输入参数{in}          |         |
| <b>spi_periph</b> | 外设SPIx  |
| <i>SPIx</i>       | x=0,1   |
| 输入参数{in}          |         |
| <b>crc_poly</b>   | CRC多项式值 |
| 输出参数{out}         |         |
| -                 | -       |
| 返回值               |         |
| -                 | -       |

例如：

```
/* set SPI0 CRC polynomial */
spi_crc_polynomial_set(SPI0,CRC_VALUE);
```

### 函数 spi\_crc\_polynomial\_get

函数spi\_crc\_polynomial\_get描述见下表：

表 3-611. 函数 spi\_crc\_polynomial\_get

|                   |   |
|-------------------|---|
| 函数名称              | spi_crc_polynomial_get                                |
| 函数原形              | uint16_t spi_crc_polynomial_get(uint32_t spi_periph); |
| 功能描述              | 获取外设SPI的CRC多项式值                                       |
| 先决条件              | -   |
| 被调用函数             | -   |
| 输入参数{in}          |   |
| <b>spi_periph</b> | 外设SPIx  |
| <i>SPIx</i>       | x=0,1   |
| 输出参数{out}         |   |
| -                 | -   |
| 返回值               |   |
| <b>uint16_t</b>   | 16位CRC多项式值（0-0xFFFF）                                  |

例如：

```
/* get SPI0 CRC polynomial */
uint16_t crc_val;
crc_val = spi_crc_polynomial_get(SPI0);
```

### 函数 spi\_crc\_on

函数spi\_crc\_on描述见下表：

表 3-612. 函数 spi\_crc\_on

|            |                                       |
|------------|---------------------------------------|
| 函数名称       | spi_crc_on                            |
| 函数原形       | void spi_crc_on(uint32_t spi_periph); |
| 功能描述       | 打开外设SPI的CRC功能                         |
| 先决条件       | -                                     |
| 被调用函数      | -                                     |
| 输入参数{in}   |                                       |
| spi_periph | 外设SPIx                                |
| SPIx       | x=0,1                                 |
| 输出参数{out}  |                                       |
| -          | -                                     |
| 返回值        |                                       |
| -          | -                                     |

例如:

```
/* turn on SPI0 CRC function */
```

```
spi_crc_on(SPI0);
```

### 函数 spi\_crc\_off

函数spi\_crc\_off描述见下表:

表 3-613. 函数 spi\_crc\_off

|            |  |
|------------|--|
| 函数名称       | spi_crc_off                            |
| 函数原形       | void spi_crc_off(uint32_t spi_periph); |
| 功能描述       | 关闭外设SPI的CRC功能                          |
| 先决条件       | -                                      |
| 被调用函数      | -                                      |
| 输入参数{in}   |  |
| spi_periph | 外设SPIx                                 |
| SPIx       | x=0,1                                  |
| 输出参数{out}  |  |
| -          | -                                      |
| 返回值        |  |
| -          | -                                      |

例如:

```
/* turn off SPI0 CRC function */
```

```
spi_crc_off(SPI0);
```

### 函数 spi\_crc\_next

函数spi\_crc\_next描述见下表:

表 3-614. 函数 spi\_crc\_next

|            |   |
|------------|---|
| 函数名称       | spi_crc_next                            |
| 函数原形       | void spi_crc_next(uint32_t spi_periph); |
| 功能描述       | 设置外设SPI下一次传输数据为CRC值                     |
| 先决条件       | -                                       |
| 被调用函数      | -                                       |
| 输入参数{in}   |   |
| spi_periph | 外设SPIx                                  |
| SPIx       | x=0,1                                   |
| 输出参数{out}  |   |
| -          | -                                       |
| 返回值        |   |
| -          | -                                       |

例如：

```
/* SPI0 next data is CRC value */
```

```
spi_crc_next(SPI0);
```

### 函数 spi\_crc\_get

函数spi\_crc\_get描述见下表：

表 3-615. 函数 spi\_crc\_get

|            |   |
|------------|---|
| 函数名称       | spi_crc_get   |
| 函数原形       | uint16_t spi_crc_get(uint32_t spi_periph, uint8_t crc); |
| 功能描述       | 外设SPI获取CRC值   |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| spi_periph | 外设SPIx  |
| SPIx       | x=0,1   |
| 输入参数{in}   |   |
| crc        | SPI crc值  |
| SPI_CRC_TX | 获取发送CRC寄存器值   |
| SPI_CRC_RX | 获取接收CRC寄存器值   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| uint16_t   | 16位CRC值（0-0xFFFF）                                       |

例如：

```
/* get SPI0 CRC send value */
```



```
uint16_t crc_val;  
  
crc_val = spi_crc_get(SPI0, SPI_CRC_TX);
```

### 函数 spi\_crc\_error\_clear

函数spi\_crc\_error\_clear描述见下表:

表 3-616. 函数 spi\_crc\_error\_clear

|            |  |
|------------|--|
| 函数名称       | spi_crc_error_clear                            |
| 函数原形       | void spi_crc_error_clear(uint32_t spi_periph); |
| 功能描述       | 清除SPI CRC错误标志状态                                |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| spi_periph | 外设SPIx   |
| SPIx       | x=0,1  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如:

```
/* clear SPI0 CRC error flag status */  
  
spi_crc_error_clear(SPI0);
```

### 函数 spi\_ti\_mode\_enable

函数spi\_ti\_mode\_enable描述见下表:

表 3-617. 函数 spi\_ti\_mode\_enable

|            |   |
|------------|---|
| 函数名称       | spi_ti_mode_enable                            |
| 函数原形       | void spi_ti_mode_enable(uint32_t spi_periph); |
| 功能描述       | 使能SPI TI模式                                    |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| spi_periph | 外设SPIx  |
| SPIx       | x=0,1   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如:

```
/* enable SPI0 TI mode */
```

```
spi_ti_mode_enable(SPI0);
```

### 函数 spi\_ti\_mode\_disable

函数spi\_ti\_mode\_disable描述见下表：

表 3-618. 函数 spi\_ti\_mode\_disable

|            |  |
|------------|--|
| 函数名称       | spi_ti_mode_disable                            |
| 函数原形       | void spi_ti_mode_disable(uint32_t spi_periph); |
| 功能描述       | 失能SPI TI模式                                     |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| spi_periph | 外设SPIx   |
| SPIx       | x=0,1  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如：

```
/* disable SPI0 TI mode */
```

```
spi_ti_mode_disable(SPI0);
```

### 函数 spi\_nssp\_mode\_enable

函数spi\_nssp\_mode\_enable描述见下表：

表 3-619. 函数 spi\_nssp\_mode\_enable

|            |   |
|------------|---|
| 函数名称       | spi_nssp_mode_enable                            |
| 函数原形       | void spi_nssp_mode_enable(uint32_t spi_periph); |
| 功能描述       | 使能SPI NSS脉冲模式                                   |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| spi_periph | 外设SPIx  |
| SPIx       | x=0,1   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如：

```
/* enable SPI0 NSS pulse mode */
```

```
spi_nssp_mode_enable(SPI0);
```

### 函数 spi\_nssp\_mode\_disable

函数spi\_nssp\_mode\_disable描述见下表:

表 3-620. 函数 spi\_nssp\_mode\_disable

|            |  |
|------------|--|
| 函数名称       | spi_nssp_mode_disable                            |
| 函数原形       | void spi_nssp_mode_disable(uint32_t spi_periph); |
| 功能描述       | 失能SPI NSS脉冲模式                                    |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| spi_periph | 外设SPIx   |
| SPIx       | x=0,1  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如:

```
/* disable SPI0 NSS pulse mode */
```

```
spi_nssp_mode_disable(SPI0);
```

### 函数 spi\_quad\_enable

函数spi\_quad\_enable描述见下表:

表 3-621. 函数 spi\_quad\_enable

|            |  |
|------------|--|
| 函数名称       | spi_quad_enable                            |
| 函数原形       | void spi_quad_enable(uint32_t spi_periph); |
| 功能描述       | 使能四线SPI模式                                  |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| spi_periph | 外设SPIx                                     |
| SPIx       | x=0  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如:

```
/* enable SPI0 quad wire mode */
```

```
spi_quad_enable(SPI0);
```

### 函数 spi\_quad\_disable

函数spi\_quad\_disable描述见下表:

表 3-622. 函数 spi\_quad\_disable

|            |   |
|------------|---|
| 函数名称       | spi_quad_disable                            |
| 函数原形       | void spi_quad_disable(uint32_t spi_periph); |
| 功能描述       | 失能四线SPI模式                                   |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| spi_periph | 外设SPIx                                      |
| SPIx       | x=0   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如:

```
/* disable SPI0 quad wire mode */
```

```
spi_quad_disable(SPI0);
```

### 函数 spi\_quad\_write\_enable

函数spi\_quad\_write\_enable描述见下表:

表 3-623. 函数 spi\_quad\_write\_enable

|            |  |
|------------|--|
| 函数名称       | spi_quad_write_enable                            |
| 函数原形       | void spi_quad_write_enable(uint32_t spi_periph); |
| 功能描述       | 使能四线SPI写   |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| spi_periph | 外设SPIx   |
| SPIx       | x=0  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如:

```
/* enable SPI0 quad wire write */
```

```
spi_quad_write_enable(SPI0);
```

### 函数 spi\_quad\_read\_enable

函数spi\_quad\_read\_enable描述见下表：

表 3-624. 函数 spi\_quad\_read\_enable

|            |   |
|------------|---|
| 函数名称       | spi_quad_read_enable                            |
| 函数原形       | void spi_quad_read_enable(uint32_t spi_periph); |
| 功能描述       | 使能四线SPI读  |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| spi_periph | 外设SPIx  |
| SPIx       | x=0   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如：

```
/* enable SPI0 quad wire read */
```

```
spi_quad_read_enable(SPI0);
```

### 函数 spi\_quad\_io23\_output\_enable

函数spi\_quad\_io23\_output\_enable描述见下表：

表 3-625. 函数 spi\_quad\_io23\_output\_enable

|            |  |
|------------|--|
| 函数名称       | spi_quad_io23_output_enable                            |
| 函数原形       | void spi_quad_io23_output_enable(uint32_t spi_periph); |
| 功能描述       | 使能SPI_IO2和SPI_IO3输出                                    |
| 先决条件       | -  |
| 被调用函数      | -  |
| 输入参数{in}   |  |
| spi_periph | 外设SPIx   |
| SPIx       | x=0  |
| 输出参数{out}  |  |
| -          | -  |
| 返回值        |  |
| -          | -  |

例如：

```
/* enable SPI0 SPI_IO2 and SPI_IO3 pin output */
```

```
spi_quad_io23_output_enable(SPI0);
```

### 函数 spi\_quad\_io23\_output\_disable

函数spi\_quad\_io23\_output\_disable描述见下表：

**表 3-626. 函数 spi\_quad\_io23\_output\_disable**

|            |   |
|------------|---|
| 函数名称       | spi_quad_io23_output_disable                            |
| 函数原形       | void spi_quad_io23_output_disable(uint32_t spi_periph); |
| 功能描述       | 失能SPI_IO2和SPI_IO3输出                                     |
| 先决条件       | -   |
| 被调用函数      | -   |
| 输入参数{in}   |   |
| spi_periph | 外设SPIx  |
| SPIx       | x=0   |
| 输出参数{out}  |   |
| -          | -   |
| 返回值        |   |
| -          | -   |

例如：

```
/* disable SPI0 SPI_IO2 and SPI_IO3 pin output */
```

```
spi_quad_io23_output_disable(SPI0);
```

### 函数 spi\_i2s\_interrupt\_enable

函数spi\_i2s\_interrupt\_enable描述见下表：

**表 3-627. 函数 spi\_i2s\_interrupt\_enable**

|                  |  |
|------------------|--|
| 函数名称             | spi_i2s_interrupt_enable   |
| 函数原形             | void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t interrupt); |
| 功能描述             | 使能外设SPI/I2S中断  |
| 先决条件             | -  |
| 被调用函数            | -  |
| 输入参数{in}         |  |
| spi_periph       | 外设SPIx   |
| SPIx             | x=0,1  |
| 输入参数{in}         |  |
| interrupt        | SPI/I2S中断  |
| SPI_I2S_INT_TBE  | 发送缓冲区空中断   |
| SPI_I2S_INT_RBNE | 接收缓冲区非空中断  |
| SPI_I2S_INT_ERR  | 错误中断使能   |

| 输出参数{out} |   |
|-----------|---|
| -         | - |
| 返回值       |   |
| -         | - |

例如：

```
/* enable SPI0 transmit buffer empty interrupt */
spi_i2s_interrupt_enable(SPI0, SPI_I2S_INT_TBE);
```

### 函数 spi\_i2s\_interrupt\_disable

函数spi\_i2s\_interrupt\_disable描述见下表：

表 3-628. 函数 spi\_i2s\_interrupt\_disable

| 函数名称             | spi_i2s_interrupt_disable   |
|------------------|---|
| 函数原形             | void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt); |
| 功能描述             | 失能外设SPI/I2S中断   |
| 先决条件             | -   |
| 被调用函数            | -   |
| 输入参数{in}         |   |
| spi_periph       | 外设SPIx  |
| SPIx             | x=0,1   |
| 输入参数{in}         |   |
| interrupt        | SPI/I2S中断   |
| SPI_I2S_INT_TBE  | 发送缓冲区空中断  |
| SPI_I2S_INT_RBNE | 接收缓冲区非空中断   |
| SPI_I2S_INT_ERR  | 错误中断使能  |
| 输出参数{out}        |   |
| -                | -   |
| 返回值              |   |
| -                | -   |

例如：

```
/* disable SPI0 transmit buffer empty interrupt */
spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_TBE);
```

### 函数 spi\_i2s\_interrupt\_flag\_get

函数spi\_i2s\_interrupt\_flag\_get描述见下表：

表 3-629. 函数 spi\_i2s\_interrupt\_flag\_get

| 函数名称 | spi_i2s_interrupt_flag_get   |
|------|--|
| 函数原形 | FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt); |

|                          |                 |
|--------------------------|-----------------|
| 功能描述                     | 获取外设SPI/I2S中断状态 |
| 先决条件                     | -               |
| 被调用函数                    | -               |
| 输入参数{in}                 |                 |
| spi_periph               | 外设SPIx          |
| SPIx                     | x=0,1           |
| 输入参数{in}                 |                 |
| interrupt                | SPI/I2S中断状态     |
| SPI_I2S_INT_FLAG_TBE     | 发送缓冲区空中断        |
| SPI_I2S_INT_FLAG_RBNE    | 接收缓冲区非空中断       |
| SPI_I2S_INT_FLAG_RXORERR | 接收过载错误中断        |
| SPI_INT_FLAG_CONFERR     | 配置错误中断          |
| SPI_INT_FLAG_CRCERR      | CRC错误中断         |
| I2S_INT_FLAG_TXURERR     | 发送欠载错误中断        |
| SPI_I2S_INT_FLAG_FERR    | 格式错误中断          |
| 输出参数{out}                |                 |
| -                        | -               |
| 返回值                      |                 |
| FlagStatus               | SET 或 RESET     |

例如：

```

/* get SPI0 transmit buffer empty interrupt status */
if(RESET != spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE)){
    while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
    spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
}

```

### 函数 spi\_i2s\_flag\_get

函数spi\_i2s\_flag\_get描述见下表：

表 3-630. 函数 spi\_i2s\_flag\_get

|      |  |
|------|--|
| 函数名称 | spi_i2s_flag_get   |
| 函数原形 | FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag); |
| 功能描述 | 获取外设SPI/I2S标志状态  |



|                        |                 |
|------------------------|-----------------|
| 先决条件                   | -               |
| 被调用函数                  | -               |
| 输入参数{in}               |                 |
| <b>spi_periph</b>      | 外设SPIx          |
| SPIx                   | x=0,1           |
| 输入参数{in}               |                 |
| <b>flag</b>            | SPI/I2S标志状态     |
| SPI_FLAG_TBE           | SPI发送缓冲区空标志     |
| SPI_FLAG_RBNE          | SPI接收缓冲区非空标志    |
| SPI_FLAG_TRANS         | SPI通信进行中标志      |
| SPI_FLAG_RXORERR       | SPI接收过载错误标志     |
| SPI_FLAG_CONFERR       | SPI配置错误标志       |
| SPI_FLAG_CRCERR        | SPI CRC错误标志     |
| SPI_FLAG_FERR          | SPI格式错误标志       |
| I2S_FLAG_TBE           | I2S发送缓冲区空标志     |
| I2S_FLAG_RBNE          | I2S接收缓冲区非空标志    |
| I2S_FLAG_TRANS         | I2S通信进行中标志      |
| I2S_FLAG_RXORERR       | I2S接收过载错误标志     |
| I2S_FLAG_TXURERR       | I2S发送欠载错误标志     |
| I2S_FLAG_CH            | I2S通道标志         |
| I2S_FLAG_FERR          | I2S格式错误标志       |
| 以下参数只适用于SPI0           |                 |
| SPI_TXLVL_EMPTY        | SPI TXFIFO空     |
| SPI_TXLVL_QUARTER_FULL | SPI TXFIFO四分之一满 |
| SPI_TXLVL_HALF_FULL    | SPI TXFIFO半满    |
| SPI_TXLVL_FULL         | TXFIFO全满        |
| SPI_RXLVL_EMPTY        | SPI RXFIFO空     |
| SPI_RXLVL_QUARTER_FULL | SPI RXFIFO四分之一满 |
| SPI_RXLVL_HALF_FULL    | SPI RXFIFO半满    |
| SPI_RXLVL_FULL         | RXFIFO全满        |
| 输出参数{out}              |                 |

|            |             |
|------------|-------------|
| -          | -           |
| 返回值        |             |
| FlagStatus | SET 或 RESET |

例如:

```
/* get SPI0 transmit buffer empty flag status */
while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
```

## 3.21. SYSCFG

章节 [3.21.1](#) 描述了 SYSCFG 的寄存器列表，章节 [3.21.2](#) 对 SYSCFG 库函数进行说明。

### 3.21.1. 外设寄存器说明

SYSCFG 寄存器列表如下表所示:

表 3-631. SYSCFG 寄存器

| 寄存器名称             | 寄存器描述         |
|-------------------|---------------|
| SYSCFG_CFG0       | 配置寄存器0        |
| SYSCFG_CFG1       | 配置寄存器1        |
| SYSCFG_EXTISS0    | EXTI源选择寄存器0   |
| SYSCFG_EXTISS1    | EXTI源选择寄存器1   |
| SYSCFG_EXTISS2    | EXTI源选择寄存器2   |
| SYSCFG_EXTISS3    | EXTI源选择寄存器3   |
| SYSCFG_CFG2       | 配置寄存器2        |
| SYSCFG_STAT       | 状态寄存器         |
| SYSCFG_CFG3       | 配置寄存器3        |
| SYSCFG_TIMERINSEL | TIMER输入源选择寄存器 |

### 3.21.2. 外设库函数说明

SYSCFG 库函数列表如下表所示:

表 3-632. SYSCFG 库函数

| 库函数名称                      | 库函数描述          |
|----------------------------|----------------|
| syscfg_deinit              | 复位SYSCFG寄存器    |
| syscfg_exti_line_config    | 配置GPIO引脚作为EXTI |
| syscfg_pin_remap_enable    | 使能引脚重映射功能      |
| syscfg_pin_remap_disable   | 禁能引脚重映射功能      |
| syscfg_adc_ch_remap_config | 配置ADC通道重映射GPIO |

| 库函数名称                            | 库函数描述                       |
|----------------------------------|-----------------------------|
| syscfg_timer_eti_sel             | 选择TIMER外部触发源                |
| syscfg_timer_bkin_select_trigsel | 选择TRIGSEL作为TIMER break输入源   |
| syscfg_timer_bkin_select_gpio    | 选择GPIO作为TIMER break输入源      |
| syscfg_timer7_ch0n_select        | 选择TIMER7通道0补充输入源            |
| syscfg_lock_config               | 配置TIMER0/7/19/20 break输入连接源 |
| syscfg_flag_get                  | 获取SYSCFG状态                  |
| syscfg_flag_clear                | 清除SYSCFG状态                  |
| syscfg_interrupt_enable          | 使能SYSCFG中断                  |
| syscfg_interrupt_disable         | 禁能SYSCFG中断                  |
| syscfg_interrupt_flag_get        | 获取SYSCFG中断状态                |
| syscfg_bootmode_get              | 获取当前BOOT启动方式                |
| syscfg_sram_ecc_address_get      | 获取SRAM ECC错误发生地址            |
| syscfg_sram_ecc_bit_get          | 获取单比特SRAM ECC错误比特           |

### 函数 syscfg\_deinit

函数syscfg\_deinit描述见下表：

表 3-633. 函数 syscfg\_deinit

|           |  |
|-----------|--|
| 函数名称      | syscfg_deinit                                      |
| 函数原形      | void syscfg_deinit(void);                          |
| 功能描述      | 复位SYSCFG寄存器  |
| 先决条件      | -  |
| 被调用函数     | rcu_periph_reset_enable / rcu_periph_reset_disable |
| 输入参数{in}  |  |
| -         | -  |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如：

```
/* reset SYSCFG registers */
syscfg_deinit();
```

### 函数 syscfg\_exti\_line\_config

函数syscfg\_exti\_line\_config描述见下表：

表 3-634. 函数 syscfg\_exti\_line\_config

|      |  |
|------|--|
| 函数名称 | syscfg_exti_line_config  |
| 函数原形 | void syscfg_exti_line_config(uint8_t exti_port, uint8_t exti_pin); |
| 功能描述 | 配置GPIO引脚作为EXTI   |

|                   |   |
|-------------------|---|
| 先决条件              | -   |
| 被调用函数             | -   |
| 输入参数{in}          |   |
| exti_port         | 指定EXTI使用的GPIO端口   |
| EXTI_SOURCE_GPIOx | x = A,B,C,D,E,F   |
| 输入参数{in}          |   |
| exti_pin          | EXTI引脚  |
| EXTI_SOURCE_PINx  | 对于GPIOA\GPIOB\GPIOC\GPIOD\GPIOE, x = 0..15, 对于GPIOF, x = 0..7 |
| 输出参数{out}         |   |
| -                 | -   |
| 返回值               |   |
| -                 | -   |

例如:

```
/* configure the PA0 pin as EXTI Line */
```

```
syscfg_exti_line_config(EXTI_SOURCE_GPIOA, EXTI_SOURCE_PIN0);
```

### 函数 syscfg\_pin\_remap\_enable

函数syscfg\_pin\_remap\_enable描述见下表:

表 3-635. 函数 syscfg\_pin\_remap\_enable

|                        |   |
|------------------------|---|
| 函数名称                   | syscfg_pin_remap_enable                           |
| 函数原形                   | void syscfg_pin_remap_enable(uint32_t remap_pin); |
| 功能描述                   | 使能引脚重映射功能   |
| 先决条件                   | -   |
| 被调用函数                  | -   |
| 输入参数{in}               |   |
| remap_pin              | 引脚重映射   |
| SYSCFG_PA9_PA12_REMAP  | PA9/PA12重映射在PA10/PA11                             |
| SYSCFG_BOOT0_REMAP_PF0 | PF0重映射在BOOT0                                      |
| 输出参数{out}              |   |
| -                      | -   |
| 返回值                    |   |
| -                      | -   |

例如:

```
/* enable BOOT0 remap to PF0 function */
```

```
syscfg_pin_remap_enable(SYSCFG_BOOT0_REMAP_PF0);
```

**函数 syscfg\_pin\_remap\_disable**

函数syscfg\_pin\_remap\_disable描述见下表：

**表 3-636. 函数 syscfg\_pin\_remap\_disable**

|                        |  |
|------------------------|--|
| 函数名称                   | syscfg_pin_remap_disable                           |
| 函数原形                   | void syscfg_pin_remap_disable(uint32_t remap_pin); |
| 功能描述                   | 禁用引脚重映射功能  |
| 先决条件                   | -  |
| 被调用函数                  | -  |
| 输入参数{in}               |  |
| remap_pin              | 引脚重映射  |
| SYSCFG_PA9_PA12_REMAP  | PA9/PA12重映射在PA10/PA11                              |
| SYSCFG_BOOT0_REMAP_PF0 | PF0重映射在BOOT0                                       |
| 输出参数{out}              |  |
| -                      | -  |
| 返回值                    |  |
| -                      | -  |

例如：

```
/* disable BOOT0 remap to PF0 function */
```

```
syscfg_pin_remap_disable(SYSCFG_BOOT0_REMAP_PF0);
```

**函数 syscfg\_adc\_ch\_remap\_config**

函数syscfg\_adc\_ch\_remap\_config描述见下表：

**表 3-637. 函数 syscfg\_adc\_ch\_remap\_config**

|                 |   |
|-----------------|---|
| 函数名称            | syscfg_adc_ch_remap_config  |
| 函数原形            | void syscfg_adc_ch_remap_config(syscfg_adc_ch_enum adcx_iny_remap, ControlStatus newvalue); |
| 功能描述            | 配置ADC通道重映射GPIO  |
| 先决条件            | -   |
| 被调用函数           | -   |
| 输入参数{in}        |   |
| adcx_iny_remap  | ADC通道   |
| ADC1_IN14_REMAP | ADC1通道14 GPIO重映射  |
| ADC1_IN15_REMAP | ADC1通道15 GPIO重映射  |
| ADC0_IN8_REMAP  | ADC0通道8 GPIO重映射   |
| ADC0_IN9_REMAP  | ADC0通道9 GPIO重映射   |

| 输出参数{out} |   |
|-----------|---|
| -         | - |
| 返回值       |   |
| -         | - |

例如：

```
/* enable ADC1 channel 14 GPIO pin remap function */
syscfg_adc_ch_remap_config (ADC1_IN14_REMAP, ENABLE);
```

### 函数 syscfg\_timer\_eti\_sel

函数syscfg\_timer\_eti\_sel描述见下表：

**表 3-638. 函数 syscfg\_timer\_eti\_sel**

| 函数名称               | syscfg_timer_eti_sel   |
|--------------------|--|
| 函数原形               | void syscfg_timer_eti_sel(syscfg_timersel_enum timer_num, uint32_t eti_num); |
| 功能描述               | 选择TIMER外部触发源   |
| 先决条件               | -  |
| 被调用函数              | -  |
| 输入参数{in}           |  |
| timer_num          | TIMER  |
| TIMER0SEL          | 选择TIMER0   |
| TIMER7SEL          | 选择TIMER7   |
| TIMER19SEL         | 选择TIMER19  |
| TIMER20SEL         | 选择TIMER20  |
| 输入参数{in}           |  |
| etr_num            | 外部触发源  |
| TIMER_ETI_TRGx     | TIMER外部触发源x, x = 0,1,2,3   |
| TIMER_ETI_TRG_NONE | 不选择TIMER外部触发源  |
| 输出参数{out}          |  |
| -                  | -  |
| 返回值                |  |
| -                  | -  |

例如：

```
/* select TIMER0 external trigger source 0 */
syscfg_timer_eti_sel (TIMER0SEL, TIMER_ETI_TRG0);
```

### 函数 syscfg\_timer\_bkin\_select\_trigsel

函数syscfg\_timer\_bkin\_select\_trigsel描述见下表：

表 3-639. 函数 syscfg\_timer\_bkin\_select\_trigsel

|                   |  |
|-------------------|--|
| 函数名称              | syscfg_timer_bkin_select_trigsel                             |
| 函数原形              | void syscfg_timer_bkin_select_trigsel(uint32_t bkin_source); |
| 功能描述              | 选择TRIGSEL作为TIMER break输入源                                    |
| 先决条件              | -  |
| 被调用函数             | -  |
| 输入参数{in}          |  |
| bkin_source       | TIMER break输入源   |
| TIMERx_BKINy_TRIG | 从TRIGSEL选择TIMERx的break输入源y, x=0,7,19,20 y=0,1,2,3            |
| 输出参数{out}         |  |
| -                 | -  |
| 返回值               |  |
| -                 | -  |

例如:

```
/* select TRIGSEL as TIMER0 break input source 0 */
syscfg_timer_bkin_select_trigsel(TIMER0_BKIN0_TRIG);
```

### 函数 syscfg\_timer\_bkin\_select\_gpio

函数syscfg\_timer\_bkin\_select\_gpio描述见下表:

表 3-640. 函数 syscfg\_timer\_bkin\_select\_gpio

|                   |   |
|-------------------|---|
| 函数名称              | syscfg_timer_bkin_select_gpio                             |
| 函数原形              | void syscfg_timer_bkin_select_gpio(uint32_t bkin_source); |
| 功能描述              | 选择GPIO作为TIMER break输入源                                    |
| 先决条件              | -   |
| 被调用函数             | -   |
| 输入参数{in}          |   |
| bkin_source       | TIMER break输入源  |
| TIMERx_BKINy_TRIG | 从TRIGSEL选择TIMERx的break输入源y, x=0,7,19,20 y=0,1,2,3         |
| 输出参数{out}         |   |
| -                 | -   |
| 返回值               |   |
| -                 | -   |

例如:

```
/* select GPIO as TIMER0 break input source 0 */
syscfg_timer_bkin_select_gpio(TIMER0_BKIN0_TRIG);
```

**函数 syscfg\_timer7\_ch0n\_select**

函数syscfg\_timer7\_ch0n\_select描述见下表：

**表 3-641. 函数 syscfg\_timer7\_ch0n\_select**

|                                   |  |
|-----------------------------------|--|
| 函数名称                              | syscfg_timer7_ch0n_select                                |
| 函数原形                              | void syscfg_timer7_ch0n_select(uint32_t timer7_ch0n_in); |
| 功能描述                              | 选择TIMER7通道0补充输入源   |
| 先决条件                              | -  |
| 被调用函数                             | -  |
| 输入参数{in}                          |  |
| timer7_ch0n_in                    | TIMER7通道0补充输入源   |
| TIMER7CH0N_TIMER7CH0_TIMER0CH0_IN | 选择TIMER7_CH0_IN, TIMER7_CH0N_IN和TIMER0_CH0_IN的异或         |
| TIMER7_CH0N_IN                    | 选择TIMER7_CH0N_IN   |
| 输出参数{out}                         |  |
| -                                 | -  |
| 返回值                               |  |
| -                                 | -  |

例如：

```
/* select TIMER7 channel0 complementary input source */
```

```
syscfg_timer7_ch0n_select (TIMER7_CH0N_IN);
```

**函数 syscfg\_lock\_config**

函数syscfg\_lock\_config描述见下表：

**表 3-642. 函数 syscfg\_lock\_config**

|                            |  |
|----------------------------|--|
| 函数名称                       | syscfg_lock_config                             |
| 函数原形                       | void syscfg_lock_config(uint32_t syscfg_lock); |
| 功能描述                       | 配置TIMER0/7/19/20 break输入连接源                    |
| 先决条件                       | -  |
| 被调用函数                      | -  |
| 输入参数{in}                   |  |
| syscfg_lock                | 选定的连接源   |
| SYSCFG_LOCK_LOCKUP         | Cortex-M33 LOCKUP输出与TIMER0/7/19/20的break输入端连接  |
| SYSCFG_LOCK_SRAM_ECC_ERROR | SRAM ECC校验错误与TIMER0/7/19/20的break输入端连接         |
| SYSCFG_LOCK_LVD            | LVD中断与TIMER0/7/19/20的break输入端连接                |
| 输出参数{out}                  |  |



|     |   |
|-----|---|
| -   | - |
| 返回值 |   |
| -   | - |

例如：

```
/* configure Cortex-M33 lockup output connected to the break input */
```

```
syscfg_lock_config(SYSCFG_LOCK_LOCKUP);
```

### 函数 syscfg\_flag\_get

函数syscfg\_flag\_get描述见下表：

表 3-643. 函数 syscfg\_flag\_get

|                             |  |
|-----------------------------|--|
| 函数名称                        | syscfg_flag_get                            |
| 函数原形                        | FlagStatus syscfg_flag_get(uint32_t flag); |
| 功能描述                        | 获取SYSCFG状态                                 |
| 先决条件                        | -  |
| 被调用函数                       | -  |
| 输入参数{in}                    |  |
| flag                        | 选定的SYSCFG_STAT中待检查的状态                      |
| SYSCFG_FLAG_S<br>RAMECCMERR | SRAM多比特不可纠错ECC错误状态                         |
| SYSCFG_FLAG_S<br>RAMECCSERR | SRAM单比特可纠错ECC错误状态                          |
| SYSCFG_FLAG_FL<br>ASHECCERR | FLASH ECC NMI错误状态                          |
| SYSCFG_FLAG_C<br>KMNMERR    | HXTAL时钟监测NMI错误状态                           |
| SYSCFG_FLAG_N<br>MIPINERR   | NMI引脚错误状态                                  |
| 输出参数{out}                   |  |
| -                           | -  |
| 返回值                         |  |
| FlagStatus                  | SET或RESET                                  |

例如：

```
/* get SRAM multi-bits non-correction ECC error flag */
```

```
FlagStatus flag;
```

```
flag = syscfg_flag_get (SYSCFG_FLAG_SRAMECCMERR);
```

### 函数 syscfg\_flag\_clear

函数syscfg\_flag\_clear描述见下表：

表 3-644. 函数 syscfg\_flag\_clear

|                             |  |
|-----------------------------|--|
| 函数名称                        | syscfg_flag_clear                      |
| 函数原形                        | void syscfg_flag_clear(uint32_t flag); |
| 功能描述                        | 清除SYSCFG状态                             |
| 先决条件                        | -                                      |
| 被调用函数                       | -                                      |
| 输入参数{in}                    |  |
| flag                        | 选定的SYSCFG_STAT中待检查的状态                  |
| SYSCFG_FLAG_S<br>RAMECCMERR | SRAM多比特不可纠错ECC错误状态                     |
| SYSCFG_FLAG_S<br>RAMECCSERR | SRAM单比特可纠错ECC错误状态                      |
| SYSCFG_FLAG_FL<br>ASHECCERR | FLASH ECC NMI错误状态                      |
| SYSCFG_FLAG_C<br>KMNMIERR   | HXTAL时钟监测NMI错误状态                       |
| 输出参数{out}                   |  |
| -                           | -                                      |
| 返回值                         |  |
| -                           | -                                      |

例如：

```
/* clear SRAM multi-bits non-correction ECC error flag */
```

```
syscfg_flag_clear (SYSCFG_FLAG_SRAM_ECCMERR);
```

### 函数 syscfg\_interrupt\_enable

函数syscfg\_interrupt\_enable描述见下表：

表 3-645. 函数 syscfg\_interrupt\_enable

|                          |   |
|--------------------------|---|
| 函数名称                     | syscfg_interrupt_enable                           |
| 函数原形                     | void syscfg_interrupt_enable(uint32_t interrupt); |
| 功能描述                     | 使能SYSCFG中断  |
| 先决条件                     | -   |
| 被调用函数                    | -   |
| 输入参数{in}                 |   |
| interrupt                | 选定的SYSCFG_CFG3中中断                                 |
| SYSCFG_INT_SRA<br>MECCME | SRAM多比特不可纠错ECC错误中断                                |
| SYSCFG_INT_SRA<br>MECCSE | SRAM单比特可纠错ECC错误中断                                 |
| SYSCFG_INT_FL<br>SHECCE  | FLASH ECC NMI错误中断                                 |

|                               |                  |
|-------------------------------|------------------|
| <i>SYSCFG_INT_CKM<br/>NMI</i> | HXTAL时钟监测NMI错误中断 |
| <i>SYSCFG_INT_NMI<br/>PIN</i> | NMI引脚错误中断        |
| 输出参数{out}                     |                  |
| -                             | -                |
| 返回值                           |                  |
| -                             | -                |

例如：

```
/* enable SRAM multi-bits non-correction ECC error interrupt */
```

```
syscfg_interrupt_enable (SYSCFG_INT_SRAM_ECCME);
```

### 函数 **syscfg\_interrupt\_disable**

函数syscfg\_interrupt\_disable描述见下表：

**表 3-646. 函数 syscfg\_interrupt\_disable**

|                                  |  |
|----------------------------------|--|
| 函数名称                             | syscfg_interrupt_disable                           |
| 函数原形                             | void syscfg_interrupt_disable(uint32_t interrupt); |
| 功能描述                             | 禁用SYSCFG中断   |
| 先决条件                             | -  |
| 被调用函数                            | -  |
| 输入参数{in}                         |  |
| <b>interrupt</b>                 | 选定的SYSCFG_CFG3中中断                                  |
| <i>SYSCFG_INT_SRA<br/>MECCME</i> | SRAM多比特不可纠错ECC错误中断                                 |
| <i>SYSCFG_INT_SRA<br/>MECCSE</i> | SRAM单比特可纠错ECC错误中断                                  |
| <i>SYSCFG_INT_FLA<br/>SHECCE</i> | FLASH ECC NMI错误中断                                  |
| <i>SYSCFG_INT_CKM<br/>NMI</i>    | HXTAL时钟监测NMI错误中断                                   |
| <i>SYSCFG_INT_NMI<br/>PIN</i>    | NMI引脚错误中断  |
| 输出参数{out}                        |  |
| -                                | -  |
| 返回值                              |  |
| -                                | -  |

例如：

```
/* disable SRAM multi-bits non-correction ECC error interrupt */
```

```
syscfg_interrupt_disable (SYSCFG_INT_SRAM_ECCME);
```

**函数 syscfg\_interrupt\_flag\_get**

函数syscfg\_interrupt\_flag\_get描述见下表:

**表 3-647. 函数 syscfg\_interrupt\_flag\_get**

|                                |   |
|--------------------------------|---|
| 函数名称                           | syscfg_interrupt_flag_get                                 |
| 函数原形                           | FlagStatus syscfg_interrupt_flag_get(uint32_t interrupt); |
| 功能描述                           | 获取SYSCFG中断状态  |
| 先决条件                           | -   |
| 被调用函数                          | -   |
| <b>输入参数{in}</b>                |   |
| <b>interrupt</b>               | 选定的SYSCFG_CFG3中中断   |
| SYSCFG_INT_FLAG_SRAM_ECC_MER_R | SRAM多比特不可纠错ECC错误中断状态                                      |
| SYSCFG_INT_FLAG_SRAM_ECC_SER   | SRAM单比特可纠错ECC错误中断状态                                       |
| SYSCFG_INT_FLAG_FLASH_ECC_ERR  | FLASH ECC NMI错误中断状态                                       |
| SYSCFG_INT_FLAG_CKMNMI_ERR     | HXTAL时钟监测NMI错误中断状态  |
| SYSCFG_INT_FLAG_NMIPIN_ERR     | NMI引脚错误中断状态   |
| <b>输出参数{out}</b>               |   |
| -                              | -   |
| <b>返回值</b>                     |   |
| <b>FlagStatus</b>              | SET或RESET   |

例如:

```
/* get SRAM multi-bits non-correction ECC error interrupt flag */
```

```
FlagStatus flag;
```

```
flag = syscfg_interrupt_flag_get (SYSCFG_INT_FLAG_SRAM_ECC_MERR);
```

**函数 syscfg\_bootmode\_get**

函数syscfg\_bootmode\_get描述见下表:

**表 3-648. 函数 syscfg\_bootmode\_get**

|       |                                    |
|-------|------------------------------------|
| 函数名称  | syscfg_bootmode_get                |
| 函数原形  | uint8_t syscfg_bootmode_get(void); |
| 功能描述  | 获取当前BOOT启动方式                       |
| 先决条件  | -                                  |
| 被调用函数 | -                                  |

| 输入参数{in}               |                 |
|------------------------|-----------------|
| -                      | -               |
| 输出参数{out}              |                 |
| -                      | -               |
| 返回值                    |                 |
| uint8_t                | 启动方式            |
| SYSCFG_BOOTMODE_FLASH  | 从片上闪存的主存引导启动    |
| SYSCFG_BOOTMODE_SYSTEM | 从片上闪存的系统存储器引导启动 |
| SYSCFG_BOOTMODE_SRAM   | 从片上SRAM引导启动     |

例如：

```
/* get the current boot mode */
```

```
uint8_t boot_mode;
```

```
boot_mode = syscfg_bootmode_get();
```

### 函数 syscfg\_sram\_ecc\_address\_get

函数syscfg\_sram\_ecc\_address\_get描述见下表：

**表 3-649. 函数 syscfg\_sram\_ecc\_address\_get**

| 函数名称      | syscfg_sram_ecc_address_get                 |
|-----------|---|
| 函数原形      | uint16_t syscfg_sram_ecc_address_get(void); |
| 功能描述      | 获取SRAM ECC错误发生地址                            |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| -         | -   |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| uint16_t  | SRAM ECC错误发生地址， 0x0000 – 0xFFFF             |

例如：

```
/* get the address where SRAM ECC error occur on */
```

```
uint16_t sram_ecc_addr;
```

```
sram_ecc_addr = syscfg_sram_ecc_address_get();
```

**函数 syscfg\_sram\_ecc\_bit\_get**

函数syscfg\_sram\_ecc\_bit\_get描述见下表：

**表 3-650. 函数 syscfg\_sram\_ecc\_bit\_get**

|           |  |
|-----------|--|
| 函数名称      | syscfg_sram_ecc_bit_get                |
| 函数原形      | uint8_t syscfg_sram_ecc_bit_get(void); |
| 功能描述      | 获取单比特SRAM ECC错误比特                      |
| 先决条件      | -                                      |
| 被调用函数     | -                                      |
| 输入参数{in}  |  |
| -         | -                                      |
| 输出参数{out} |  |
| -         | -                                      |
| 返回值       |  |
| uint8_t   | 单比特SRAM ECC错误比特，0x00 – 0xFF            |

例如：

```
/* get the bit which has SRAM ECC signle error */
```

```
uint8_t sram_ecc_bit;
```

```
sram_ecc_bit = syscfg_sram_ecc_bit_get();
```

## 3.22. TIMER

定时器含有可编程的一个无符号计数器，支持输入捕获和输出比较，分为三种类型：高级定时器（TIMEx, x=0, 7, 19, 20），通用定时器L0（TIMEx, x=1），基本定时器（TIMEx, x=5, 6），不同类型的定时器具体功能有所差别。章节[3.22.1](#)描述了TIMER的寄存器列表，章节[3.22.2](#)对TIMER库函数进行说明。

### 3.22.1. 外设寄存器说明

TIMER寄存器列表如下表所示：

**表 3-651. TIMER 寄存器**

| 寄存器名称          | 寄存器描述       |
|----------------|-------------|
| TIMER_CTL0     | 控制寄存器0      |
| TIMER_CTL1     | 控制寄存器1      |
| TIMER_SMCFG    | 从模式配置寄存器    |
| TIMER_DMAINTEN | DMA和中断使能寄存器 |
| TIMER_INTF     | 中断标志寄存器     |
| TIMER_SWEVG    | 软件事件产生寄存器   |
| TIMER_CHCTL0   | 通道控制寄存器0    |

| 寄存器名称             | 寄存器描述                      |
|-------------------|----------------------------|
| TIMER_CHCTL1      | 通道控制寄存器1                   |
| TIMER_CHCTL2      | 通道控制寄存器2                   |
| TIMER_CNT         | 计数器寄存器                     |
| TIMER_PSC         | 预分频寄存器                     |
| TIMER_CAR         | 计数器自动重载寄存器                 |
| TIMER_CREP        | 重复计数寄存器                    |
| TIMER_CH0CV       | 通道0捕获/比较寄存器                |
| TIMER_CH1CV       | 通道1捕获/比较寄存器                |
| TIMER_CH2CV       | 通道2捕获/比较寄存器                |
| TIMER_CH3CV       | 通道3捕获/比较寄存器                |
| TIMER_CCHP        | 互补通道保护寄存器                  |
| TIMER_MCHCTL0     | TIMER多模式通道控制寄存器0           |
| TIMER_MCHCTL1     | TIMER多模式通道控制寄存器1           |
| TIMER_MCHCTL2     | TIMER多模式通道控制寄存器2           |
| TIMER_IRMP        | TIMER通道输入重映射寄存器（仅用于TIMER1） |
| TIMER_MCH0CV      | TIMER多模式通道0比较/捕获寄存器        |
| TIMER_MCH1CV      | TIMER多模式通道1比较/捕获寄存器        |
| TIMER_MCH2CV      | TIMER多模式通道2比较/捕获寄存器        |
| TIMER_MCH3CV      | TIMER多模式通道3比较/捕获寄存器        |
| TIMER_CH0COMV_ADD | TIMER通道0附加比较寄存器            |
| TIMER_CH1COMV_ADD | TIMER通道1附加比较寄存器            |
| TIMER_CH2COMV_ADD | TIMER通道2附加比较寄存器            |
| TIMER_CH3COMV_ADD | TIMER通道3附加比较寄存器            |
| TIMER_CTL2        | TIMER控制寄存器2                |
| TIMER_BRKCFG      | TIMER中止配置寄存器               |
| TIMER_FCCHP0      | TIMER独立互补通道捕获寄存器0          |
| TIMER_FCCHP1      | TIMER独立互补通道捕获寄存器1          |
| TIMER_FCCHP2      | TIMER独立互补通道捕获寄存器2          |
| TIMER_FCCHP3      | TIMER独立互补通道捕获寄存器3          |
| TIMER_DMACFG      | DMA配置寄存器                   |
| TIMER_DMATB       | DMA发送缓冲区寄存器                |
| TIMER_CFG         | 配置寄存器                      |

### 3.22.2. 外设库函数说明

TIMER库函数列表如下表所示：

表 3-652. TIMER 库函数

| 库函数名称                                      | 库函数描述                       |
|--|-----------------------------|
| timer_deinit                               | 复位外设TIMER                   |
| timer_struct_para_init                     | 将TIMER初始化结构体中所有参数初始化为默认值    |
| timer_init                                 | 初始化外设TIMER                  |
| timer_enable                               | 使能外设TIMER                   |
| timer_disable                              | 禁能外设TIMER                   |
| timer_auto_reload_shadow_enable            | TIMER自动重载影子使能               |
| timer_auto_reload_shadow_disable           | TIMER自动重载影子禁能               |
| timer_update_event_enable                  | TIMER更新使能                   |
| timer_update_event_disable                 | TIMER更新禁能                   |
| timer_counter_alignment                    | 设置外设TIMER的对齐模式              |
| timer_counter_up_direction                 | 设置外设TIMER向上计数               |
| timer_counter_down_direction               | 设置外设TIMER向下计数               |
| timer_prescaler_config                     | 配置外设TIMER预分频器               |
| timer_repetition_value_config              | 配置外设TIMER的重复计数器             |
| timer_autoreload_value_config              | 配置外设TIMER的自动重载寄存器           |
| timer_counter_value_config                 | 配置外设TIMER的计数器值              |
| timer_counter_read                         | 读取外设TIMER的计数器值              |
| timer_prescaler_read                       | 读取外设TIMER的预分频器值             |
| timer_single_pulse_mode_config             | 配置外设TIMER的单脉冲模式             |
| timer_update_source_config                 | 配置外设TIMER的更新源               |
| timer_channel_control_shadow_config        | 配置通道控制影子寄存器                 |
| timer_channel_control_shadow_update_config | 配置TIMER通道控制影子寄存器更新控制        |
| timer_dma_enable                           | 外设TIMER的DMA使能               |
| timer_dma_disable                          | 外设TIMER的DMA禁能               |
| timer_channel_dma_request_source_select    | 外设TIMER的通道DMA请求源选择          |
| timer_dma_transfer_config                  | 配置外设TIMER的DMA模式             |
| timer_event_software_generate              | 软件产生事件                      |
| timer_break_struct_para_init               | 将TIMER中止功能参数结构体中所有参数初始化为默认值 |
| timer_break_config                         | 配置中止功能                      |
| timer_break_enable                         | 使能TIMER的中止功能                |
| timer_break_disable                        | 禁能TIMER的中止功能                |
| timer_automatic_output_enable              | 自动输出使能                      |
| timer_automatic_output_disable             | 自动输出禁能                      |
| timer_primary_output_config                | 所有的通道输出使能                   |
| timer_channel_output_struct_para_init      | 将TIMER通道输出参数结构体中所有参数初始化为默认值 |
| timer_channel_output_config                | 外设TIMER的通道输出配置              |



| 库函数名称   | 库函数描述                          |
|---|--------------------------------|
| timer_channel_output_mode_config                      | 配置外设TIMER通道输出比较模式              |
| timer_channel_output_pulse_value_config               | 配置外设TIMER的通道输出比较值              |
| timer_channel_output_shadow_config                    | 配置TIMER通道输出比较影子寄存器功能           |
| timer_channel_output_clear_config                     | 配置TIMER的通道输出比较清0功能             |
| timer_channel_output_polarity_config                  | 通道输出极性配置                       |
| timer_channel_complementary_output_polarity_config    | 互补通道输出极性配置                     |
| timer_channel_output_state_config                     | 配置通道状态                         |
| timer_channel_complementary_output_state_config       | 配置互补通道输出状态                     |
| timer_channel_input_struct_para_init                  | 将TIMER通道输入参数结构体中所有参数初始化为默认值    |
| timer_input_capture_config                            | 配置TIMER输入捕获参数                  |
| timer_channel_input_capture_prescaler_config          | 配置TIMER通道输入捕获预分频值              |
| timer_channel_capture_value_register_read             | 读取通道输入捕获值                      |
| timer_input_pwm_capture_config                        | 配置TIMER捕获PWM输入参数               |
| timer_hall_mode_config                                | 配置TIMER的HALL接口功能               |
| timer_multi_mode_channel_output_parameter_struct_init | 将TIMER多模式通道输出参数结构体中所有参数初始化为默认值 |
| timer_multi_mode_channel_output_config                | 外设TIMER的多模式通道输出配置              |
| timer_multi_mode_channel_mode_config                  | 外设TIMER多模式通道模式选择               |
| timer_input_trigger_source_select                     | TIMER的输入触发源选择                  |
| timer_master_output_trigger_source_select             | 选择TIMER主模式输出触发                 |
| timer_slave_mode_select                               | TIMER从模式配置                     |
| timer_master_slave_mode_config                        | TIMER主从模式配置                    |
| timer_external_trigger_config                         | 配置TIMER外部触发输入                  |
| timer_quadrature_decoder_mode_config                  | TIMER配置为编码器模式                  |
| timer_internal_clock_config                           | TIMER配置为内部时钟模式                 |
| timer_internal_trigger_as_external_clock_config       | 配置TIMER的内部触发为时钟源               |
| timer_external_trigger_as_external_clock_config       | 配置TIMER的外部触发作为时钟源              |

| 库函数名称  | 库函数描述                         |
|--|-------------------------------|
| timer_external_clock_mode0_config                          | 配置TIMER外部时钟模式0，ETI作为时钟源       |
| timer_external_clock_mode1_config                          | 配置TIMER外部时钟模式1                |
| timer_external_clock_mode1_disable                         | TIMER外部时钟模式1禁能                |
| timer_channel_remap_config                                 | 配置TIMER通道重映射功能                |
| timer_write_chxval_register_config                         | 配置TIMER写CHxVAL选择位             |
| timer_output_value_selection_config                        | 配置TIMER输出值选择位                 |
| timer_output_match_pulse_select                            | 配置TIMER输出匹配脉冲选择               |
| timer_channel_composite_pwm_mode_config                    | 配置TIMER的复合PWM模式               |
| timer_channel_composite_pwm_mode_output_pulse_value_config | 配置TIMER的复合PWM模式输出脉冲值          |
| timer_channel_additional_compare_value_config              | 配置TIMER通道附加比较寄存器值             |
| timer_channel_additional_output_shadow_config              | 配置TIMER通道附加输出比较影子寄存器功能        |
| timer_break_external_input_struct_para_init                | 初始化TIMER中止功能外部输入参数结构体         |
| timer_break_external_input_config                          | 配置TIMER中止功能外部输入               |
| timer_break_external_input_enable                          | 中止功能外部输入使能                    |
| timer_break_external_input_disable                         | 中止功能外部输入禁能                    |
| timer_break_external_input_polarity_config                 | 配置TIMER中止功能外部输入极性             |
| timer_channel_break_control_config                         | 配置TIMER通道的中止功能                |
| timer_channel_dead_time_config                             | 配置TIMER通道的死区功能                |
| timer_free_complementary_struct_para_init                  | 将TIMER通道独立互补参数结构体中所有参数初始化为默认值 |
| timer_channel_free_complementary_config                    | 配置TIMER独立互补通道保护功能             |
| timer_flag_get   | 获取外设TIMER的状态标志                |
| timer_flag_clear   | 清除外设TIMER状态标志                 |
| timer_interrupt_enable                                     | 外设TIMER中断使能                   |
| timer_interrupt_disable                                    | 外设TIMER中断禁能                   |
| timer_interrupt_flag_get                                   | 获取外设TIMER中断标志                 |
| timer_interrupt_flag_clear                                 | 清除外设TIMER的中断标志                |

## 结构体 timer\_parameter\_struct

表 3-653. 结构体 timer\_parameter\_struct

| 成员名称      | 功能描述          |
|-----------|---------------|
| prescaler | 预分频值（0~65535） |

| 成员名称              | 功能描述   |
|-------------------|--|
| alignedmode       | 对齐模式 (TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH) |
| counterdirection  | 计数方向 (TIMER_COUNTER_UP, TIMER_COUNTER_DOWN)  |
| period            | 周期   |
| clockdivision     | 时钟分频因子 (TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4)  |
| repetitioncounter | 重复计数器值 (0~255)   |

### 结构体 timer\_break\_parameter\_struct

表 3-654. 结构体 timer\_break\_parameter\_struct

| 成员名称            | 功能描述   |
|-----------------|--|
| runoffstate     | 运行模式下“关闭状态”配置 (TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE)                          |
| ideloffstate    | 空闲模式下“关闭状态”配置 (TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE)                          |
| deadtime        | 死区时间 (0~255)   |
| breakpolarity   | 中止信号极性 (TIMER_BREAK_POLARITY_LOW, TIMER_BREAK_POLARITY_HIGH)                             |
| outputautostate | 自动输出使能 (TIMER_OUTAUTO_ENABLE, TIMER_OUTAUTO_DISABLE)                                     |
| protectmode     | 互补寄存器保护控制 (TIMER_CCHP_PROT_OFF, TIMER_CCHP_PROT_0, TIMER_CCHP_PROT_1, TIMER_CCHP_PROT_2) |
| breakstate      | 中止使能 (TIMER_BREAK_ENABLE, TIMER_BREAK_DISABLE)   |

### 结构体 timer\_oc\_parameter\_struct

表 3-655. 结构体 timer\_oc\_parameter\_struct

| 成员名称         | 功能描述  |
|--------------|---|
| outputstate  | 通道输出状态 (TIMER_CCX_ENABLE, TIMER_CCX_DISABLE)                      |
| outputnstate | 互补通道输出状态 (TIMER_CCXN_ENABLE, TIMER_CCXN_DISABLE)                  |
| ocpolarity   | 通道输出极性 (TIMER_OC_POLARITY_HIGH, TIMER_OC_POLARITY_LOW)            |
| ocnpolarity  | 互补通道输出极性 (TIMER_OCN_POLARITY_HIGH, TIMER_OCN_POLARITY_LOW)        |
| ocidlestate  | 空闲状态下通道输出 (TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH)     |
| ocnidlestate | 空闲状态下互补通道输出 (TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH) |

### 结构体 timer\_omc\_parameter\_struct

表 3-656. 结构体 timer\_omc\_parameter\_struct

| 成员名称        | 功能描述  |
|-------------|---|
| outputmode  | 多模式通道输出模式选择 (TIMER_MCH_MODE_INDEPENDENTLY, TIMER_MCH_MODE_MIRRORED, TIMER_MCH_MODE_COMPLEMENTARY) |
| outputstate | 多模式通道输出状态 (TIMER_MCCX_ENABLE, TIMER_MCCX_DISABLE)   |
| ocpolarity  | 多模式通道输出极性 (TIMER_OMC_POLARITY_HIGH, TIMER_OMC_POLARITY_LOW)                                       |

### 结构体 timer\_ic\_parameter\_struct

表 3-657. 结构体 timer\_ic\_parameter\_struct

| 成员名称        | 功能描述   |
|-------------|--|
| icpolarity  | 通道输入极性 (TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_FALLING, TIMER_IC_POLARITY_BOTH_EDGE)                              |
| icselection | 通道输入模式选择 (TIMER_IC_SELECTION_DIRECTTI, TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS, TIMER_IC_SELECTION_PAIR) |
| icprescaler | 通道输入捕获预分频 (TIMER_IC_PSC_DIV1, TIMER_IC_PSC_DIV2, TIMER_IC_PSC_DIV4, TIMER_IC_PSC_DIV8)                                 |
| icfilter    | 通道输入捕获滤波 (0~15)  |

### 结构体 timer\_break\_ext\_input\_struct

表 3-658. 结构体 timer\_break\_ext\_input\_struct

| 成员名称     | 功能描述   |
|----------|--|
| filter   | 中止功能外部输入滤波 (0~15)  |
| enable   | 中止功能外部输入使能 (ENABLE, DISABLE)                                     |
| polarity | 中止功能外部输入极性 (TIMER_BRKIN_POLARITY_HIGH, TIMER_BRKIN_POLARITY_LOW) |

### 结构体 timer\_free\_complementary\_parameter\_struct

表 3-659. 结构体 timer\_free\_complementary\_parameter\_struct

| 成员名称         | 功能描述   |
|--------------|--|
| freecomstate | 独立互补通道保护使能 (TIMER_FCCHP_STATE_ENABLE, TIMER_FCCHP_STATE_DISABLE) |
| runoffstate  | 运行模式下“关闭状态”配置 (TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE)  |
| ideloffstate | 空闲模式下“关闭状态”配置 (TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE)  |
| deadtime     | 死区时间 (0~255)   |

## 函数 timer\_deinit

函数timer\_deinit描述见下表:

表 3-660. 函数 timer\_deinit

|                           |  |
|---------------------------|--|
| 函数名称                      | timer_deinit                                       |
| 函数原型                      | void timer_deinit(uint32_t timer_periph);          |
| 功能描述                      | 复位外设TIMER  |
| 先决条件                      | -  |
| 被调用函数                     | rcu_periph_reset_enable / rcu_periph_reset_disable |
| 输入参数{in}                  |  |
| timer_periph              | TIMER外设  |
| TIMERx(x=0,1,5,6,7,19,20) | TIMER外设选择  |
| 输出参数{out}                 |  |
| -                         | -  |
| 返回值                       |  |
| -                         | -  |

例如:

```
/* reset TIMER0 */
```

```
timer_deinit (TIMER0);
```

## 函数 timer\_struct\_para\_init

函数timer\_struct\_para\_init描述见下表:

表 3-661. 函数 timer\_struct\_para\_init

|           |  |
|-----------|--|
| 函数名称      | timer_struct_para_init   |
| 函数原型      | void timer_struct_para_init(timer_parameter_struct* initpara);   |
| 功能描述      | 将TIMER初始化参数结构体中所有参数初始化为默认值                                       |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| initpara  | TIMER初始化结构体, 结构体成员参考 <a href="#">结构体timer_parameter_struct</a> 。 |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如:

```
/* initialize TIMER init parameter struct with a default value */
```

```
timer_parameter_struct timer_initpara;
```

```
timer_struct_para_init(&timer_initpara);
```

## 函数 timer\_init

函数timer\_init描述见下表:

**表 3-662. 函数 timer\_init**

|                          |   |
|--------------------------|---|
| 函数名称                     | timer_init  |
| 函数原型                     | void timer_init(uint32_t timer_periph, timer_parameter_struct* initpara); |
| 功能描述                     | 初始化外设TIMER  |
| 先决条件                     | -   |
| 被调用函数                    | -   |
| 输入参数{in}                 |   |
| timer_periph             | TIMER外设   |
| TIMEx(x=0,1,5,6,7,19,20) | TIMER外设选择   |
| 输入参数{in}                 |   |
| initpara                 | TIMER初始化结构体, 结构体成员参考 <a href="#">结构体timer_parameter_struct</a> 。          |
| 输出参数{out}                |   |
| -                        | -   |
| 返回值                      |   |
| -                        | -   |

例如:

```
/* 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);
```

## 函数 timer\_enable

函数timer\_enable描述见下表:

**表 3-663. 函数 timer\_enable**

|      |   |
|------|---|
| 函数名称 | timer_enable                              |
| 函数原型 | void timer_enable(uint32_t timer_periph); |

|                          |           |
|--------------------------|-----------|
| 功能描述                     | 使能外设TIMER |
| 先决条件                     | -         |
| 被调用函数                    | -         |
| 输入参数{in}                 |           |
| timer_periph             | TIMER外设   |
| TIMEx(x=0,1,5,6,7,19,20) | TIMER外设选择 |
| 输出参数{out}                |           |
| -                        | -         |
| 返回值                      |           |
| -                        | -         |

例如:

```
/* enable TIMER0 */
timer_enable(TIMER0);
```

### 函数 timer\_disable

函数timer\_disable描述见下表:

表 3-664. 函数 timer\_disable

|                          |  |
|--------------------------|--|
| 函数名称                     | timer_disable                              |
| 函数原型                     | void timer_disable(uint32_t timer_periph); |
| 功能描述                     | 禁能外设TIMER                                  |
| 先决条件                     | -  |
| 被调用函数                    | -  |
| 输入参数{in}                 |  |
| timer_periph             | TIMER外设                                    |
| TIMEx(x=0,1,5,6,7,19,20) | TIMER外设选择                                  |
| 输出参数{out}                |  |
| -                        | -  |
| 返回值                      |  |
| -                        | -  |

例如:

```
/* disable TIMER0 */
timer_disable(TIMER0);
```

### 函数 timer\_auto\_reload\_shadow\_enable

函数timer\_auto\_reload\_shadow\_enable描述见下表:

表 3-665. 函数 timer\_auto\_reload\_shadow\_enable

|                           |  |
|---------------------------|--|
| 函数名称                      | timer_auto_reload_shadow_enable                              |
| 函数原型                      | void timer_auto_reload_shadow_enable(uint32_t timer_periph); |
| 功能描述                      | TIMER自动重载影子使能  |
| 先决条件                      | -  |
| 被调用函数                     | -  |
| 输入参数{in}                  |  |
| timer_periph              | TIMER外设  |
| TIMERx(x=0,1,5,6,7,19,20) | TIMER外设选择  |
| 输出参数{out}                 |  |
| -                         | -  |
| 返回值                       |  |
| -                         | -  |

例如:

```
/* enable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_enable(TIMER0);
```

### 函数 timer\_auto\_reload\_shadow\_disable

函数timer\_auto\_reload\_shadow\_disable描述见下表:

表 3-666. 函数 timer\_auto\_reload\_shadow\_disable

|                           |  |
|---------------------------|--|
| 函数名称                      | timer_auto_reload_shadow_disable                               |
| 函数原型                      | void timer_auto_reload_shadow_disable (uint32_t timer_periph); |
| 功能描述                      | TIMER自动重载影子禁能  |
| 先决条件                      | -  |
| 被调用函数                     | -  |
| 输入参数{in}                  |  |
| timer_periph              | TIMER外设  |
| TIMERx(x=0,1,5,6,7,19,20) | TIMER外设选择  |
| 输出参数{out}                 |  |
| -                         | -  |
| 返回值                       |  |
| -                         | -  |

例如:

```
/* disable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_disable(TIMER0);
```



**函数 timer\_update\_event\_enable**

函数timer\_update\_event\_enable描述见下表：

**表 3-667. 函数 timer\_update\_event\_enable**

|                           |  |
|---------------------------|--|
| 函数名称                      | timer_update_event_enable                              |
| 函数原型                      | void timer_update_event_enable(uint32_t timer_periph); |
| 功能描述                      | TIMER更新使能  |
| 先决条件                      | -  |
| 被调用函数                     | -  |
| 输入参数{in}                  |  |
| timer_periph              | TIMER外设  |
| TIMERx(x=0,1,5,6,7,19,20) | TIMER外设选择  |
| 输出参数{out}                 |  |
| -                         | -  |
| 返回值                       |  |
| -                         | -  |

例如：

```
/* enable TIMER0 the update event */
```

```
timer_update_event_enable(TIMER0);
```

**函数 timer\_update\_event\_disable**

函数timer\_update\_event\_disable描述见下表：

**表 3-668. 函数 timer\_update\_event\_disable**

|                           |  |
|---------------------------|--|
| 函数名称                      | timer_update_event_disable                               |
| 函数原型                      | void timer_update_event_disable (uint32_t timer_periph); |
| 功能描述                      | TIMER更新禁能  |
| 先决条件                      | -  |
| 被调用函数                     | -  |
| 输入参数{in}                  |  |
| timer_periph              | TIMER外设  |
| TIMERx(x=0,1,5,6,7,19,20) | TIMER外设选择  |
| 输出参数{out}                 |  |
| -                         | -  |
| 返回值                       |  |
| -                         | -  |

例如：

```
/* disable TIMER0 the update event */
```

```
timer_update_event_disable(TIMERO);
```

### 函数 timer\_counter\_alignment

函数timer\_counter\_alignment描述见下表：

**表 3-669. 函数 timer\_counter\_alignment**

|                           |  |
|---------------------------|--|
| 函数名称                      | timer_counter_alignment  |
| 函数原型                      | void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned); |
| 功能描述                      | 设置外设TIMER的对齐模式   |
| 先决条件                      | -  |
| 被调用函数                     | -  |
| 输入参数{in}                  |  |
| timer_periph              | TIMER外设  |
| TIMERx(x=0,1,7,19,20)     | TIMER外设选择  |
| 输入参数{in}                  |  |
| aligned                   | 对齐模式   |
| TIMER_COUNTER_EDGE        | 边沿对齐模式   |
| TIMER_COUNTER_CENTER_DOWN | 中央对齐向下计数模式   |
| TIMER_COUNTER_CENTER_UP   | 中央对齐向上计数模式   |
| TIMER_COUNTER_CENTER_BOTH | 中央对齐上下计数模式   |
| 输出参数{out}                 |  |
| -                         | -  |
| 返回值                       |  |
| -                         | -  |

例如：

```
/* set TIMERO counter center-aligned and counting up assert mode */
```

```
timer_counter_alignment(TIMERO, TIMER_COUNTER_CENTER_UP);
```

### 函数 timer\_counter\_up\_direction

函数timer\_counter\_up\_direction描述见下表：

**表 3-670. 函数 timer\_counter\_up\_direction**

|      |   |
|------|---|
| 函数名称 | timer_counter_up_direction                              |
| 函数原型 | void timer_counter_up_direction(uint32_t timer_periph); |
| 功能描述 | 设置外设TIMER向上计数   |
| 先决条件 | 计数器设置为无中央对齐计数模式（边沿对齐模式）                                 |

|                       |           |
|-----------------------|-----------|
| 被调用函数                 | -         |
| 输入参数{in}              |           |
| timer_periph          | TIMER外设   |
| TIMERx(x=0,1,7,19,20) | TIMER外设选择 |
| 输出参数{out}             |           |
| -                     | -         |
| 返回值                   |           |
| -                     | -         |

例如：

```
/* set TIMERO counter up direction */
```

```
timer_counter_up_direction(TIMERO);
```

### 函数 timer\_counter\_down\_direction

函数timer\_counter\_down\_direction描述见下表：

表 3-671. 函数 timer\_counter\_down\_direction

|                           |   |
|---------------------------|---|
| 函数名称                      | timer_counter_down_direction                              |
| 函数原型                      | void timer_counter_down_direction(uint32_t timer_periph); |
| 功能描述                      | 设置外设TIMER向下计数   |
| 先决条件                      | 计数器设置为无中央对齐计数模式（边沿对齐模式）                                   |
| 被调用函数                     | -   |
| 输入参数{in}                  |   |
| timer_periph              | TIMER外设   |
| TIMERx<br>(x=0,1,7,19,20) | TIMER外设选择   |
| 输出参数{out}                 |   |
| -                         | -   |
| 返回值                       |   |
| -                         | -   |

例如：

```
/* set TIMERO counter down direction */
```

```
timer_counter_down_direction(TIMERO);
```

### 函数 timer\_prescaler\_config

函数timer\_prescaler\_config描述见下表：

表 3-672. 函数 timer\_prescaler\_config

|      |                        |
|------|------------------------|
| 函数名称 | timer_prescaler_config |
|------|------------------------|

|                           |   |
|---------------------------|---|
| 函数原型                      | void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint32_t pscreload); |
| 功能描述                      | 配置外设TIMER预分频器   |
| 先决条件                      | -   |
| 被调用函数                     | -   |
| 输入参数{in}                  |   |
| timer_periph              | TIMER外设   |
| TIMERx(x=0,1,5,6,7,19,20) | TIMER外设选择   |
| 输入参数{in}                  |   |
| prescaler                 | 预分频值, 0~65535   |
| 输入参数{in}                  |   |
| pscreload                 | 预分频值加载模式  |
| TIMER_PSC_RELOAD_NOW      | 预分频值立即加载  |
| TIMER_PSC_RELOAD_UPDATE   | 预分频值在下次更新事件发生时加载  |
| 输出参数{out}                 |   |
| -                         | -   |
| 返回值                       |   |
| -                         | -   |

例如:

```
/* configure TIMER0 prescaler */
```

```
timer_prescaler_config(TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

### 函数 timer\_repetition\_value\_config

函数timer\_repetition\_value\_config描述见下表:

表 3-673. 函数 timer\_repetition\_value\_config

|                     |   |
|---------------------|---|
| 函数名称                | timer_repetition_value_config   |
| 函数原型                | void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition); |
| 功能描述                | 配置外设TIMER的重复计数器   |
| 先决条件                | -   |
| 被调用函数               | -   |
| 输入参数{in}            |   |
| timer_periph        | TIMER外设   |
| TIMERx(x=0,7,19,20) | TIMER外设选择   |
| 输入参数{in}            |   |
| repetition          | 重复计数器值, 取值范围 (0~255)  |
| 输出参数{out}           |   |

|     |   |
|-----|---|
| -   | - |
| 返回值 |   |
| -   | - |

例如：

```
/* configure TIMER0 repetition register value */
```

```
timer_repetition_value_config(TIMER0, 98);
```

### 函数 timer\_autoreload\_value\_config

函数timer\_autoreload\_value\_config描述见下表：

表 3-674. 函数 timer\_autoreload\_value\_config

|                           |   |
|---------------------------|---|
| 函数名称                      | timer_autoreload_value_config   |
| 函数原型                      | void timer_autoreload_value_config(uint32_t timer_periph, uint16_t autoreload); |
| 功能描述                      | 配置外设TIMER的自动重载寄存器   |
| 先决条件                      | -   |
| 被调用函数                     | -   |
| 输入参数{in}                  |   |
| timer_periph              | TIMER外设   |
| TIMERx(x=0,1,5,6,7,19,20) | TIMER外设选择   |
| 输入参数{in}                  |   |
| autoreload                | 计数器自动重载值（0~65535）   |
| 输出参数{out}                 |   |
| -                         | -   |
| 返回值                       |   |
| -                         | -   |

例如：

```
/* configure TIMER autoreload register value */
```

```
timer_autoreload_value_config(TIMER0, 3000);
```

### 函数 timer\_counter\_value\_config

函数timer\_counter\_value\_config描述见下表：

表 3-675. 函数 timer\_counter\_value\_config

|       |   |
|-------|---|
| 函数名称  | timer_counter_value_config  |
| 函数原型  | void timer_counter_value_config(uint32_t timer_periph, uint16_t counter); |
| 功能描述  | 配置外设TIMER的计数器值  |
| 先决条件  | -   |
| 被调用函数 | -   |

| 输入参数{in}                                   |                |
|--|----------------|
| <b>timer_periph</b>                        | TIMER外设        |
| <i>TIMERx</i> ( <i>x</i> =0,1,5,6,7,19,20) | TIMER外设选择      |
| 输入参数{in}                                   |                |
| <b>counter</b>                             | 计数器值 (0~65535) |
| 输出参数{out}                                  |                |
| -  | -              |
| 返回值  |                |
| -  | -              |

例如:

```
/* configure TIMER0 counter register value */
timer_counter_value_config(TIMER0, 999);
```

### 函数 timer\_counter\_read

函数timer\_counter\_read描述见下表:

表 3-676. 函数 timer\_counter\_read

| <b>函数名称</b>                                | timer_counter_read                                  |
|--|---|
| <b>函数原型</b>                                | uint32_t timer_counter_read(uint32_t timer_periph); |
| <b>功能描述</b>                                | 读取外设TIMER的计数器值                                      |
| <b>先决条件</b>                                | -   |
| <b>被调用函数</b>                               | -   |
| 输入参数{in}                                   |   |
| <b>timer_periph</b>                        | TIMER外设   |
| <i>TIMERx</i> ( <i>x</i> =0,1,5,6,7,19,20) | TIMER外设选择   |
| 输出参数{out}                                  |   |
| -  | -   |
| 返回值  |   |
| <b>uint32_t</b>                            | 外设TIMER的计数器值 (0~65535)                              |

例如:

```
/* read TIMER0 counter value */
uint32_t i = 0;
i = timer_counter_read(TIMER0);
```

### 函数 timer\_prescaler\_read

函数timer\_prescaler\_read描述见下表:

表 3-677. 函数 timer\_prescaler\_read

|                           |   |
|---------------------------|---|
| 函数名称                      | timer_prescaler_read                                  |
| 函数原型                      | uint16_t timer_prescaler_read(uint32_t timer_periph); |
| 功能描述                      | 读取外设TIMER的预分频器值                                       |
| 先决条件                      | -   |
| 被调用函数                     | -   |
| 输入参数{in}                  |   |
| timer_periph              | TIMER外设   |
| TIMERx(x=0,1,5,6,7,19,20) | TIMER外设选择   |
| 输出参数{out}                 |   |
| -                         | -   |
| 返回值                       |   |
| uint16_t                  | 外设TIMER的预分频器值（0~65535）                                |

例如：

```
/* read TIMER0 prescaler value */
```

```
uint16_t i = 0;
```

```
i = timer_prescaler_read(TIMER0);
```

### 函数 timer\_single\_pulse\_mode\_config

函数timer\_single\_pulse\_mode\_config描述见下表：

表 3-678. 函数 timer\_single\_pulse\_mode\_config

|                           |  |
|---------------------------|--|
| 函数名称                      | timer_single_pulse_mode_config   |
| 函数原型                      | void timer_single_pulse_mode_config(uint32_t timer_periph, uint32_t spmode); |
| 功能描述                      | 配置外设TIMER的单脉冲模式  |
| 先决条件                      | -  |
| 被调用函数                     | -  |
| 输入参数{in}                  |  |
| timer_periph              | TIMER外设  |
| TIMERx(x=0,1,5,6,7,19,20) | TIMER外设选择  |
| 输入参数{in}                  |  |
| spmode                    | 脉冲模式   |
| TIMER_SP_MODE_SINGLE      | 单脉冲模式计数  |
| TIMER_SP_MODE_REPETITIVE  | 重复模式计数   |
| 输出参数{out}                 |  |
| -                         | -  |

| 返回值 |   |
|-----|---|
| -   | - |

例如：

```
/* configure TIMER0 single pulse mode */
```

```
timer_single_pulse_mode_config(TIMER0, TIMER_SP_MODE_SINGLE);
```

### 函数 timer\_update\_source\_config

函数timer\_update\_source\_config描述见下表：

表 3-679. 函数 timer\_update\_source\_config

| 函数名称                     | timer_update_source_config   |
|--------------------------|--|
| 函数原型                     | void timer_update_source_config(uint32_t timer_periph, uint32_t update); |
| 功能描述                     | 配置外设TIMER的更新源  |
| 先决条件                     | -  |
| 被调用函数                    | -  |
| 输入参数{in}                 |  |
| timer_periph             | TIMER外设  |
| TIMEx(x=0,1,5,6,7,19,20) | TIMER外设选择  |
| 输入参数{in}                 |  |
| update                   | 更新源  |
| TIMER_UPDATE_SRC_GLOBAL  | 下述任一事件产生更新中断或DMA请求：<br>- UPG位被置1<br>- 计数器溢出/下溢<br>- 从模式控制器产生的更新          |
| TIMER_UPDATE_SRC_REGULAR | 只有计数器溢出/ 下溢才产生更新中断或DMA请求   |
| 输出参数{out}                |  |
| -                        | -  |
| 返回值                      |  |
| -                        | -  |

例如：

```
/* configure TIMER update only by counter overflow/underflow */
```

```
timer_update_source_config(TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

### 函数 timer\_channel\_control\_shadow\_config

函数timer\_channel\_control\_shadow\_config描述见下表：

表 3-680. 函数 timer\_channel\_control\_shadow\_config

| 函数名称 | timer_channel_control_shadow_config |
|------|-------------------------------------|
|------|-------------------------------------|



|                     |   |
|---------------------|---|
| 函数原型                | void timer_channel_control_shadow_config(uint32_t timer_periph,<br>ControlStatus newvalue); |
| 功能描述                | 配置通道控制影子寄存器   |
| 先决条件                | -   |
| 被调用函数               | -   |
| 输入参数{in}            |   |
| timer_periph        | TIMER外设   |
| TIMERx(x=0,7,19,20) | TIMER外设选择   |
| 输入参数{in}            |   |
| newvalue            | 控制状态  |
| ENABLE              | 使能  |
| DISABLE             | 禁能  |
| 输出参数{out}           |   |
| -                   | -   |
| 返回值                 |   |
| -                   | -   |

例如:

```
/* channel capture/compare control shadow register enable */
```

```
timer_channel_control_shadow_config(TIMER0, ENABLE);
```

### 函数 timer\_channel\_control\_shadow\_update\_config

函数timer\_channel\_control\_shadow\_update\_config描述见下表:

表 3-681. 函数 timer\_channel\_control\_shadow\_update\_config

|                            |   |
|----------------------------|---|
| 函数名称                       | timer_channel_control_shadow_update_config  |
| 函数原型                       | void timer_channel_control_shadow_update_config(uint32_t timer_periph,<br>uint32_t ccuctl); |
| 功能描述                       | 配置TIMER通道控制影子寄存器更新控制  |
| 先决条件                       | -   |
| 被调用函数                      | -   |
| 输入参数{in}                   |   |
| timer_periph               | TIMER外设   |
| TIMERx(x=0,7,19,20)        | TIMER外设选择   |
| 输入参数{in}                   |   |
| ccuctl                     | 通道换相控制影子寄存器更新控制   |
| TIMER_UPDATECT<br>L_CCUC   | CMTG位被置1时更新影子寄存器  |
| TIMER_UPDATECT<br>L_CCUTRI | 当CMTG位被置1或检测到TRIGI上升沿时, 影子寄存器更新   |

| 输出参数{out} |   |
|-----------|---|
| -         | - |
| 返回值       |   |
| -         | - |

例如:

```
/* configure TIMER0 channel control shadow register update when CMTG bit is set */
timer_channel_control_shadow_update_config(TIMER0, TIMER_UPDATECTL_CCU);
```

### 函数 timer\_dma\_enable

函数timer\_dma\_enable描述见下表:

表 3-682. 函数 timer\_dma\_enable

| 函数名称                      | timer_dma_enable  |
|---------------------------|---|
| 函数原型                      | void timer_dma_enable(uint32_t timer_periph, uint32_t dma); |
| 功能描述                      | 外设TIMER的DMA使能   |
| 先决条件                      | -   |
| 被调用函数                     | -   |
| 输入参数{in}                  |   |
| <b>timer_periph</b>       | TIMER外设   |
| TIMERx(x=0,1,5,6,7,19,20) | 参考具体参数  |
| 输入参数{in}                  |   |
| <b>dma</b>                | DMA源  |
| TIMER_DMA_UPD             | 更新DMA请求, TIMERx(x=0,1,5,6,7,19,20)                          |
| TIMER_DMA_CH0D            | 通道0比较/捕获DMA请求, TIMERx(x=0,1,7,19,20)                        |
| TIMER_DMA_CH1D            | 通道1比较/捕获DMA请求, TIMERx(x=0,1,7,19,20)                        |
| TIMER_DMA_CH2D            | 通道2比较/捕获DMA请求, TIMERx(x=0,1,7,19,20)                        |
| TIMER_DMA_CH3D            | 通道3比较/捕获DMA请求, TIMERx(x=0,1,7,19,20)                        |
| TIMER_DMA_CMTD            | 换相DMA更新请求, TIMERx(x=0,7,19,20)                              |
| TIMER_DMA_TRGD            | 触发DMA请求, TIMERx(x=0,1,7,19,20)                              |
| TIMER_DMA_MCH0D           | 多模式通道0比较/捕获DMA请求, TIMERx(x=0,1,7,19,20)                     |
| TIMER_DMA_MCH1D           | 多模式通道0通道1比较/捕获DMA请求, TIMERx(x=0,1,7,19,20)                  |
| TIMER_DMA_MCH             | 多模式通道0通道2比较/捕获DMA请求, TIMERx(x=0,1,7,19,20)                  |

|                     |  |
|---------------------|--|
| 2D                  |  |
| TIMER_DMA_MCH<br>3D | 多模式通道0通道3比较/捕获DMA请求, TIMERx(x=0,1,7,19,20) |
| 输出参数{out}           |  |
| -                   | -  |
| 返回值                 |  |
| -                   | -  |

例如:

```
/* enable the TIMER0 update DMA */
```

```
timer_dma_enable(TIMER0, TIMER_DMA_UPD);
```

### 函数 timer\_dma\_disable

函数timer\_dma\_disable描述见下表:

表 3-683. 函数 timer\_dma\_disable

|                           |   |
|---------------------------|---|
| 函数名称                      | timer_dma_disable   |
| 函数原型                      | void timer_dma_disable (uint32_t timer_periph, uint32_t dma); |
| 功能描述                      | 外设TIMER的DMA禁能   |
| 先决条件                      | -   |
| 被调用函数                     | -   |
| 输入参数{in}                  |   |
| timer_periph              | TIMER外设   |
| TIMERx(x=0,1,5,6,7,19,20) | 参考具体参数  |
| 输入参数{in}                  |   |
| dma                       | DMA源  |
| TIMER_DMA_UPD             | 更新DMA请求, TIMERx(x=0,1,5,6,7,19,20)                            |
| TIMER_DMA_CH0<br>D        | 通道0比较/捕获DMA请求, TIMERx(x=0,1,7,19,20)                          |
| TIMER_DMA_CH1<br>D        | 通道1比较/捕获DMA请求, TIMERx(x=0,1,7,19,20)                          |
| TIMER_DMA_CH2<br>D        | 通道2比较/捕获DMA请求, TIMERx(x=0,1,7,19,20)                          |
| TIMER_DMA_CH3<br>D        | 通道3比较/捕获DMA请求, TIMERx(x=0,1,7,19,20)                          |
| TIMER_DMA_CMT<br>D        | 换相DMA更新请求, TIMERx(x=0,7,19,20)                                |
| TIMER_DMA_TRG<br>D        | 触发DMA请求, TIMERx(x=0,1,7,19,20)                                |
| TIMER_DMA_MCH<br>0D       | 多模式通道0比较/捕获DMA请求, TIMERx(x=0,7,19,20)                         |

|                            |                                       |
|----------------------------|---------------------------------------|
| <i>TIMER_DMA_MCH</i><br>1D | 多模式通道1比较/捕获DMA请求, TIMERx(x=0,7,19,20) |
| <i>TIMER_DMA_MCH</i><br>2D | 多模式通道2比较/捕获DMA请求, TIMERx(x=0,7,19,20) |
| <i>TIMER_DMA_MCH</i><br>3D | 多模式通道3比较/捕获DMA请求, TIMERx(x=0,7,19,20) |
| 输出参数{out}                  |                                       |
| -                          | -                                     |
| 返回值                        |                                       |
| -                          | -                                     |

例如:

```
/* disable the TIMER0 update DMA */
```

```
timer_dma_disable(TIMER0, TIMER_DMA_UPD);
```

### 函数 timer\_channel\_dma\_request\_source\_select

函数timer\_channel\_dma\_request\_source\_select描述见下表:

表 3-684. 函数 timer\_channel\_dma\_request\_source\_select

|                               |  |
|-------------------------------|--|
| 函数名称                          | timer_channel_dma_request_source_select  |
| 函数原型                          | void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request); |
| 功能描述                          | 外设TIMER的通道DMA请求源选择   |
| 先决条件                          | -  |
| 被调用函数                         | -  |
| 输入参数{in}                      |  |
| timer_periph                  | TIMER外设  |
| TIMERx(x=0,1,5,6,7,19,20)     | TIMER外设选择  |
| 输入参数{in}                      |  |
| dma_request                   | 通道的DMA请求源选择  |
| TIMER_DMAREQUEST_CHANNELEVENT | 当通道捕获/比较事件发生时, 发送通道n的DMA请求   |
| TIMER_DMAREQUEST_UPDATEEVENT  | 当更新事件发生, 发送通道n的DMA请求   |
| 输出参数{out}                     |  |
| -                             | -  |
| 返回值                           |  |
| -                             | -  |

例如:

/\* TIMER0 channel DMA request of channel n is sent when channel event occurs \*/

```
timer_channel_dma_request_source_select (TIMER0,  
TIMER_DMAREQUEST_CHANNELEVENT);
```

### 函数 timer\_dma\_transfer\_config

函数timer\_dma\_transfer\_config描述见下表：

表 3-685. 函数 timer\_dma\_transfer\_config

| 函数名称                        | timer_dma_transfer_config   |
|-----------------------------|---|
| 函数原型                        | void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth); |
| 功能描述                        | 配置外设TIMER的DMA模式   |
| 先决条件                        | -   |
| 被调用函数                       | -   |
| 输入参数{in}                    |   |
| timer_periph                | TIMER外设   |
| TIMERx                      | 参考具体参数  |
| 输入参数{in}                    |   |
| dma_baseaddr                | DMA传输起始地址   |
| TIMER_DMACFG_DMATA_CTL0     | DMA传输起始地址：TIMER_CTL0, TIMERx(x=0,1,7,19,20)   |
| TIMER_DMACFG_DMATA_CTL1     | DMA传输起始地址：TIMER_CTL1, TIMERx(x=0,1,7,19,20)   |
| TIMER_DMACFG_DMATA_SMCFG    | DMA传输起始地址：TIMER_SMCFG, TIMERx(x=0,1,7,19,20)  |
| TIMER_DMACFG_DMATA_DMAINTEN | DMA传输起始地址：TIMER_DMAINTEN, TIMERx(x=0,1,7,19,20)   |
| TIMER_DMACFG_DMATA_INTF     | DMA传输起始地址：TIMER_INTF, TIMERx(x=0,1,7,19,20)   |
| TIMER_DMACFG_DMATA_SWEVG    | DMA传输起始地址：TIMER_SWEVG, TIMERx(x=0,1,7,19,20)  |
| TIMER_DMACFG_DMATA_CHCTL0   | DMA传输起始地址：TIMER_CHCTL0, TIMERx(x=0,1,7,19,20)   |
| TIMER_DMACFG_DMATA_CHCTL1   | DMA传输起始地址：TIMER_CHCTL1, TIMERx(x=0,1,7,19,20)   |
| TIMER_DMACFG_DMATA_CHCTL2   | DMA传输起始地址：TIMER_CHCTL2, TIMERx(x=0,1,7,19,20)   |
| TIMER_DMACFG_DMATA_CNT      | DMA传输起始地址：TIMER_CNT, TIMERx(x=0,1,7,19,20)  |
| TIMER_DMACFG_DMATA_PSC      | DMA传输起始地址：TIMER_PSC, TIMERx(x=0,1,7,19,20)  |

|                                       |   |
|---------------------------------------|---|
| <i>TIMER_DMACFG_DMATA_CAR</i>         | DMA传输起始地址: TIMER_CAR, TIMERx(x=0,1,7,19,20)       |
| <i>TIMER_DMACFG_DMATA_CREP</i>        | DMA传输起始地址: TIMER_CREP, TIMERx(x=0,7,19,20)        |
| <i>TIMER_DMACFG_DMATA_CH0CV</i>       | DMA传输起始地址: TIMER_CH0CV, TIMERx(x=0,1,7,19,20)     |
| <i>TIMER_DMACFG_DMATA_CH1CV</i>       | DMA传输起始地址: TIMER_CH1CV, TIMERx(x=0,1,7,19,20)     |
| <i>TIMER_DMACFG_DMATA_CH2CV</i>       | DMA传输起始地址: TIMER_CH2CV, TIMERx(x=0,1,7,19,20)     |
| <i>TIMER_DMACFG_DMATA_CH3CV</i>       | DMA传输起始地址: TIMER_CH3CV, TIMERx(x=0,1,7,19,20)     |
| <i>TIMER_DMACFG_DMATA_CCHP</i>        | DMA传输起始地址: TIMER_CCHP, TIMERx(x=0,7,19,20)        |
| <i>TIMER_DMACFG_DMATA_MCHCTL0</i>     | DMA传输起始地址: TIMER_MCHCTL0, TIMERx(x=0,7,19,20)     |
| <i>TIMER_DMACFG_DMATA_MCHCTL1</i>     | DMA传输起始地址: TIMER_MCHCTL1, TIMERx(x=0,7,19,20)     |
| <i>TIMER_DMACFG_DMATA_MCHCTL2</i>     | DMA传输起始地址: TIMER_MCHCTL2, TIMERx(x=0,7,19,20)     |
| <i>TIMER_DMACFG_DMATA_MCH0CV</i>      | DMA传输起始地址: TIMER_MCH0CV, TIMERx(x=0,7,19,20)      |
| <i>TIMER_DMACFG_DMATA_MCH1CV</i>      | DMA传输起始地址: TIMER_MCH1CV, TIMERx(x=0,7,19,20)      |
| <i>TIMER_DMACFG_DMATA_MCH2CV</i>      | DMA传输起始地址: TIMER_MCH2CV, TIMERx(x=0,7,19,20)      |
| <i>TIMER_DMACFG_DMATA_MCH3CV</i>      | DMA传输起始地址: TIMER_MCH3CV, TIMERx(x=0,7,19,20)      |
| <i>TIMER_DMACFG_DMATA_CH0COMV_ADD</i> | DMA传输起始地址: TIMER_CH0COMV_ADD, TIMERx(x=0,7,19,20) |
| <i>TIMER_DMACFG_DMATA_CH1COMV_ADD</i> | DMA传输起始地址: TIMER_CH1COMV_ADD, TIMERx(x=0,7,19,20) |
| <i>TIMER_DMACFG_DMATA_CH2COMV_ADD</i> | DMA传输起始地址: TIMER_CH2COMV_ADD, TIMERx(x=0,7,19,20) |
| <i>TIMER_DMACFG_DMATA_CH3COMV_ADD</i> | DMA传输起始地址: TIMER_CH3COMV_ADD, TIMERx(x=0,7,19,20) |
| <i>TIMER_DMACFG_DMATA_CTL2</i>        | DMA传输起始地址: TIMER_CTL2, TIMERx(x=0,7,19,20)        |
| <i>TIMER_DMACFG_</i>                  | DMA传输起始地址: TIMER_BRKCFG, TIMERx(x=0,7,19,20)      |

|                                     |   |
|-------------------------------------|---|
| <i>DMATA_BRKCFG</i>                 |   |
| <i>TIMER_DMACFG_DMATA_FCCHP0</i>    | DMA传输起始地址: <i>TIMER_FCCHP0</i> , <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_DMACFG_DMATA_FCCHP1</i>    | DMA传输起始地址: <i>TIMER_FCCHP1</i> , <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_DMACFG_DMATA_FCCHP2</i>    | DMA传输起始地址: <i>TIMER_FCCHP2</i> , <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_DMACFG_DMATA_FCCHP3</i>    | DMA传输起始地址: <i>TIMER_FCCHP3</i> , <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| 输入参数{in}                            |   |
| <b>dma_lenth</b>                    | DMA传输长度   |
| <i>TIMER_DMACFG_DMATC_xTRANSFER</i> | ( <i>x</i> =1~35), DMA传输 <i>x</i> 次                                   |
| 输出参数{out}                           |   |
| -                                   | -   |
| 返回值                                 |   |
| -                                   | -   |

例如:

```
/* configure the TIMER0 DMA transfer */
```

```
timer_dma_transfer_config(TIMER0, TIMER_DMACFG_DMATA_CTL0,  
TIMER_DMACFG_DMATC_5TRANSFER);
```

### 函数 timer\_event\_software\_generate

函数timer\_event\_software\_generate描述见下表:

表 3-686. 函数 timer\_event\_software\_generate

|                             |  |
|-----------------------------|--|
| 函数名称                        | timer_event_software_generate  |
| 函数原型                        | void timer_event_software_generate(uint32_t timer_periph, uint32_t event); |
| 功能描述                        | 软件产生事件   |
| 先决条件                        | -  |
| 被调用函数                       | -  |
| 输入参数{in}                    |  |
| <b>timer_periph</b>         | TIMER外设  |
| <i>TIMERx</i>               | 参考具体参数   |
| 输入参数{in}                    |  |
| <b>event</b>                | 事件源  |
| <i>TIMER_EVENT_SRC_UPG</i>  | 更新事件产生, <i>TIMERx</i> ( <i>x</i> =0,1,5,6,7,19,20)                         |
| <i>TIMER_EVENT_SRC_CH0G</i> | 通道0捕获或比较事件发生, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)                       |

|                                |                                      |
|--------------------------------|--------------------------------------|
| TIMER_EVENT_SR<br>C_CH1G       | 通道1捕获或比较事件发生, TIMERx(x=0,1,7,19,20)  |
| TIMER_EVENT_SR<br>C_CH2G       | 通道2捕获或比较事件发生, TIMERx(x=0,1,7,19,20)  |
| TIMER_EVENT_SR<br>C_CH3G       | 通道3捕获或比较事件发生, TIMERx(x=0,1,7,19,20)  |
| TIMER_EVENT_SR<br>C_CMTG       | 通道换相更新事件发生, TIMERx(x=0,7,19,20)      |
| TIMER_EVENT_SR<br>C_TRGG       | 触发事件产生, TIMERx(x=0,1,7,19,20)        |
| TIMER_EVENT_SR<br>C_BRKG       | 产生中止事件, TIMERx(x=0,7,19,20)          |
| TIMER_EVENT_SR<br>C_MCH0G      | 多模式通道0捕获或比较事件发生, TIMERx(x=0,7,19,20) |
| TIMER_EVENT_SR<br>C_MCH1G      | 多模式通道1捕获或比较事件发生, TIMERx(x=0,7,19,20) |
| TIMER_EVENT_SR<br>C_MCH2G      | 多模式通道2捕获或比较事件发生, TIMERx(x=0,7,19,20) |
| TIMER_EVENT_SR<br>C_MCH3G      | 多模式通道3捕获或比较事件发生, TIMERx(x=0,7,19,20) |
| TIMER_EVENT_SR<br>C_CH0COMADDG | 通道0附加比较事件发生, TIMERx(x=0,7,19,20)     |
| TIMER_EVENT_SR<br>C_CH1COMADDG | 通道1附加比较事件发生, TIMERx(x=0,7,19,20)     |
| TIMER_EVENT_SR<br>C_CH2COMADDG | 通道2附加比较事件发生, TIMERx(x=0,7,19,20)     |
| TIMER_EVENT_SR<br>C_CH3COMADDG | 通道3附加比较事件发生, TIMERx(x=0,7,19,20)     |
| 输出参数{out}                      |                                      |
| -                              | -                                    |
| 返回值                            |                                      |
| -                              | -                                    |

例如:

```
/* software generate update event*/
```

```
timer_event_software_generate(TIMER0, TIMER_EVENT_SRC_UPG);
```

### 函数 timer\_break\_struct\_para\_init

函数timer\_break\_struct\_para\_init描述见下表:

表 3-687. 函数 timer\_break\_struct\_para\_init

|      |                              |
|------|------------------------------|
| 函数名称 | timer_break_struct_para_init |
|------|------------------------------|



|           |   |
|-----------|---|
| 函数原型      | void timer_break_struct_para_init(timer_break_parameter_struct* breakpara); |
| 功能描述      | 将TIMER中止功能参数结构体中所有参数初始化为默认值   |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| breakpara | 中止功能配置结构体，详见 <a href="#">结构体timer_break_parameter_struct</a>                |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

```
/* initialize TIMER break parameter struct with a default value */
```

```
timer_break_parameter_struct timer_breakpara;
```

```
timer_break_struct_para_init(&timer_breakpara);
```

### 函数 timer\_break\_config

函数timer\_break\_config描述见下表：

表 3-688. 函数 timer\_break\_config

|                     |  |
|---------------------|--|
| 函数名称                | timer_break_config   |
| 函数原型                | void timer_break_config(uint32_t timer_periph, timer_break_parameter_struct* breakpara); |
| 功能描述                | 配置中止功能   |
| 先决条件                | -  |
| 被调用函数               | -  |
| 输入参数{in}            |  |
| timer_periph        | TIMER外设  |
| TIMERx(x=0,7,19,20) | TIMER外设选择  |
| 输入参数{in}            |  |
| breakpara           | 中止功能配置结构体，详见 <a href="#">结构体timer_break_parameter_struct</a> 。                           |
| 输出参数{out}           |  |
| -                   | -  |
| 返回值                 |  |
| -                   | -  |

例如：

```
/* configure TIMER0 break function */
```

```
timer_break_parameter_struct timer_breakpara;
```

```
timer_breakpara.runoffstate = TIMER_ROS_STATE_DISABLE;
```

```

timer_breakpara.ideloffstate    = TIMER_IOS_STATE_DISABLE ;

timer_breakpara.deadtime        = 255;

timer_breakpara.breakpolarity   = TIMER_BREAK_POLARITY_LOW;

timer_breakpara.outputautostate = TIMER_OUTAUTO_ENABLE;

timer_breakpara.protectmode     = TIMER_CCHP_PROT_0;

timer_breakpara.breakstate      = TIMER_BREAK_ENABLE;

timer_break_config(TIMER0,&timer_breakpara);

```

### 函数 timer\_break\_enable

函数timer\_break\_enable描述见下表:

表 3-689. 函数 timer\_break\_enable

|                     |   |
|---------------------|---|
| 函数名称                | timer_break_enable                              |
| 函数原型                | void timer_break_enable(uint32_t timer_periph); |
| 功能描述                | 使能TIMER的中止功能                                    |
| 先决条件                | 只有在TIMERx_CCHP寄存器的PROT [1:0] =00 时，才可修改         |
| 被调用函数               | -   |
| 输入参数{in}            |   |
| timer_periph        | TIMER外设   |
| TIMERx(x=0,7,19,20) | TIMER外设选择                                       |
| 输出参数{out}           |   |
| -                   | -   |
| 返回值                 |   |
| -                   | -   |

例如:

```

/* enable TIMER0 break function*/

timer_break_enable(TIMER0);

```

### 函数 timer\_break\_disable

函数timer\_break\_disable描述见下表:

表 3-690. 函数 timer\_break\_disable

|       |   |
|-------|---|
| 函数名称  | timer_break_disable                               |
| 函数原型  | void timer_break_disable (uint32_t timer_periph); |
| 功能描述  | 禁能TIMER的中止功能                                      |
| 先决条件  | 只有在TIMERx_CCHP寄存器的PROT [1:0] =00 时，才可修改           |
| 被调用函数 | -   |

| 输入参数{in}           |           |
|--------------------|-----------|
| timer_periph       | TIMER外设   |
| TIMEx(x=0,7,19,20) | TIMER外设选择 |
| 输出参数{out}          |           |
| -                  | -         |
| 返回值                |           |
| -                  | -         |

例如:

```
/* disable TIMER0 break function*/
```

```
timer_break_disable(TIMER0);
```

### 函数 timer\_automatic\_output\_enable

函数timer\_automatic\_output\_enable描述见下表:

表 3-691. 函数 timer\_automatic\_output\_enable

| 函数名称               | timer_automatic_output_enable                              |
|--------------------|--|
| 函数原型               | void timer_automatic_output_enable(uint32_t timer_periph); |
| 功能描述               | 自动输出使能   |
| 先决条件               | 只有在TIMEx_CCHP寄存器的PROT [1:0] =00 时, 才可修改                    |
| 被调函数               | -  |
| 输入参数{in}           |  |
| timer_periph       | TIMER外设  |
| TIMEx(x=0,7,19,20) | TIMER外设选择  |
| 输出参数{out}          |  |
| -                  | -  |
| 返回值                |  |
| -                  | -  |

例如:

```
/* enable TIMER0 output automatic function */
```

```
timer_automatic_output_enable(TIMER0);
```

### 函数 timer\_automatic\_output\_disable

函数timer\_automatic\_output\_disable描述见下表:

表 3-692. 函数 timer\_automatic\_output\_disable

|      |  |
|------|--|
| 函数名称 | timer_automatic_output_disable                               |
| 函数原型 | void timer_automatic_output_disable (uint32_t timer_periph); |

|                     |   |
|---------------------|---|
| 功能描述                | 自动输出禁能                                  |
| 先决条件                | 只有在TIMERx_CCHP寄存器的PROT [1:0] = 00时，才可修改 |
| 被调用函数               | -                                       |
| 输入参数{in}            |   |
| timer_periph        | TIMER外设                                 |
| TIMERx(x=0,7,19,20) | TIMER外设选择                               |
| 输出参数{out}           |   |
| -                   | -                                       |
| 返回值                 |   |
| -                   | -                                       |

例如：

```
/* disable TIMER0 output automatic function */
```

```
timer_automatic_output_disable(TIMER0);
```

### 函数 timer\_primary\_output\_config

函数timer\_primary\_output\_config描述见下表：

表 3-693. 函数 timer\_primary\_output\_config

|                     |  |
|---------------------|--|
| 函数名称                | timer_primary_output_config  |
| 函数原型                | void timer_primary_output_config(uint32_t timer_periph, ControlStatus newvalue); |
| 功能描述                | 所有的通道输出使能  |
| 先决条件                | -  |
| 被调用函数               | -  |
| 输入参数{in}            |  |
| timer_periph        | TIMER外设  |
| TIMERx(x=0,7,19,20) | TIMER外设选择  |
| 输入参数{in}            |  |
| newvalue            | 控制状态   |
| ENABLE              | 使能   |
| DISABLE             | 禁能   |
| 输出参数{out}           |  |
| -                   | -  |
| 返回值                 |  |
| -                   | -  |

例如：

```
/* enable TIMER0 primary output function */
```

```
timer_primary_output_config(TIMER0, ENABLE);
```

## 函数 timer\_channel\_output\_struct\_para\_init

函数timer\_channel\_output\_struct\_para\_init描述见下表:

表 3-694. 函数 timer\_channel\_output\_struct\_para\_init

|           |  |
|-----------|--|
| 函数名称      | timer_channel_output_struct_para_init  |
| 函数原型      | void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpara); |
| 功能描述      | 将TIMER通道输出参数结构体中所有参数初始化为默认值  |
| 先决条件      | -  |
| 被调用函数     | -  |
| 输入参数{in}  |  |
| ocpara    | 输出通道结构体, 详见 <a href="#">结构体timer oc parameter struct.</a>                      |
| 输出参数{out} |  |
| -         | -  |
| 返回值       |  |
| -         | -  |

例如:

```
/* initialize TIMER channel output parameter struct with a default value */
```

```
timer_oc_parameter_struct timer_ocinitpara;
```

```
timer_channel_output_struct_para_init(&timer_ocinitpara);
```

## 函数 timer\_channel\_output\_config

函数timer\_channel\_output\_config描述见下表:

表 3-695. 函数 timer\_channel\_output\_config

|                       |   |
|-----------------------|---|
| 函数名称                  | timer_channel_output_config   |
| 函数原型                  | void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara); |
| 功能描述                  | 外设TIMER的通道输出配置  |
| 先决条件                  | -   |
| 被调用函数                 | -   |
| 输入参数{in}              |   |
| timer_periph          | TIMER外设   |
| TIMERx(x=0,1,7,19,20) | 参考具体参数  |
| 输入参数{in}              |   |
| channel               | TIMER通道   |
| TIMER_CH_0            | 通道0, TIMERx(x=0,1,7,19,20)  |
| TIMER_CH_1            | 通道1, TIMERx(x=0,1,7,19,20)  |
| TIMER_CH_2            | 通道2, TIMERx(x=0,1,7,19,20)  |

|                   |  |
|-------------------|--|
| <i>TIMER_CH_3</i> | 通道3, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)              |
| 输入参数{in}          |  |
| <b>ocpara</b>     | 输出通道结构体, 详见 <a href="#">结构体timer oc parameter struct</a> |
| 输出参数{out}         |  |
| -                 | -  |
| 返回值               |  |
| -                 | -  |

例如:

```
/* 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);
```

### 函数 timer\_channel\_output\_mode\_config

函数timer\_channel\_output\_mode\_config描述见下表:

表 3-696. 函数 timer\_channel\_output\_mode\_config

|  |  |
|--|--|
| 函数名称                                   | timer_channel_output_mode_config   |
| 函数原型                                   | void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint16_t ocmode); |
| 功能描述                                   | 配置外设TIMER通道输出比较模式  |
| 先决条件                                   | -  |
| 被调用函数                                  | -  |
| 输入参数{in}                               |  |
| <b>timer_periph</b>                    | TIMER外设  |
| <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20) | 参考具体参数   |
| 输入参数{in}                               |  |
| <b>channel</b>                         | TIMER通道  |
| <i>TIMER_CH_0</i>                      | 通道0, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)  |
| <i>TIMER_CH_1</i>                      | 通道1, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)  |
| <i>TIMER_CH_2</i>                      | 通道2, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)  |
| <i>TIMER_CH_3</i>                      | 通道3, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)  |

|                               |                              |
|-------------------------------|------------------------------|
| <i>TIMER_MCH_0</i>            | 多模式通道0, TIMERx (x=0,7,19,20) |
| <i>TIMER_MCH_1</i>            | 多模式通道1, TIMERx (x=0,7,19,20) |
| <i>TIMER_MCH_2</i>            | 多模式通道2, TIMERx (x=0,7,19,20) |
| <i>TIMER_MCH_3</i>            | 多模式通道3, TIMERx (x=0,7,19,20) |
| 输入参数{in}                      |                              |
| <b>ocmode</b>                 | 通道输出比较模式                     |
| <i>TIMER_OC_MODE_TIMING</i>   | 时基模式                         |
| <i>TIMER_OC_MODE_ACTIVE</i>   | 匹配时设置为高                      |
| <i>TIMER_OC_MODE_INACTIVE</i> | 匹配时设置为低                      |
| <i>TIMER_OC_MODE_TOGGLE</i>   | 匹配时翻转                        |
| <i>TIMER_OC_MODE_LOW</i>      | 强制为低                         |
| <i>TIMER_OC_MODE_HIGH</i>     | 强制为高                         |
| <i>TIMER_OC_MODE_PWM0</i>     | PWM模式0                       |
| <i>TIMER_OC_MODE_PWM1</i>     | PWM模式1                       |
| 输出参数{out}                     |                              |
| -                             | -                            |
| 返回值                           |                              |
| -                             | -                            |

例如:

```
/* configure TIMER0 channel PWM 0 mode */
```

```
timer_channel_output_mode_config(TIMER0,TIMER_CH_0,TIMER_OC_MODE_PWM0);
```

### 函数 timer\_channel\_output\_pulse\_value\_config

函数timer\_channel\_output\_pulse\_value\_config描述见下表:

表 3-697. 函数 timer\_channel\_output\_pulse\_value\_config

|          |  |
|----------|--|
| 函数名称     | timer_channel_output_pulse_value_config  |
| 函数原型     | void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse); |
| 功能描述     | 配置外设TIMER的通道输出比较值  |
| 先决条件     | -  |
| 被调用函数    | -  |
| 输入参数{in} |  |

|  |  |
|--|--|
| <b>timer_periph</b>                    | TIMER外设                                      |
| <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20) | 参考具体参数                                       |
| <b>输入参数{in}</b>                        |  |
| <b>channel</b>                         | TIMER通道                                      |
| <i>TIMER_CH_0</i>                      | 通道0, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)  |
| <i>TIMER_CH_1</i>                      | 通道1, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)  |
| <i>TIMER_CH_2</i>                      | 通道2, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)  |
| <i>TIMER_CH_3</i>                      | 通道3, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)  |
| <i>TIMER_MCH_0</i>                     | 多模式通道0, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_MCH_1</i>                     | 多模式通道1, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_MCH_2</i>                     | 多模式通道2, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_MCH_3</i>                     | 多模式通道3, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <b>输入参数{in}</b>                        |  |
| <b>pulse</b>                           | 通道输出比较值 (0~65535)                            |
| <b>输出参数{out}</b>                       |  |
| -                                      | -  |
| <b>返回值</b>                             |  |
| -                                      | -  |

例如:

```
/* configure TIMER0 channel 0 output pulse value */
```

```
timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

### 函数 timer\_channel\_output\_shadow\_config

函数timer\_channel\_output\_shadow\_config描述见下表:

**表 3-698. 函数 timer\_channel\_output\_shadow\_config**

|  |  |
|--|--|
| <b>函数名称</b>                            | timer_channel_output_shadow_config   |
| <b>函数原型</b>                            | void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow); |
| <b>功能描述</b>                            | 配置TIMER通道输出比较影子寄存器功能   |
| <b>先决条件</b>                            | -  |
| <b>被调用函数</b>                           | -  |
| <b>输入参数{in}</b>                        |  |
| <b>timer_periph</b>                    | TIMER外设  |
| <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20) | 参考具体参数   |
| <b>输入参数{in}</b>                        |  |
| <b>channel</b>                         | TIMER通道  |
| <i>TIMER_CH_0</i>                      | 通道0, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)  |
| <i>TIMER_CH_1</i>                      | 通道1, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)  |



|                                 |  |
|---------------------------------|--|
| <i>TIMER_CH_2</i>               | 通道2, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)  |
| <i>TIMER_CH_3</i>               | 通道3, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)  |
| <i>TIMER_MCH_0</i>              | 多模式通道0, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_MCH_1</i>              | 多模式通道1, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_MCH_2</i>              | 多模式通道2, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_MCH_3</i>              | 多模式通道3, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| 输入参数{in}                        |  |
| <b>ocshadow</b>                 | 输出比较影子寄存器功能状态                                |
| <i>TIMER_OC_SHADOW_ENABLE</i>   | 通道输出比较影子寄存器使能                                |
| <i>TIMER_OC_SHADOW_DISABLE</i>  | 通道输出比较影子寄存器禁能                                |
| <i>TIMER_OMC_SHADOW_ENABLE</i>  | 多模式通道输出比较影子寄存器使能                             |
| <i>TIMER_OMC_SHADOW_DISABLE</i> | 多模式通道输出比较影子寄存器禁能                             |
| 输出参数{out}                       |  |
| -                               | -  |
| 返回值                             |  |
| -                               | -  |

例如:

```
/*configure TIMER0 channel 0 output shadow function */
```

```
timer_channel_output_shadow_config(TIMER0,                                TIMER_CH_0,
TIMER_OC_SHADOW_ENABLE);
```

### 函数 timer\_channel\_output\_clear\_config

函数timer\_channel\_output\_clear\_config描述见下表:

表 3-699. 函数 timer\_channel\_output\_clear\_config

|                     |  |
|---------------------|--|
| 函数名称                | timer_channel_output_clear_config  |
| 函数原型                | void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear); |
| 功能描述                | 配置TIMER的通道输出比较清除0功能  |
| 先决条件                | -  |
| 被调用函数               | -  |
| 输入参数{in}            |  |
| <b>timer_periph</b> | TIMER外设  |
| <i>TIMERx</i>       | 参考具体参数   |
| 输入参数{in}            |  |
| <b>channel</b>      | TIMER通道  |
| <i>TIMER_CH_0</i>   | 通道0, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)  |

|                                |                              |
|--------------------------------|------------------------------|
| <i>TIMER_CH_1</i>              | 通道1, TIMERx (x=0,1,7,19,20)  |
| <i>TIMER_CH_2</i>              | 通道2, TIMERx (x=0,1,7,19,20)  |
| <i>TIMER_CH_3</i>              | 通道3, TIMERx (x=0,1,7,19,20)  |
| <i>TIMER_MCH_0</i>             | 多模式通道0, TIMERx (x=0,7,19,20) |
| <i>TIMER_MCH_1</i>             | 多模式通道1, TIMERx (x=0,7,19,20) |
| <i>TIMER_MCH_2</i>             | 多模式通道2, TIMERx (x=0,7,19,20) |
| <i>TIMER_MCH_3</i>             | 多模式通道3, TIMERx (x=0,7,19,20) |
| 输入参数{in}                       |                              |
| <b>occlear</b>                 | 通道比较输出清0功能状态                 |
| <i>TIMER_OC_CLEAR_ENABLE</i>   | 通道比较输出清0功能使能                 |
| <i>TIMER_OC_CLEAR_DISABLE</i>  | 通道比较输出清0功能禁能                 |
| <i>TIMER_OMC_CLEAR_ENABLE</i>  | 多模式通道比较输出清0功能使能              |
| <i>TIMER_OMC_CLEAR_DISABLE</i> | 多模式通道比较输出清0功能禁能              |
| 输出参数{out}                      |                              |
| -                              | -                            |
| 返回值                            |                              |
| -                              | -                            |

例如:

```
/* configure TIMER0 channel 0 output clear function */
```

```
timer_channel_output_clear_config(TIMER0, TIMER_CH_0,  
TIMER_OC_CLEAR_ENABLE);
```

### 函数 timer\_channel\_output\_polarity\_config

函数timer\_channel\_output\_polarity\_config描述见下表:

表 3-700. 函数 timer\_channel\_output\_polarity\_config

|                     |  |
|---------------------|--|
| 函数名称                | timer_channel_output_polarity_config   |
| 函数原型                | void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity); |
| 功能描述                | 通道输出极性配置   |
| 先决条件                | -  |
| 被调用函数               | -  |
| 输入参数{in}            |  |
| <b>timer_periph</b> | TIMER外设  |
| <i>TIMERx</i>       | 参考具体参数   |
| 输入参数{in}            |  |
| <b>channel</b>      | TIMER通道  |

|                                |                              |
|--------------------------------|------------------------------|
| <i>TIMER_CH_0</i>              | 通道0, TIMERx (x=0,1,7,19,20)  |
| <i>TIMER_CH_1</i>              | 通道1, TIMERx (x=0,1,7,19,20)  |
| <i>TIMER_CH_2</i>              | 通道2, TIMERx (x=0,1,7,19,20)  |
| <i>TIMER_CH_3</i>              | 通道3, TIMERx (x=0,1,7,19,20)  |
| <i>TIMER_MCH_0</i>             | 多模式通道0, TIMERx (x=0,7,19,20) |
| <i>TIMER_MCH_1</i>             | 多模式通道1, TIMERx (x=0,7,19,20) |
| <i>TIMER_MCH_2</i>             | 多模式通道2, TIMERx (x=0,7,19,20) |
| <i>TIMER_MCH_3</i>             | 多模式通道3, TIMERx (x=0,7,19,20) |
| 输入参数{in}                       |                              |
| <b>ocpolarity</b>              | 通道输出极性                       |
| <i>TIMER_OC_POLARITY_HIGH</i>  | 通道输出极性高电平有效                  |
| <i>TIMER_OC_POLARITY_LOW</i>   | 通道输出极性低电平有效                  |
| <i>TIMER_OMC_POLARITY_HIGH</i> | 多模式通道输出极性高电平有效               |
| <i>TIMER_OMC_POLARITY_LOW</i>  | 多模式通道输出极性低电平有效               |
| 输出参数{out}                      |                              |
| -                              | -                            |
| 返回值                            |                              |
| -                              | -                            |

例如:

```
/* configure TIMER0 channel 0 output polarity */
```

```
timer_channel_output_polarity_config(TIMER0, TIMER_CH_0,  
TIMER_OC_POLARITY_HIGH);
```

### 函数 timer\_channel\_complementary\_output\_polarity\_config

函数timer\_channel\_complementary\_output\_polarity\_config描述见下表:

表 3-701. 函数 timer\_channel\_complementary\_output\_polarity\_config

|                            |   |
|----------------------------|---|
| 函数名称                       | timer_channel_complementary_output_polarity_config  |
| 函数原型                       | void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnpolarity); |
| 功能描述                       | 互补通道输出极性配置  |
| 先决条件                       | -   |
| 被调用函数                      | -   |
| 输入参数{in}                   |   |
| <b>timer_periph</b>        | TIMER外设   |
| <i>TIMERx(x=0,7,19,20)</i> | TIMER外设选择   |

| 输入参数{in}                       |               |
|--------------------------------|---------------|
| <b>channel</b>                 | TIMER通道       |
| <i>TIMER_CH_0</i>              | 通道0           |
| <i>TIMER_CH_1</i>              | 通道1           |
| <i>TIMER_CH_2</i>              | 通道2           |
| <i>TIMER_CH_3</i>              | 通道3           |
| 输入参数{in}                       |               |
| <b>ocpolarity</b>              | 互补通道输出极性      |
| <i>TIMER_OCN_POLARITY_HIGH</i> | 互补通道输出极性高电平有效 |
| <i>TIMER_OCN_POLARITY_LOW</i>  | 互补通道输出极性低电平有效 |
| 输出参数{out}                      |               |
| -                              | -             |
| 返回值                            |               |
| -                              | -             |

例如：

```
/* configure TIMER0 channel 0 complementary output polarity */
```

```
timer_channel_complementary_output_polarity_config(TIMER0, TIMER_CH_0,
TIMER_OCN_POLARITY_HIGH);
```

### 函数 timer\_channel\_output\_state\_config

函数timer\_channel\_output\_state\_config描述见下表：

表 3-702. 函数 timer\_channel\_output\_state\_config

| <b>函数名称</b>         | timer_channel_output_state_config  |
|---------------------|--|
| <b>函数原型</b>         | void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state); |
| <b>功能描述</b>         | 配置通道状态   |
| <b>先决条件</b>         | -  |
| <b>被调用函数</b>        | -  |
| 输入参数{in}            |  |
| <b>timer_periph</b> | TIMER外设  |
| <i>TIMERx</i>       | 参考具体参数   |
| 输入参数{in}            |  |
| <b>channel</b>      | TIMER通道  |
| <i>TIMER_CH_0</i>   | 通道0, TIMERx (x=0,1,7,19,20)  |
| <i>TIMER_CH_1</i>   | 通道1, TIMERx (x=0,1,7,19,20)  |
| <i>TIMER_CH_2</i>   | 通道2, TIMERx (x=0,1,7,19,20)  |
| <i>TIMER_CH_3</i>   | 通道3, TIMERx (x=0,1,7,19,20)  |
| <i>TIMER_MCH_0</i>  | 多模式通道0, TIMERx (x=0,7,19,20)   |

|                           |                              |
|---------------------------|------------------------------|
| <i>TIMER_MCH_1</i>        | 多模式通道1, TIMERx (x=0,7,19,20) |
| <i>TIMER_MCH_2</i>        | 多模式通道2, TIMERx (x=0,7,19,20) |
| <i>TIMER_MCH_3</i>        | 多模式通道3, TIMERx (x=0,7,19,20) |
| 输入参数{in}                  |                              |
| <b>state</b>              | 通道状态                         |
| <i>TIMER_CCX_ENABLE</i>   | 通道使能                         |
| <i>TIMER_CCX_DISABLE</i>  | 通道禁能                         |
| <i>TIMER_MCCX_ENABLE</i>  | 多模式通道使能                      |
| <i>TIMER_MCCX_DISABLE</i> | 多模式通道禁能                      |
| 输出参数{out}                 |                              |
| -                         | -                            |
| 返回值                       |                              |
| -                         | -                            |

例如:

```
/* configure TIMER0 channel 0 enable state */
```

```
timer_channel_output_state_config(TIMER0, TIMER_CH_0, TIMER_CCX_ENABLE);
```

### 函数 timer\_channel\_complementary\_output\_state\_config

函数timer\_channel\_complementary\_output\_state\_config描述见下表:

表 3-703. 函数 timer\_channel\_complementary\_output\_state\_config

|                            |   |
|----------------------------|---|
| 函数名称                       | timer_channel_complementary_output_state_config   |
| 函数原型                       | void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate); |
| 功能描述                       | 配置互补通道输出状态  |
| 先决条件                       | -   |
| 被调用函数                      | -   |
| 输入参数{in}                   |   |
| <b>timer_periph</b>        | TIMER外设   |
| <i>TIMERx(x=0,7,19,20)</i> | TIMER外设选择   |
| 输入参数{in}                   |   |
| <b>channel</b>             | TIMER通道   |
| <i>TIMER_CH_0</i>          | 通道0   |
| <i>TIMER_CH_1</i>          | 通道1   |
| <i>TIMER_CH_2</i>          | 通道2   |
| <i>TIMER_CH_3</i>          | 通道3   |

| 输入参数{in}           |        |
|--------------------|--------|
| state              | 互补通道状态 |
| TIMER_CCXN_ENABLE  | 互补通道使能 |
| TIMER_CCXN_DISABLE | 互补通道禁能 |
| 输出参数{out}          |        |
| -                  | -      |
| 返回值                |        |
| -                  | -      |

例如：

```
/* configure TIMER0 channel 0 complementary output enable state */
timer_channel_complementary_output_state_config(TIMER0, TIMER_CH_0,
TIMER_CCXN_ENABLE);
```

### 函数 timer\_channel\_input\_struct\_para\_init

函数timer\_channel\_input\_struct\_para\_init描述见下表：

表 3-704. 函数 timer\_channel\_input\_struct\_para\_init

| 函数名称      | timer_channel_input_struct_para_init  |
|-----------|---|
| 函数原型      | void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara); |
| 功能描述      | 将TIMER通道输入参数结构体中所有参数初始化为默认值   |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| icpara    | 通道输入结构体，详见 <a href="#">结构体timer ic parameter struct</a> 。                     |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

```
/* initialize TIMER channel input parameter struct with a default value */
timer_ic_parameter_struct timer_icinitpara;
timer_channel_input_struct_para_init(&timer_icinitpara);
```

### 函数 timer\_input\_capture\_config

函数timer\_input\_capture\_config描述见下表：

表 3-705. 函数 timer\_input\_capture\_config

|              |  |
|--------------|--|
| 函数名称         | timer_input_capture_config   |
| 函数原型         | void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara); |
| 功能描述         | 配置TIMER输入捕获参数  |
| 先决条件         | -  |
| 被调用函数        | timer_channel_input_capture_prescaler_config   |
| 输入参数{in}     |  |
| timer_periph | TIMER外设  |
| TIMERx       | 参考具体参数   |
| 输入参数{in}     |  |
| channel      | TIMER通道  |
| TIMER_CH_0   | 通道0, TIMERx (x=0,1,7,19,20)  |
| TIMER_CH_1   | 通道1, TIMERx (x=0,1,7,19,20)  |
| TIMER_CH_2   | 通道2, TIMERx (x=0,1,7,19,20)  |
| TIMER_CH_3   | 通道3, TIMERx (x=0,1,7,19,20)  |
| TIMER_MCH_0  | 多模式通道0, TIMERx (x=0,7,19,20)   |
| TIMER_MCH_1  | 多模式通道1, TIMERx (x=0,7,19,20)   |
| TIMER_MCH_2  | 多模式通道2, TIMERx (x=0,7,19,20)   |
| TIMER_MCH_3  | 多模式通道3, TIMERx (x=0,7,19,20)   |
| 输入参数{in}     |  |
| icpara       | 通道输入结构体, 详见 <a href="#">结构体timer_ic_parameter_struct</a> 。   |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| -            | -  |

例如:

```
/* configure TIMER0 input capture parameter */
timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);
```

#### 函数 timer\_channel\_input\_capture\_prescaler\_config

函数timer\_channel\_input\_capture\_prescaler\_config描述见下表:

表 3-706. 函数 timer\_channel\_input\_capture\_prescaler\_config

|                   |   |
|-------------------|---|
| 函数名称              | timer_channel_input_capture_prescaler_config  |
| 函数原型              | void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler); |
| 功能描述              | 配置TIMER通道输入捕获预分频值   |
| 先决条件              | -   |
| 被调用函数             | -   |
| 输入参数{in}          |   |
| timer_periph      | TIMER外设   |
| TIMERx            | 参考具体参数  |
| 输入参数{in}          |   |
| channel           | TIMER通道   |
| TIMER_CH_0        | 通道0, TIMERx (x=0,1,7,19,20)   |
| TIMER_CH_1        | 通道1, TIMERx (x=0,1,7,19,20)   |
| TIMER_CH_2        | 通道2, TIMERx (x=0,1,7,19,20)   |
| TIMER_CH_3        | 通道3, TIMERx (x=0,1,7,19,20)   |
| TIMER_MCH_0       | 多模式通道0, TIMERx (x=0,7,19,20)  |
| TIMER_MCH_1       | 多模式通道1, TIMERx (x=0,7,19,20)  |
| TIMER_MCH_2       | 多模式通道2, TIMERx (x=0,7,19,20)  |
| TIMER_MCH_3       | 多模式通道3, TIMERx (x=0,7,19,20)  |
| 输入参数{in}          |   |
| prescaler         | 通道输入捕获预分频值  |
| TIMER_IC_PSC_DIV1 | 不分频   |
| TIMER_IC_PSC_DIV2 | 2分频   |
| TIMER_IC_PSC_DIV4 | 4分频   |
| TIMER_IC_PSC_DIV8 | 8分频   |
| 输出参数{out}         |   |
| -                 | -   |
| 返回值               |   |
| -                 | -   |

例如:

```
/* configure TIMER0 channel 0 input capture prescaler value */
timer_channel_input_capture_prescaler_config(TIMER0, TIMER_CH_0,
TIMER_IC_PSC_DIV2);
```



**函数 timer\_channel\_capture\_value\_register\_read**

函数timer\_channel\_capture\_value\_register\_read描述见下表:

**表 3-707. 函数 timer\_channel\_capture\_value\_register\_read**

|              |  |
|--------------|--|
| 函数名称         | timer_channel_capture_value_register_read  |
| 函数原型         | uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel); |
| 功能描述         | 读取通道捕获值  |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| timer_periph | TIMER外设  |
| TIMERx       | 参考具体参数   |
| 输入参数{in}     |  |
| channel      | TIMER通道  |
| TIMER_CH_0   | 通道0, TIMERx (x=0,1,7,19,20)  |
| TIMER_CH_1   | 通道1, TIMERx (x=0,1,7,19,20)  |
| TIMER_CH_2   | 通道2, TIMERx (x=0,1,7,19,20)  |
| TIMER_CH_3   | 通道3, TIMERx (x=0,1,7,19,20)  |
| TIMER_MCH_0  | 多模式通道0, TIMERx (x=0,7,19,20)   |
| TIMER_MCH_1  | 多模式通道1, TIMERx (x=0,7,19,20)   |
| TIMER_MCH_2  | 多模式通道2, TIMERx (x=0,7,19,20)   |
| TIMER_MCH_3  | 多模式通道3, TIMERx (x=0,7,19,20)   |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| uint32_t     | 通道输入捕获值 (0~65535)  |

例如:

```
/* 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);
```

**函数 timer\_input\_pwm\_capture\_config**

函数timer\_input\_pwm\_capture\_config描述见下表:

**表 3-708. 函数 timer\_input\_pwm\_capture\_config**

|      |   |
|------|---|
| 函数名称 | timer_input_pwm_capture_config  |
| 函数原型 | void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm); |
| 功能描述 | 配置TIMER捕获PWM输入参数  |

|                       |  |
|-----------------------|--|
| 先决条件                  | -  |
| 被调用函数                 | timer_channel_input_capture_prescaler_config                 |
| 输入参数{in}              |  |
| timer_periph          | TIMER外设  |
| TIMERx(x=0,1,7,19,20) | TIMER外设选择  |
| 输入参数{in}              |  |
| channel               | TIMER通道  |
| TIMER_CH_0            | 通道0  |
| TIMER_CH_1            | 通道1  |
| 输入参数{in}              |  |
| icpwm                 | 输入PWM捕获结构体，详见 <a href="#">结构体timer_ic_parameter_struct</a> 。 |
| 输出参数{out}             |  |
| -                     | -  |
| 返回值                   |  |
| -                     | -  |

例如：

```
/* configure TIMER0 input pwm capture parameter */
timer_ic_parameter_struct timer_ic_initpara;

timer_ic_initpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_ic_initpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_ic_initpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_ic_initpara.icfilter = 0x0;

timer_input_pwm_capture_config(TIMER0, TIMER_CH_0, &timer_ic_initpara);
```

### 函数 timer\_hall\_mode\_config

函数timer\_hall\_mode\_config描述见下表：

表 3-709. 函数 timer\_hall\_mode\_config

|                       |  |
|-----------------------|--|
| 函数名称                  | timer_hall_mode_config   |
| 函数原型                  | void timer_hall_mode_config(uint32_t timer_periph, uint32_t hallmode); |
| 功能描述                  | 配置TIMER的HALL接口功能   |
| 先决条件                  | -  |
| 被调用函数                 | -  |
| 输入参数{in}              |  |
| timer_periph          | TIMER外设  |
| TIMERx(x=0,1,7,19,20) | TIMER外设选择  |

| 输入参数{in}                        |            |
|---------------------------------|------------|
| hallmode                        | HALL接口功能状态 |
| TIMER_HALLINTE<br>RFACE_ENABLE  | HALL接口使能   |
| TIMER_HALLINTE<br>RFACE_DISABLE | HALL接口禁能   |
| 输出参数{out}                       |            |
| -                               | -          |
| 返回值                             |            |
| -                               | -          |

例如：

```
/* configure TIMER0 hall sensor mode */
timer_hall_mode_config(TIMER0, TIMER_HALLINTERFACE_ENABLE);
```

### 函数 timer\_multi\_mode\_channel\_output\_parameter\_struct\_init

函数timer\_multi\_mode\_channel\_output\_parameter\_struct\_init描述见下表：

表 3-710. 函数 timer\_multi\_mode\_channel\_output\_parameter\_struct\_init

| 函数名称      | timer_multi_mode_channel_output_parameter_struct_init   |
|-----------|---|
| 函数原型      | void<br>timer_multi_mode_channel_output_parameter_struct_init(timer_omc_parameter_struct *omcpara); |
| 功能描述      | 将TIMER多模式通道输出参数结构体中所有参数初始化为默认值  |
| 先决条件      | -   |
| 被调用函数     | -   |
| 输入参数{in}  |   |
| omcpara   | 多模式通道输出参数结构体，详见 <a href="#">结构体timer_omc_parameter_struct</a> .                                     |
| 输出参数{out} |   |
| -         | -   |
| 返回值       |   |
| -         | -   |

例如：

```
/* initialize TIMER multi mode channel output parameter struct with a default value */
timer_omc_parameter_struct timer_omcinitpara;
timer_multi_mode_channel_output_parameter_struct_init(&timer_omcinitpara);
```

### 函数 timer\_multi\_mode\_channel\_output\_config

函数timer\_multi\_mode\_channel\_output\_config描述见下表：

表 3-711. 函数 timer\_multi\_mode\_channel\_output\_config

|                     |  |
|---------------------|--|
| 函数名称                | timer_multi_mode_channel_output_config   |
| 函数原型                | void timer_multi_mode_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_omc_parameter_struct *omcpara); |
| 功能描述                | 外设TIMER的多模式通道输出配置  |
| 先决条件                | -  |
| 被调用函数               | -  |
| 输入参数{in}            |  |
| timer_periph        | TIMER外设  |
| TIMERx(x=0,7,19,20) | 参考具体参数   |
| 输入参数{in}            |  |
| channel             | TIMER通道  |
| TIMER_MCH_0         | 多模式通道0, TIMERx (x=0,7,19,20)   |
| TIMER_MCH_1         | 多模式通道1, TIMERx (x=0,7,19,20)   |
| TIMER_MCH_2         | 多模式通道2, TIMERx (x=0,7,19,20)   |
| TIMER_MCH_3         | 多模式通道3, TIMERx (x=0,7,19,20)   |
| 输入参数{in}            |  |
| omcpara             | 多模式通道输出参数结构体, 详见 <a href="#">结构体timer_omc_parameter_struct</a> .   |
| 输出参数{out}           |  |
| -                   | -  |
| 返回值                 |  |
| -                   | -  |

例如:

```
/* configure TIMER0 multi mode channel 0 output function */
```

```
timer_omc_parameter_struct timer_omcinitpara;
```

```
omcpara->outputmode = TIMER_MCH_MODE_INDEPENDENTLY;
```

```
omcpara->outputstate = TIMER_MCCX_ENABLE;
```

```
omcpara->ocpolarity = TIMER_OMC_POLARITY_HIGH;
```

```
timer_multi_mode_channel_output_parameter_struct_init(TIMER0, TIMER_MCH_0, &timer_omcinitpara);
```

### 函数 timer\_multi\_mode\_channel\_mode\_config

函数timer\_multi\_mode\_channel\_mode\_config描述见下表:

表 3-712. 函数 timer\_multi\_mode\_channel\_mode\_config

|      |  |
|------|--|
| 函数名称 | timer_multi_mode_channel_mode_config   |
| 函数原型 | void timer_multi_mode_channel_mode_config(uint32_t timer_periph, uint32_t channel, uint32_t multi_mode_sel); |

|                              |                              |
|------------------------------|------------------------------|
| 功能描述                         | 外设TIMER多模式通道模式选择             |
| 先决条件                         | -                            |
| 被调用函数                        | -                            |
| 输入参数{in}                     |                              |
| timer_periph                 | TIMER外设                      |
| TIMERx(x=0,7,19,20)          | 参考具体参数                       |
| 输入参数{in}                     |                              |
| channel                      | TIMER通道                      |
| TIMER_MCH_0                  | 多模式通道0, TIMERx (x=0,7,19,20) |
| TIMER_MCH_1                  | 多模式通道1, TIMERx (x=0,7,19,20) |
| TIMER_MCH_2                  | 多模式通道2, TIMERx (x=0,7,19,20) |
| TIMER_MCH_3                  | 多模式通道3, TIMERx (x=0,7,19,20) |
| 输入参数{in}                     |                              |
| ocmode                       | 通道输出比较模式                     |
| TIMER_MCH_MODE_INDEPENDENTLY | 多模式通道为独立模式                   |
| TIMER_MCH_MODE_MIRRORED      | 多模式通道为镜像模式                   |
| TIMER_MCH_MODE_COMPLEMENTARY | 多模式通道为互补模式                   |
| 输出参数{out}                    |                              |
| -                            | -                            |
| 返回值                          |                              |
| -                            | -                            |

例如:

```
/* select TIMER0 multi mode channel 0 mode */
```

```
timer_multi_mode_channel_mode_config (TIMER0, TIMER_MCH_0,
TIMER_MCH_MODE_INDEPENDENTLY);
```

### 函数 timer\_input\_trigger\_source\_select

函数timer\_input\_trigger\_source\_select描述见下表:

表 3-713. 函数 timer\_input\_trigger\_source\_select

|      |  |
|------|--|
| 函数名称 | timer_input_trigger_source_select  |
| 函数原型 | void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger); |
| 功能描述 | TIMER的输入触发源选择  |
| 先决条件 | SMC[2:0] = 000   |

|                                 |   |
|---------------------------------|---|
| 被调用函数                           | -   |
| 输入参数{in}                        |   |
| timer_periph                    | TIMER外设                                   |
| TIMERx(x=0,1,7,19,20)           | TIMER外设选择                                 |
| 输入参数{in}                        |   |
| intrigger                       | 输入触发源                                     |
| TIMER_SMCFG_T<br>RGSEL_ITI0     | 内部触发输入0(ITI0, TIMERx(x=0,1,7,19,20))      |
| TIMER_SMCFG_T<br>RGSEL_ITI1     | 内部触发输入1(ITI1, TIMERx(x=0,1,7,19,20))      |
| TIMER_SMCFG_T<br>RGSEL_ITI2     | 内部触发输入2(ITI2, TIMERx(x=0,1,7,19,20))      |
| TIMER_SMCFG_T<br>RGSEL_ITI3     | 内部触发输入3(ITI3, TIMERx(x=0,1,7,19,20))      |
| TIMER_SMCFG_T<br>RGSEL_CIOF_ED  | CIO的边沿标志位(CIOF_ED, TIMERx(x=0,1,7,19,20)) |
| TIMER_SMCFG_T<br>RGSEL_CIOFE0   | 滤波后的通道0输入(CIOFE0, TIMERx(x=0,1,7,19,20))  |
| TIMER_SMCFG_T<br>RGSEL_CI1FE1   | 滤波后的通道1输入(CI1FE1, TIMERx(x=0,1,7,19,20))  |
| TIMER_SMCFG_T<br>RGSEL_ETIFP    | 滤波后的外部触发输入(ETIFP, TIMERx(x=0,1,7,19,20))  |
| TIMER_SMCFG_T<br>RGSEL_CI2FE2   | 滤波后的通道2输入(TIMERx(x=0,7,19,20))            |
| TIMER_SMCFG_T<br>RGSEL_CI3FE3   | 滤波后的通道3输入(TIMERx(x=0,7,19,20))            |
| TIMER_SMCFG_T<br>RGSEL_MCI0FEM0 | 滤波后的多模式通道0输入(TIMERx(x=0,7,19,20))         |
| TIMER_SMCFG_T<br>RGSEL_MCI1FEM1 | 滤波后的多模式通道1输入(TIMERx(x=0,7,19,20))         |
| TIMER_SMCFG_T<br>RGSEL_MCI2FEM2 | 滤波后的多模式通道2输入(TIMERx(x=0,7,19,20))         |
| TIMER_SMCFG_T<br>RGSEL_MCI3FEM3 | 滤波后的多模式通道3输入(TIMERx(x=0,7,19,20))         |
| 输出参数{out}                       |   |
| -                               | -   |
| 返回值                             |   |
| -                               | -   |

例如:

```
/* select TIMER0 input trigger source */
```

timer\_input\_trigger\_source\_select (TIMER0, TIMER\_SMCFG\_TRGSEL\_ITI0);

### 函数 timer\_master\_output\_trigger\_source\_select

函数timer\_master\_output\_trigger\_source\_select描述见下表:

表 3-714. 函数 timer\_master\_output\_trigger\_source\_select

|                           |   |
|---------------------------|---|
| 函数名称                      | timer_master_output_trigger_source_select   |
| 函数原型                      | void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outtrigger); |
| 功能描述                      | 选择TIMER主模式输出触发  |
| 先决条件                      | -   |
| 被调用函数                     | -   |
| 输入参数{in}                  |   |
| timer_periph              | TIMER外设   |
| TIMERx(x=0,1,5,6,7,19,20) | TIMER外设选择   |
| 输入参数{in}                  |   |
| outtrigger                | 输出触发源   |
| TIMER_TRI_OUT_SRC_RESET   | UPG位作为TRGO (TIMERx(x=0,1,5,6,7,19,20))  |
| TIMER_TRI_OUT_SRC_ENABLE  | TIMER_CTL0_CEN使能信号作为TRGO(TIMERx(x=0,1,5,6,7,19,20))   |
| TIMER_TRI_OUT_SRC_UPDATE  | 更新事件作为TRGO(TIMERx(x=0,1,5,6,7,19,20))   |
| TIMER_TRI_OUT_SRC_CH0     | 通道0捕获/比较事件作为TRGO(TIMERx(x=0,1,5,6,7,19,20))   |
| TIMER_TRI_OUT_SRC_O0CPRE  | O0CPRE作为触发输出TRGO(TIMERx(x=0,1,7,19,20))   |
| TIMER_TRI_OUT_SRC_O1CPRE  | O1CPRE作为触发输出TRGO(TIMERx(x=0,1,7,19,20))   |
| TIMER_TRI_OUT_SRC_O2CPRE  | O2CPRE作为触发输出TRGO(TIMERx(x=0,1,7,19,20))   |
| TIMER_TRI_OUT_SRC_O3CPRE  | O3CPRE作为触发输出TRGO(TIMERx(x=0,1,7,19,20))   |
| 输出参数{out}                 |   |
| -                         | -   |
| 返回值                       |   |
| -                         | -   |

例如:

```
/* select TIMER0 master mode output trigger source */
```

```
timer_master_output_trigger_source_select(TIMER0, TIMER_TRI_OUT_SRC_RESET);
```

## 函数 timer\_slave\_mode\_select

函数timer\_slave\_mode\_select描述见下表:

表 3-715. 函数 timer\_slave\_mode\_select

|                            |  |
|----------------------------|--|
| 函数名称                       | timer_slave_mode_select  |
| 函数原型                       | void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode); |
| 功能描述                       | TIMER从模式配置   |
| 先决条件                       | -  |
| 被调用函数                      | -  |
| 输入参数{in}                   |  |
| timer_periph               | TIMER外设  |
| TIMERx<br>(x=0,1,7,19,20)  | TIMER外设选择  |
| 输入参数{in}                   |  |
| slavemode                  | 从模式  |
| TIMER_SLAVE_MODE_DISABLE   | 关闭从模式  |
| TIMER_QUAD_DECODER_MODE0   | 正交译码器模式0   |
| TIMER_QUAD_DECODER_MODE1   | 正交译码器模式1   |
| TIMER_QUAD_DECODER_MODE2   | 正交译码器模式2   |
| TIMER_SLAVE_MODE_RESTART   | 复位模式   |
| TIMER_SLAVE_MODE_PAUSE     | 暂停模式   |
| TIMER_SLAVE_MODE_EVENT     | 事件模式   |
| TIMER_SLAVE_MODE_EXTERNAL0 | 外部时钟模式0  |
| 输出参数{out}                  |  |
| -                          | -  |
| 返回值                        |  |
| -                          | -  |

例如:

```
/* select TIMER0 slave mode */
```

```
timer_slave_mode_select(TIMER0, TIMER_QUAD_DECODER_MODE0);
```



**函数 timer\_master\_slave\_mode\_config**

函数timer\_master\_slave\_mode\_config描述见下表:

**表 3-716. 函数 timer\_master\_slave\_mode\_config**

|                                 |   |
|---------------------------------|---|
| 函数名称                            | timer_master_slave_mode_config  |
| 函数原型                            | void timer_master_slave_mode_config(uint32_t timer_periph, uint32_t masterslave); |
| 功能描述                            | TIMER主从模式配置   |
| 先决条件                            | -   |
| 被调用函数                           | -   |
| 输入参数{in}                        |   |
| timer_periph                    | TIMER外设   |
| TIMERx(x=0,1,7,19,20)           | TIMER外设选择   |
| 输入参数{in}                        |   |
| masterslave                     | 主从模式使能状态  |
| TIMER_MASTER_SLAVE_MODE_ENABLE  | 主从模式使能  |
| TIMER_MASTER_SLAVE_MODE_DISABLE | 主从模式禁能  |
| 输出参数{out}                       |   |
| -                               | -   |
| 返回值                             |   |
| -                               | -   |

例如:

```
/* configure TIMER0 master slave mode */
```

```
timer_master_slave_mode_config(TIMER0, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

**函数 timer\_external\_trigger\_config**

函数timer\_external\_trigger\_config描述见下表:

**表 3-717. 函数 timer\_external\_trigger\_config**

|          |  |
|----------|--|
| 函数名称     | timer_external_trigger_config  |
| 函数原型     | void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter); |
| 功能描述     | 配置TIMER外部触发输入  |
| 先决条件     | -  |
| 被调用函数    | -  |
| 输入参数{in} |  |

|  |                |
|--|----------------|
| <b>timer_periph</b>                    | TIMER外设        |
| <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20) | TIMER外设选择      |
| <b>输入参数{in}</b>                        |                |
| <b>extprescaler</b>                    | 外部触发预分频        |
| <i>TIMER_EXT_TRI_PSC_OFF</i>           | 不分频            |
| <i>TIMER_EXT_TRI_PSC_DIV2</i>          | 2分频            |
| <i>TIMER_EXT_TRI_PSC_DIV4</i>          | 4分频            |
| <i>TIMER_EXT_TRI_PSC_DIV8</i>          | 8分频            |
| <b>输入参数{in}</b>                        |                |
| <b>expolarity</b>                      | 外部触发输入极性       |
| <i>TIMER_ETP_FALLING</i>               | 低电平或者下降沿有效     |
| <i>TIMER_ETP_RISING</i>                | 高电平或者上升沿有效     |
| <b>输入参数{in}</b>                        |                |
| <b>extfilter</b>                       | 外部触发滤波控制（0~15） |
| <b>输出参数{out}</b>                       |                |
| -                                      | -              |
| <b>返回值</b>                             |                |
| -                                      | -              |

例如：

```
/* configure TIMER0 external trigger input */
```

```
timer_external_trigger_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TIMER_ETP_FALLING, 10);
```

### 函数 timer\_quadrature\_decoder\_mode\_config

函数timer\_quadrature\_decoder\_mode\_config描述见下表：

**表 3-718. 函数 timer\_quadrature\_decoder\_mode\_config**

|                 |  |
|-----------------|--|
| <b>函数名称</b>     | timer_quadrature_decoder_mode_config   |
| <b>函数原型</b>     | void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity); |
| <b>功能描述</b>     | TIMER配置为编码器模式  |
| <b>先决条件</b>     | -  |
| <b>被调用函数</b>    | -  |
| <b>输入参数{in}</b> |  |

|  |   |
|--|---|
| <b>timer_periph</b>                    | TIMER外设                                 |
| <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20) | TIMER外设选择                               |
| <b>输入参数{in}</b>                        |   |
| <b>decomode</b>                        | 编码器模式                                   |
| <i>TIMER_QUAD_DECODER_MODE0</i>        | 根据CI0FE0的电平，计数器在CI1FE1的边沿向上/下计数         |
| <i>TIMER_QUAD_DECODER_MODE1</i>        | 根据CI1FE1的电平，计数器在CI0FE0的边沿向上/下计数         |
| <i>TIMER_QUAD_DECODER_MODE2</i>        | 根据另一个信号的输入电平，计数器在CI0FE0和CI1FE1的边沿向上/下计数 |
| <b>输入参数{in}</b>                        |   |
| <b>ic0polarity</b>                     | IC0极性                                   |
| <i>TIMER_IC_POLARITY_RISING</i>        | 捕获上升边沿                                  |
| <i>TIMER_IC_POLARITY_FALLING</i>       | 捕获下降边沿                                  |
| <i>TIMER_IC_POLARITY_BOTH_EDGE</i>     | 双边沿有效                                   |
| <b>输入参数{in}</b>                        |   |
| <b>ic1polarity</b>                     | IC1极性                                   |
| <i>TIMER_IC_POLARITY_RISING</i>        | 捕获上升边沿                                  |
| <i>TIMER_IC_POLARITY_FALLING</i>       | 捕获下降边沿                                  |
| <i>TIMER_IC_POLARITY_BOTH_EDGE</i>     | 双边沿有效                                   |
| <b>输出参数{out}</b>                       |   |
| -                                      | -                                       |
| <b>返回值</b>                             |   |
| -                                      | -                                       |

例如：

```
/* configure TIMER0 quadrature decoder mode */
```

```
timer_quadrature_decoder_mode_config(TIMER0, TIMER_QUAD_DECODER_MODE0,  
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

### 函数 timer\_internal\_clock\_config

函数timer\_internal\_clock\_config描述见下表：

表 3-719. 函数 timer\_internal\_clock\_config

|      |                             |
|------|-----------------------------|
| 函数名称 | timer_internal_clock_config |
|------|-----------------------------|

|                       |  |
|-----------------------|--|
| 函数原型                  | void timer_internal_clock_config(uint32_t timer_periph); |
| 功能描述                  | TIMER配置为内部时钟模式   |
| 先决条件                  | -  |
| 被调用函数                 | -  |
| 输入参数{in}              |  |
| timer_periph          | TIMER外设  |
| TIMERx(x=0,1,7,19,20) | TIMER外设选择  |
| 输出参数{out}             |  |
| -                     | -  |
| 返回值                   |  |
| -                     | -  |

例如:

```
/* configure TIMER0 internal clock mode */
```

```
timer_internal_clock_config (TIMER0);
```

### 函数 timer\_internal\_trigger\_as\_external\_clock\_config

函数timer\_internal\_trigger\_as\_external\_clock\_config描述见下表:

表 3-720. 函数 timer\_internal\_trigger\_as\_external\_clock\_config

|                             |  |
|-----------------------------|--|
| 函数名称                        | timer_internal_trigger_as_external_clock_config  |
| 函数原型                        | void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger); |
| 功能描述                        | 配置TIMER的内部触发为时钟源   |
| 先决条件                        | -  |
| 被调用函数                       | timer_input_trigger_source_select  |
| 输入参数{in}                    |  |
| timer_periph                | TIMER外设  |
| TIMERx(x=0,1,7,19,20)       | TIMER外设选择  |
| 输入参数{in}                    |  |
| intrigger                   | 被选择的内部触发源  |
| TIMER_SMCFG_T<br>RGSEL_ITI0 | 选择内部触发0 (ITI0)为时钟源   |
| TIMER_SMCFG_T<br>RGSEL_ITI1 | 选择内部触发1 (ITI1)为时钟源   |
| TIMER_SMCFG_T<br>RGSEL_ITI2 | 选择内部触发2 (ITI2)为时钟源   |
| TIMER_SMCFG_T<br>RGSEL_ITI3 | 选择内部触发3 (ITI3)为时钟源   |
| 输出参数{out}                   |  |

|     |   |
|-----|---|
| -   | - |
| 返回值 |   |
| -   | - |

例如：

```
/* configure TIMER0 the internal trigger ITIO as external clock input */
```

```
timer_internal_trigger_as_external_clock_config(TIMER0, TIMER_SMCFG_TRGSEL_ITIO);
```

### 函数 timer\_external\_trigger\_as\_external\_clock\_config

函数timer\_external\_trigger\_as\_external\_clock\_config描述见下表：

表 3-721. 函数 timer\_external\_trigger\_as\_external\_clock\_config

|                             |   |
|-----------------------------|---|
| 函数名称                        | timer_external_trigger_as_external_clock_config   |
| 函数原型                        | void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t extrigger, uint16_t expolarity, uint32_t extfilter); |
| 功能描述                        | 配置TIMER的外部触发作为时钟源   |
| 先决条件                        | -   |
| 被调用函数                       | timer_input_trigger_source_select   |
| 输入参数{in}                    |   |
| timer_periph                | TIMER外设   |
| TIMERx(x=0,1,7,19,20)       | TIMER外设选择   |
| 输入参数{in}                    |   |
| extrigger                   | 外部触发源   |
| TIMER_SMCFG_TRGSEL_CIOF_ED  | CIO的边沿标志 (CIOF_ED)  |
| TIMER_SMCFG_TRGSEL_CIOFE0   | 滤波后的通道0输入 (CIOFE0)  |
| TIMER_SMCFG_TRGSEL_C11FE1   | 滤波后的通道1输入 (C11FE1)  |
| 输入参数{in}                    |   |
| expolarity                  | 外部触发源极性   |
| TIMER_IC_POLARITY_RISING    | 外部触发源高电平或者上升沿有效   |
| TIMER_IC_POLARITY_FALLING   | 外部触发源低电平或者下降沿有效   |
| TIMER_IC_POLARITY_BOTH_EDGE | 外部触发双边沿有效   |
| 输入参数{in}                    |   |
| extfilter                   | 滤波参数 (0~15)   |
| 输出参数{out}                   |   |
| -                           | -   |

| 返回值 |   |
|-----|---|
| -   | - |

例如：

```
/* 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);
```

### 函数 timer\_external\_clock\_mode0\_config

函数timer\_external\_clock\_mode0\_config描述见下表：

表 3-722. 函数 timer\_external\_clock\_mode0\_config

| 函数名称                       | timer_external_clock_mode0_config  |
|----------------------------|--|
| 函数原型                       | void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter); |
| 功能描述                       | 配置TIMER外部时钟模式0，ETI作为时钟源  |
| 先决条件                       | -  |
| 被调用函数                      | timer_external_trigger_config  |
| 输入参数{in}                   |  |
| timer_periph               | TIMER外设  |
| TIMERx(x=0,1,7,19,20)      | TIMER外设选择  |
| 输入参数{in}                   |  |
| extprescaler               | ETI触发源预分频值   |
| TIMER_EXT_TRI_P<br>SC_OFF  | 不分频  |
| TIMER_EXT_TRI_P<br>SC_DIV2 | 2分频  |
| TIMER_EXT_TRI_P<br>SC_DIV4 | 4分频  |
| TIMER_EXT_TRI_P<br>SC_DIV8 | 8分频  |
| 输入参数{in}                   |  |
| expolarity                 | ETI触发源极性   |
| TIMER_ETP_FALLI<br>NG      | 下降沿或者低电平有效   |
| TIMER_ETP_RISIN<br>G       | 上升沿或者高电平有效   |
| 输入参数{in}                   |  |
| extfilter                  | ETI触发源滤波参数（0~15）   |
| 输出参数{out}                  |  |
| -                          | -  |

| 返回值 |   |
|-----|---|
| -   | - |

例如:

```
/* configure TIMER0 the external clock mode0 */
```

```
timer_external_clock_mode0_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TIMER_ETP_FALLING, 0);
```

### 函数 timer\_external\_clock\_mode1\_config

函数timer\_external\_clock\_mode1\_config描述见下表:

表 3-723. 函数 timer\_external\_clock\_mode1\_config

| 函数名称                   | timer_external_clock_mode1_config  |
|------------------------|--|
| 函数原型                   | void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter); |
| 功能描述                   | 配置TIMER外部时钟模式1   |
| 先决条件                   | -  |
| 被调用函数                  | timer_external_trigger_config  |
| 输入参数{in}               |  |
| timer_periph           | TIMER外设  |
| TIMERx(x=0,1,7,19,20)  | TIMER外设选择  |
| 输入参数{in}               |  |
| extprescaler           | ETI触发源预分频值   |
| TIMER_EXT_TRI_PSC_OFF  | 不分频  |
| TIMER_EXT_TRI_PSC_DIV2 | 2分频  |
| TIMER_EXT_TRI_PSC_DIV4 | 4分频  |
| TIMER_EXT_TRI_PSC_DIV8 | 8分频  |
| 输入参数{in}               |  |
| expolarity             | ETI触发源极性   |
| TIMER_ETP_FALLING      | 下降沿或者低电平有效   |
| TIMER_ETP_RISING       | 上升沿或者高电平有效   |
| 输入参数{in}               |  |
| extfilter              | ETI触发源滤波参数 (0~15)  |
| 输出参数{out}              |  |
| -                      | -  |

| 返回值 |   |
|-----|---|
| -   | - |

例如：

```
/* configure TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 0);
```

### 函数 timer\_external\_clock\_mode1\_disable

函数timer\_external\_clock\_mode1\_disable描述见下表：

表 3-724. 函数 timer\_external\_clock\_mode1\_disable

|                       |   |
|-----------------------|---|
| 函数名称                  | timer_external_clock_mode1_disable                              |
| 函数原型                  | void timer_external_clock_mode1_disable(uint32_t timer_periph); |
| 功能描述                  | TIMER外部时钟模式1禁能  |
| 先决条件                  | -   |
| 被调用函数                 | -   |
| 输入参数{in}              |   |
| timer_periph          | TIMER外设   |
| TIMERx(x=0,1,7,19,20) | TIMER外设选择   |
| 输出参数{out}             |   |
| -                     | -   |
| 返回值                   |   |
| -                     | -   |

例如：

```
/* disable TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_disable(TIMER0);
```

### 函数 timer\_channel\_remap\_config

函数timer\_channel\_remap\_config描述见下表：

表 3-725. 函数 timer\_channel\_remap\_config

|              |   |
|--------------|---|
| 函数名称         | timer_channel_remap_config  |
| 函数原型         | void timer_channel_remap_config(uint32_t timer_periph, uint32_t remap); |
| 功能描述         | 配置 TIMER 通道重映射功能  |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| timer_periph | TIMER 外设  |



|                                |                             |
|--------------------------------|-----------------------------|
| <i>TIMERx(x=1)</i>             | TIMER 外设选择                  |
| 输入参数{in}                       |                             |
| <b>remap</b>                   | 重映射功能选择                     |
| <i>TIMER1_CIO_RMP_GPIO</i>     | TIMER1 通道 0 输入连接到 GPIO      |
| <i>TIMER1_CIO_RMP_LXTAL</i>    | TIMER1 通道 0 输入连接到 LXTAL     |
| <i>TIMER1_CIO_RMP_HXTAL</i>    | TIMER1 通道 0 输入连接到 HXTAL/128 |
| <i>TIMER1_CIO_RMP_CKOUTSEL</i> | TIMER1 通道 0 输入连接到 CKOUTSEL  |
| 输出参数{out}                      |                             |
| -                              | -                           |
| 返回值                            |                             |
| -                              | -                           |

例如：

```
/* configure TIMER1 channel 0 input is connected to GPIO */
```

```
timer_channel_remap_config(TIMER1, TIMER1_CIO_RMP_GPIO);
```

### 函数 timer\_write\_chxval\_register\_config

函数timer\_write\_chxval\_register\_config描述见下表：

表 3-726. 函数 timer\_write\_chxval\_register\_config

|                              |   |
|------------------------------|---|
| 函数名称                         | timer_write_chxval_register_config  |
| 函数原型                         | void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel); |
| 功能描述                         | 配置TIMER写CHxVAL选择位   |
| 先决条件                         | -   |
| 被调用函数                        | -   |
| 输入参数{in}                     |   |
| <b>timer_periph</b>          | TIMER外设   |
| <i>TIMERx(x=0,1,7,19,20)</i> | TIMER外设选择   |
| 输入参数{in}                     |   |
| <b>ccsel</b>                 | 写CHxVAL寄存器选择位   |
| <i>TIMER_CHVSEL_DISABLE</i>  | 无影响   |
| <i>TIMER_CHVSEL_ENABLE</i>   | 当写入捕获比较寄存器的值与寄存器当前值相等时，写入操作无效。  |
| 输出参数{out}                    |   |
| -                            | -   |
| 返回值                          |   |

例如：

```
/* configure TIMER0 write CHxVAL register selection */
timer_write_chxval_register_config(TIMER0, TIMER_CHVSEL_ENABLE);
```

### 函数 timer\_output\_value\_selection\_config

函数timer\_output\_value\_selection\_config描述见下表：

表 3-727. 函数 timer\_output\_value\_selection\_config

|                         |   |
|-------------------------|---|
| 函数名称                    | timer_output_value_selection_config   |
| 函数原型                    | void timer_output_value_selection_config(uint32_t timer_periph, uint16_t outsel); |
| 功能描述                    | 配置TIMER输出值选择位   |
| 先决条件                    | -   |
| 被调用函数                   | -   |
| 输入参数{in}                |   |
| timer_periph            | TIMER外设   |
| TIMERx<br>(x=0,7,19,20) | TIMER外设选择   |
| 输入参数{in}                |   |
| ccsel                   | 输出值选择位  |
| TIMER_OUTSEL_DISABLE    | 无影响   |
| TIMER_OUTSEL_ENABLE     | 如果POEN位与IOS位均为0，则输出无效。  |
| 输出参数{out}               |   |
| -                       | -   |
| 返回值                     |   |
| -                       | -   |

例如：

```
/* configure TIMER output value selection */
timer_output_value_selection_config(TIMER0, TIMER_OUTSEL_ENABLE);
```

### 函数 timer\_output\_match\_pulse\_select

函数timer\_output\_match\_pulse\_select描述见下表：

表 3-728. 函数 timer\_output\_match\_pulse\_select

|      |   |
|------|---|
| 函数名称 | timer_output_match_pulse_select   |
| 函数原型 | void timer_output_match_pulse_select(uint32_t timer_periph, uint32_t channel, uint16_t pulsesel); |

|                             |                          |
|-----------------------------|--------------------------|
| 功能描述                        | 通道TIMER输出匹配脉冲选择          |
| 先决条件                        | -                        |
| 被调用函数                       | -                        |
| 输入参数{in}                    |                          |
| timer_periph                | TIMER外设                  |
| TIMERx(x=0,7,19,20)         | 参考具体参数                   |
| 输入参数{in}                    |                          |
| channel                     | TIMER通道                  |
| TIMER_CH_0                  | 通道0, TIMERx(x=0,7,19,20) |
| TIMER_CH_1                  | 通道1, TIMERx(x=0,7,19,20) |
| TIMER_CH_2                  | 通道2, TIMERx(x=0,7,19,20) |
| TIMER_CH_3                  | 通道3, TIMERx(x=0,7,19,20) |
| 输入参数{in}                    |                          |
| pulsesel                    | 输出匹配脉冲选择                 |
| TIMER_PULSE_OUTPUT_NORMAL   | 通道输出正常                   |
| TIMER_PULSE_OUTPUT_CNT_UP   | 仅在向上计数时, 通道输出脉冲          |
| TIMER_PULSE_OUTPUT_CNT_DOWN | 仅在向下计数时, 通道输出脉冲          |
| TIMER_PULSE_OUTPUT_CNT_BOTH | 向上/向下计数时, 通道输出脉冲         |
| 输出参数{out}                   |                          |
| -                           | -                        |
| 返回值                         |                          |
| -                           | -                        |

例如:

```
/* configure TIMER0 channel 0 match pulse selection */
```

```
timer_output_match_pulse_select(TIMER0, TIMER_CH_0,
TIMER_PULSE_OUTPUT_CNT_UP);
```

### 函数 timer\_channel\_composite\_pwm\_mode\_config

函数timer\_channel\_composite\_pwm\_mode\_config描述见下表:

表 3-729. 函数 timer\_channel\_composite\_pwm\_mode\_config

|      |  |
|------|--|
| 函数名称 | timer_channel_composite_pwm_mode_config  |
| 函数原型 | void timer_channel_composite_pwm_mode_config(uint32_t timer_periph, uint32_t channel, ControlStatus newvalue); |
| 功能描述 | 配置TIMER的复合PWM模式  |
| 先决条件 | -  |

|                             |                          |
|-----------------------------|--------------------------|
| 被调用函数                       | -                        |
| 输入参数{in}                    |                          |
| timer_periph                | TIMER外设                  |
| TIMERx(x=0,7,19,20)         | 参考具体参数                   |
| 输入参数{in}                    |                          |
| channel                     | TIMER通道                  |
| TIMER_CH_0                  | 通道0, TIMERx(x=0,7,19,20) |
| TIMER_CH_1                  | 通道1, TIMERx(x=0,7,19,20) |
| TIMER_CH_2                  | 通道2, TIMERx(x=0,7,19,20) |
| TIMER_CH_3                  | 通道3, TIMERx(x=0,7,19,20) |
| 输入参数{in}                    |                          |
| pulsesel                    | 输出匹配脉冲选择                 |
| TIMER_PULSE_OUTPUT_NORMAL   | 通道输出正常                   |
| TIMER_PULSE_OUTPUT_CNT_UP   | 仅在向上计数时, 通道输出脉冲          |
| TIMER_PULSE_OUTPUT_CNT_DOWN | 仅在向下计数时, 通道输出脉冲          |
| TIMER_PULSE_OUTPUT_CNT_BOTH | 向上/向下计数时, 通道输出脉冲         |
| 输出参数{out}                   |                          |
| -                           | -                        |
| 返回值                         |                          |
| -                           | -                        |

例如:

```
/* configure the TIMER composite PWM mode */
```

```
timer_channel_composite_pwm_mode_config (TIMER0, TIMER_CH_0, ENABLE);
```

### 函数 timer\_channel\_composite\_pwm\_mode\_output\_pulse\_value\_config

函数timer\_channel\_composite\_pwm\_mode\_output\_pulse\_value\_config描述见下表:

表 3-730. 函数 timer\_channel\_composite\_pwm\_mode\_output\_pulse\_value\_config

|          |  |
|----------|--|
| 函数名称     | timer_channel_composite_pwm_mode_output_pulse_value_config   |
| 函数原型     | void<br>timer_channel_composite_pwm_mode_output_pulse_value_config(uint32_t timer_periph, uint32_t channel, uint32_t pulse, uint32_t add_pulse); |
| 功能描述     | 配置TIMER的复合PWM模式输出脉冲值   |
| 先决条件     | -  |
| 被调用函数    | -  |
| 输入参数{in} |  |

|                                      |   |
|--------------------------------------|---|
| <b>timer_periph</b>                  | TIMER外设                                   |
| <i>TIMERx</i> ( <i>x</i> =0,7,19,20) | 参考具体参数                                    |
| 输入参数{in}                             |   |
| <b>channel</b>                       | TIMER通道                                   |
| <i>TIMER_CH_0</i>                    | 通道0, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_CH_1</i>                    | 通道1, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_CH_2</i>                    | 通道2, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_CH_3</i>                    | 通道3, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| 输入参数{in}                             |   |
| <b>pulse</b>                         | 通道比较值 (0~65535)                           |
| 输入参数{in}                             |   |
| <b>add_pulse</b>                     | 通道附加比较值 (0~65535)                         |
| 输出参数{out}                            |   |
| -                                    | -   |
| 返回值                                  |   |
| -                                    | -   |

例如:

```
/* configure TIMER0 channel 0 output pulse value */
```

```
timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399, 3999);
```

### 函数 timer\_channel\_additional\_compare\_value\_config

函数timer\_channel\_additional\_compare\_value\_config描述见下表:

表 3-731. 函数 timer\_channel\_additional\_compare\_value\_config

|                                      |   |
|--------------------------------------|---|
| <b>函数名称</b>                          | timer_channel_additional_compare_value_config   |
| <b>函数原型</b>                          | void timer_channel_additional_compare_value_config(uint32_t timer_periph, uint16_t channel, uint32_t value) |
| <b>功能描述</b>                          | 配置TIMER通道附加比较寄存器  |
| <b>先决条件</b>                          | -   |
| <b>被调用函数</b>                         | -   |
| 输入参数{in}                             |   |
| <b>timer_periph</b>                  | TIMER外设   |
| <i>TIMERx</i> ( <i>x</i> =0,7,19,20) | 参考具体参数  |
| 输入参数{in}                             |   |
| <b>channel</b>                       | TIMER通道   |
| <i>TIMER_CH_0</i>                    | 通道0, <i>TIMERx</i> ( <i>x</i> =0,7,19,20)   |
| <i>TIMER_CH_1</i>                    | 通道1, <i>TIMERx</i> ( <i>x</i> =0,7,19,20)   |
| <i>TIMER_CH_2</i>                    | 通道2, <i>TIMERx</i> ( <i>x</i> =0,7,19,20)   |
| <i>TIMER_CH_3</i>                    | 通道3, <i>TIMERx</i> ( <i>x</i> =0,7,19,20)   |

| 输入参数{in}  |                  |
|-----------|------------------|
| value     | 通道附加比较值（0~65535） |
| 输出参数{out} |                  |
| -         | -                |
| 返回值       |                  |
| -         | -                |

例如：

```
/* configure TIMER0 channel 0 additional compare value */
```

```
timer_channel_additional_compare_value_config (TIMER0, TIMER_CH_0, 399);
```

### 函数 timer\_channel\_additional\_output\_shadow\_config

函数timer\_channel\_additional\_output\_shadow\_config描述见下表：

表 3-732. 函数 timer\_channel\_additional\_output\_shadow\_config

| 函数名称                     | timer_channel_additional_output_shadow_config  |
|--------------------------|--|
| 函数原型                     | void timer_channel_additional_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t aocshadow); |
| 功能描述                     | 配置TIMER通道附加输出比较影子寄存器功能   |
| 先决条件                     | -  |
| 被调用函数                    | -  |
| 输入参数{in}                 |  |
| timer_periph             | TIMER外设  |
| TIMERx(x=0,1,7,19,20)    | 参考具体参数   |
| 输入参数{in}                 |  |
| channel                  | TIMER通道  |
| TIMER_CH_0               | 通道0, TIMERx (x=0,1,7,19,20)  |
| TIMER_CH_1               | 通道1, TIMERx (x=0,1,7,19,20)  |
| TIMER_CH_2               | 通道2, TIMERx (x=0,1,7,19,20)  |
| TIMER_CH_3               | 通道3, TIMERx (x=0,1,7,19,20)  |
| 输入参数{in}                 |  |
| aocshadow                | 通道附加输出比较影子寄存器状态  |
| TIMER_ADD_SHADOW_ENABLE  | 通道附加输出比较影子寄存器使能  |
| TIMER_ADD_SHADOW_DISABLE | 通道附加输出比较影子寄存器禁能  |
| 输出参数{out}                |  |
| -                        | -  |
| 返回值                      |  |
| -                        | -  |

例如：

```
/*configure TIMER0 channel 0 additional output shadow function */
```

```
timer_channel_additional_output_shadow_config (TIMER0,          TIMER_CH_0,
TIMER_OC_SHADOW_ENABLE);
```

### 函数 timer\_break\_external\_input\_struct\_para\_init

函数timer\_break\_external\_input\_struct\_para\_init描述见下表：

表 3-733. 函数 timer\_break\_external\_input\_struct\_para\_init

|             |   |
|-------------|---|
| 函数名称        | timer_break_external_input_struct_para_init   |
| 函数原型        | void timer_break_external_input_struct_para_init(timer_break_ext_input_struct *breakinpara) |
| 功能描述        | 将TIMER中止功能外部输入参数结构体中所有参数初始化为默认值   |
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| breakinpara | 中止功能外部输入参数结构体，详见 <a href="#">结构体timer_break_ext_input_struct</a> 。                          |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如：

```
/* initialize TIMER break external input parameter struct with a default value */
```

```
timer_break_ext_input_struct breakinpara;
```

```
timer_break_external_input_struct_para_init (&breakinpara);
```

### 函数 timer\_break\_external\_input\_config

函数timer\_break\_external\_input\_config描述见下表：

表 3-734. 函数 timer\_break\_external\_input\_config

|                     |   |
|---------------------|---|
| 函数名称                | timer_break_external_input_config   |
| 函数原型                | void timer_break_external_input_config(uint32_t timer_periph, uint32_t break_input, timer_break_ext_input_struct *breakinpara); |
| 功能描述                | 配置TIMER中止功能外部输入   |
| 先决条件                | -   |
| 被调用函数               | -   |
| 输入参数{in}            |   |
| timer_periph        | TIMER外设   |
| TIMERx(x=0,7,19,20) | 参考具体参数  |

| 输入参数{in}                                |  |
|---|--|
| <b>break_input</b>                      | 中止功能外部输入   |
| <i>TIMER_BREAKINP</i><br><i>UT_BRK0</i> | TIMER中止外部输入0   |
| <i>TIMER_BREAKINP</i><br><i>UT_BRK1</i> | TIMER中止外部输入1   |
| <i>TIMER_BREAKINP</i><br><i>UT_BRK2</i> | TIMER中止外部输入2   |
| <i>TIMER_BREAKINP</i><br><i>UT_BRK3</i> | TIMER中止外部输入3   |
| 输入参数{in}                                |  |
| <b>breakinpara</b>                      | 中止功能外部输入参数结构体，详见 <a href="#">结构体timer_break_ext_input_struct</a> 。 |
| 输出参数{out}                               |  |
| -                                       | -  |
| 返回值                                     |  |
| -                                       | -  |

例如：

```

/* configure TIMER0 break external input */

timer_break_ext_input_struct timer_breakinpara;

timer_breakinpara.filter    = 15;

timer_breakinpara.enable = ENABLE;

timer_breakinpara.polarity = TIMER_BRKIN_POLARITY_HIGH;

timer_break_external_input_config(TIMER0,          TIMER_BREAKINPUT_BRK0,          &
timer_breakinpara);

```

### 函数 timer\_break\_external\_input\_enable

函数timer\_break\_external\_input\_enable描述见下表：

表 3-735. 函数 timer\_break\_external\_input\_enable

| 函数名称                       | timer_break_external_input_enable  |
|----------------------------|--|
| 函数原型                       | void timer_break_external_input_enable(uint32_t timer_periph, uint32_t break_input); |
| 功能描述                       | 中止功能外部输入使能   |
| 先决条件                       | -  |
| 被调用函数                      | -  |
| 输入参数{in}                   |  |
| <b>timer_periph</b>        | TIMER外设  |
| <i>TIMERx(x=0,7,19,20)</i> | 参考具体参数   |



| 输入参数{in}                                |              |
|---|--------------|
| <b>break_input</b>                      | 中止功能外部输入     |
| <i>TIMER_BREAKINP</i><br><i>UT_BRK0</i> | TIMER中止外部输入0 |
| <i>TIMER_BREAKINP</i><br><i>UT_BRK1</i> | TIMER中止外部输入1 |
| <i>TIMER_BREAKINP</i><br><i>UT_BRK2</i> | TIMER中止外部输入2 |
| <i>TIMER_BREAKINP</i><br><i>UT_BRK3</i> | TIMER中止外部输入3 |
| 输出参数{out}                               |              |
| -                                       | -            |
| 返回值                                     |              |
| -                                       | -            |

例如：

```
/* enable TIMER0 break external input */
```

```
timer_break_external_input_enable(TIMER0, TIMER_BREAKINPUT_BRK0);
```

### 函数 timer\_break\_external\_input\_disable

函数timer\_break\_external\_input\_disable描述见下表：

**表 3-736. 函数 timer\_break\_external\_input\_disable**

| <b>函数名称</b>                             | timer_break_external_input_disable  |
|---|---|
| <b>函数原型</b>                             | void timer_break_external_input_disable(uint32_t timer_periph, uint32_t break_input); |
| <b>功能描述</b>                             | 中止功能外部输入禁能  |
| <b>先决条件</b>                             | -   |
| <b>被调用函数</b>                            | -   |
| 输入参数{in}                                |   |
| <b>timer_periph</b>                     | TIMER外设   |
| <i>TIMERx(x=0,7,19,20)</i>              | 参考具体参数  |
| 输入参数{in}                                |   |
| <b>break_input</b>                      | 中止功能外部输入  |
| <i>TIMER_BREAKINP</i><br><i>UT_BRK0</i> | TIMER中止外部输入0  |
| <i>TIMER_BREAKINP</i><br><i>UT_BRK1</i> | TIMER中止外部输入1  |
| <i>TIMER_BREAKINP</i><br><i>UT_BRK2</i> | TIMER中止外部输入2  |
| <i>TIMER_BREAKINP</i>                   | TIMER中止外部输入3  |

|                |   |
|----------------|---|
| <i>UT_BRK3</i> |   |
| 输出参数{out}      |   |
| -              | - |
| 返回值            |   |
| -              | - |

例如:

```
/* disable TIMER0 break external input */
```

```
timer_break_external_input_disable(TIMER0, TIMER_BREAKINPUT_BRK0);
```

### 函数 timer\_break\_external\_input\_polarity\_config

函数timer\_break\_external\_input\_polarity\_config描述见下表:

表 3-737. 函数 timer\_break\_external\_input\_polarity\_config

|                               |  |
|-------------------------------|--|
| 函数名称                          | timer_break_external_input_polarity_config   |
| 函数原型                          | void timer_break_external_input_polarity_config(uint32_t timer_periph, uint32_t break_input, uint32_t polarity); |
| 功能描述                          | 配置TIMER中止功能外部输入极性  |
| 先决条件                          | -  |
| 被调用函数                         | -  |
| 输入参数{in}                      |  |
| timer_periph                  | TIMER外设  |
| TIMERx(x=0,7,19,20)           | 参考具体参数   |
| 输入参数{in}                      |  |
| break_input                   | 中止功能外部输入   |
| TIMER_BREAKINP<br>UT_BRK0     | TIMER中止外部输入0   |
| TIMER_BREAKINP<br>UT_BRK1     | TIMER中止外部输入1   |
| TIMER_BREAKINP<br>UT_BRK2     | TIMER中止外部输入2   |
| TIMER_BREAKINP<br>UT_BRK3     | TIMER中止外部输入3   |
| 输入参数{in}                      |  |
| polarity                      | 中止功能外部输入极性   |
| TIMER_BRKIN_PO<br>LARITY_HIGH | 中止功能外部输入极性为高电平   |
| TIMER_BRKIN_PO<br>LARITY_LOW  | 中止功能外部输入极性为低电平   |
| 输出参数{out}                     |  |
| -                             | -  |

| 返回值 |   |
|-----|---|
| -   | - |

例如：

```
/* configure TIMER0 break external input 0 polarity */
```

```
timer_break_external_input_polarity_config (TIMER0,    TIMER_BREAKINPUT_BRK0,
TIMER_BRKIN_POLARITY_HIGH);
```

### 函数 timer\_channel\_break\_control\_config

函数timer\_channel\_break\_control\_config描述见下表：

**表 3-738. 函数 timer\_channel\_break\_control\_config**

| 函数名称                  | timer_channel_break_control_config  |
|-----------------------|---|
| 函数原型                  | void timer_channel_break_control_config(uint32_t timer_periph, uint32_t channel, ControlStatus newvalue); |
| 功能描述                  | 配置TIMER通道的中止功能  |
| 先决条件                  | -   |
| 被调用函数                 | -   |
| 输入参数{in}              |   |
| <b>timer_periph</b>   | TIMER外设   |
| TIMERx(x=0,1,7,19,20) | 参考具体参数  |
| 输入参数{in}              |   |
| <b>channel</b>        | TIMER通道   |
| TIMER_CH_0            | 通道0   |
| TIMER_CH_1            | 通道1   |
| TIMER_CH_2            | 通道2   |
| TIMER_CH_3            | 通道3   |
| 输入参数{in}              |   |
| <b>newvalue</b>       | 控制状态  |
| ENABLE                | 使能  |
| DISABLE               | 禁能  |
| 输出参数{out}             |   |
| -                     | -   |
| 返回值                   |   |
| -                     | -   |

例如：

```
/* configure TIMER0 channel 0 break function */
```

```
timer_channel_break_control_config (TIMER0, TIMER_CH_0, ENABLE);
```

## 函数 timer\_channel\_dead\_time\_config

函数timer\_channel\_dead\_time\_config描述见下表:

表 3-739. 函数 timer\_channel\_dead\_time\_config

|                       |   |
|-----------------------|---|
| 函数名称                  | timer_channel_dead_time_config  |
| 函数原型                  | void timer_channel_dead_time_config(uint32_t timer_periph, uint32_t channel, ControlStatus newvalue); |
| 功能描述                  | 配置TIMER通道的死区功能  |
| 先决条件                  | -   |
| 被调用函数                 | -   |
| 输入参数{in}              |   |
| timer_periph          | TIMER外设   |
| TIMERx(x=0,1,7,19,20) | 参考具体参数  |
| 输入参数{in}              |   |
| channel               | TIMER通道   |
| TIMER_CH_0            | 通道0   |
| TIMER_CH_1            | 通道1   |
| TIMER_CH_2            | 通道2   |
| TIMER_CH_3            | 通道3   |
| 输入参数{in}              |   |
| newvalue              | 控制状态  |
| ENABLE                | 使能  |
| DISABLE               | 禁能  |
| 输出参数{out}             |   |
| -                     | -   |
| 返回值                   |   |
| -                     | -   |

例如:

```
/* configure TIMER0 channel 0 free dead time function */
```

```
timer_channel_dead_time_config (TIMER0, TIMER_CH_0, ENABLE);
```

## 函数 timer\_free\_complementary\_struct\_para\_init

函数timer\_free\_complementary\_struct\_para\_init描述见下表:

表 3-740. 函数 timer\_free\_complementary\_struct\_para\_init

|      |  |
|------|--|
| 函数名称 | timer_free_complementary_struct_para_init  |
| 函数原型 | void<br>timer_free_complementary_struct_para_init(timer_free_complementary_parameter_struct *freecompara); |
| 功能描述 | 将TIMER通道独立互补参数结构体中所有参数初始化为默认值  |

|             |   |
|-------------|---|
| 先决条件        | -   |
| 被调用函数       | -   |
| 输入参数{in}    |   |
| freecompara | 独立互补参数结构体，详见 <a href="#">结构体</a><br><a href="#">timer_free_complementary_parameter_struct</a> . |
| 输出参数{out}   |   |
| -           | -   |
| 返回值         |   |
| -           | -   |

例如：

```
/* initialize TIMER channel free complementary parameter struct with a default value */
```

```
timer_free_complementary_parameter_struct timer_freecompara;
```

```
timer_free_complementary_struct_para_init (&timer_freecompara);
```

### 函数 timer\_channel\_free\_complementary\_config

函数timer\_channel\_free\_complementary\_config描述见下表：

表 3-741. 函数 timer\_channel\_free\_complementary\_config

|                     |   |
|---------------------|---|
| 函数名称                | timer_channel_free_complementary_config   |
| 函数原型                | void timer_channel_free_complementary_config(uint32_t timer_periph, uint16_t channel, timer_free_complementary_parameter_struct *fcpa); |
| 功能描述                | 配置TIMER独立互补通道保护功能   |
| 先决条件                | -   |
| 被调用函数               | -   |
| 输入参数{in}            |   |
| timer_periph        | TIMER外设   |
| TIMERx(x=0,7,19,20) | 参考具体参数  |
| 输入参数{in}            |   |
| channel             | TIMER通道   |
| TIMER_CH_0          | 通道0   |
| TIMER_CH_1          | 通道1   |
| TIMER_CH_2          | 通道2   |
| TIMER_CH_3          | 通道3   |
| 输入参数{in}            |   |
| freecompara         | 独立互补参数结构体，详见 <a href="#">结构体</a><br><a href="#">timer_free_complementary_parameter_struct</a> .   |
| 输出参数{out}           |   |
| -                   | -   |
| 返回值                 |   |

例如：

```
/* initialize TIMER break parameter struct with a default value */
timer_free_complementary_parameter_struct timer_freecompara;

timer_freecompara.runoffstate = TIMER_FCCHP_STATE_ENABLE;

timer_freecompara.ideloffstate = TIMER_ROS_STATE_ENABLE;

timer_freecompara.deadtime = 255;

timer_freecompara.breakpolarity = TIMER_IOS_STATE_ENABLE;

timer_channel_free_complementary_config(&timer_freecompara);
```

### 函数 timer\_flag\_get

函数timer\_flag\_get描述见下表：

表 3-742. 函数 timer\_flag\_get

| 函数名称                          | timer_flag_get   |
|-------------------------------|--|
| 函数原型                          | FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag); |
| 功能描述                          | 获取外设TIMERx的状态标志  |
| 先决条件                          | -  |
| 被调用函数                         | -  |
| 输入参数{in}                      |  |
| timer_periph                  | TIMER外设  |
| TIMERx<br>(x=0,1,5,6,7,19,20) | 参考具体参数   |
| 输入参数{in}                      |  |
| flag                          | 状态标志   |
| TIMER_FLAG_UP                 | 更新标志, TIMERx(x=0,1,5,6,7,19,20)                                  |
| TIMER_FLAG_CH0                | 通道0比较/捕获标志, TIMERx(x=0,1,7,19,20)                                |
| TIMER_FLAG_CH1                | 通道1比较/捕获标志, TIMERx(x=0,1,7,19,20)                                |
| TIMER_FLAG_CH2                | 通道2比较/捕获标志, TIMERx(x=0,1,7,19,20)                                |
| TIMER_FLAG_CH3                | 通道3比较/捕获标志, TIMERx(x=0,1,7,19,20)                                |
| TIMER_FLAG_CMT                | 通道换相更新标志, TIMERx(x=0,7,19,20)                                    |
| TIMER_FLAG_TRG                | 触发标志, TIMERx(x=0,1,7,19,20)                                      |
| TIMER_FLAG_BRK                | 中止标志位, TIMERx(x=0,7,19,20)                                       |
| TIMER_FLAG_CH0<br>O           | 通道0捕获溢出标志, TIMERx(x=0,1,7,19,20)                                 |
| TIMER_FLAG_CH1<br>O           | 通道1捕获溢出标志, TIMERx(x=0,1,7,19,20)                                 |
| TIMER_FLAG_CH2<br>O           | 通道2捕获溢出标志, TIMERx(x=0,1,7,19,20)                                 |

|  |   |
|--|---|
| <code>TIMER_FLAG_CH3</code><br><code>0</code>      | 通道3捕获溢出标志, <code>TIMERx(x=0,1,7,19,20)</code>   |
| <code>TIMER_FLAG_MCH</code><br><code>0</code>      | 多模式通道0比较/捕获标志, <code>TIMERx(x=0,7,19,20)</code> |
| <code>TIMER_FLAG_MCH</code><br><code>1</code>      | 多模式通道1比较/捕获标志, <code>TIMERx(x=0,7,19,20)</code> |
| <code>TIMER_FLAG_MCH</code><br><code>2</code>      | 多模式通道2比较/捕获标志, <code>TIMERx(x=0,7,19,20)</code> |
| <code>TIMER_FLAG_MCH</code><br><code>3</code>      | 多模式通道3比较/捕获标志, <code>TIMERx(x=0,7,19,20)</code> |
| <code>TIMER_FLAG_MCH</code><br><code>00</code>     | 多模式通道0捕获溢出标志, <code>TIMERx(x=0,7,19,20)</code>  |
| <code>TIMER_FLAG_MCH</code><br><code>10</code>     | 多模式通道1捕获溢出标志, <code>TIMERx(x=0,7,19,20)</code>  |
| <code>TIMER_FLAG_MCH</code><br><code>20</code>     | 多模式通道2捕获溢出标志, <code>TIMERx(x=0,7,19,20)</code>  |
| <code>TIMER_FLAG_MCH</code><br><code>30</code>     | 多模式通道3捕获溢出标志, <code>TIMERx(x=0,7,19,20)</code>  |
| <code>TIMER_FLAG_CH0</code><br><code>COMADD</code> | 通道0附加比较标志, <code>TIMERx(x=0,7,19,20)</code>     |
| <code>TIMER_FLAG_CH1</code><br><code>COMADD</code> | 通道1附加比较标志, <code>TIMERx(x=0,7,19,20)</code>     |
| <code>TIMER_FLAG_CH2</code><br><code>COMADD</code> | 通道2附加比较标志, <code>TIMERx(x=0,7,19,20)</code>     |
| <code>TIMER_FLAG_CH3</code><br><code>COMADD</code> | 通道3附加比较标志, <code>TIMERx(x=0,7,19,20)</code>     |
| 输出参数{out}  |   |
| -  | -   |
| 返回值  |   |
| FlagStatus   | SET或者RESET                                      |

例如:

```
/* get TIMER0 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get(TIMER0, TIMER_FLAG_UP);
```

### 函数 timer\_flag\_clear

函数timer\_flag\_clear描述见下表:

表 3-743. 函数 timer\_flag\_clear

|      |  |
|------|--|
| 函数名称 | timer_flag_clear   |
| 函数原型 | void timer_flag_clear(uint32_t timer_periph, uint32_t flag); |

|                          |                                    |
|--------------------------|------------------------------------|
| 功能描述                     | 清除外设TIMERx状态标志                     |
| 先决条件                     | -                                  |
| 被调用函数                    | -                                  |
| 输入参数{in}                 |                                    |
| timer_periph             | TIMER外设                            |
| TIMERx                   | 参考具体参数                             |
| 输入参数{in}                 |                                    |
| flag                     | 状态标志                               |
| TIMER_FLAG_UP            | 更新标志, TIMERx(x=0,1,5,6,7,19,20)    |
| TIMER_FLAG_CH0           | 通道0比较/捕获标志, TIMERx(x=0,1,7,19,20)  |
| TIMER_FLAG_CH1           | 通道1比较/捕获标志, TIMERx(x=0,1,7,19,20)  |
| TIMER_FLAG_CH2           | 通道2比较/捕获标志, TIMERx(x=0,1,7,19,20)  |
| TIMER_FLAG_CH3           | 通道3比较/捕获标志, TIMERx(x=0,1,7,19,20)  |
| TIMER_FLAG_CMT           | 通道换相更新标志, TIMERx(x=0,7,19,20)      |
| TIMER_FLAG_TRG           | 触发标志, TIMERx(x=0,1,7,19,20)        |
| TIMER_FLAG_BRK           | 中止标志位, TIMERx(x=0,7,19,20)         |
| TIMER_FLAG_CH0<br>0      | 通道0捕获溢出标志, TIMERx(x=0,1,7,19,20)   |
| TIMER_FLAG_CH1<br>0      | 通道1捕获溢出标志, TIMERx(x=0,1,7,19,20)   |
| TIMER_FLAG_CH2<br>0      | 通道2捕获溢出标志, TIMERx(x=0,1,7,19,20)   |
| TIMER_FLAG_CH3<br>0      | 通道3捕获溢出标志, TIMERx(x=0,1,7,19,20)   |
| TIMER_FLAG_MCH<br>0      | 多模式通道0比较/捕获标志, TIMERx(x=0,7,19,20) |
| TIMER_FLAG_MCH<br>1      | 多模式通道1比较/捕获标志, TIMERx(x=0,7,19,20) |
| TIMER_FLAG_MCH<br>2      | 多模式通道2比较/捕获标志, TIMERx(x=0,7,19,20) |
| TIMER_FLAG_MCH<br>3      | 多模式通道3比较/捕获标志, TIMERx(x=0,7,19,20) |
| TIMER_FLAG_MCH<br>00     | 多模式通道0捕获溢出标志, TIMERx(x=0,7,19,20)  |
| TIMER_FLAG_MCH<br>10     | 多模式通道1捕获溢出标志, TIMERx(x=0,7,19,20)  |
| TIMER_FLAG_MCH<br>20     | 多模式通道2捕获溢出标志, TIMERx(x=0,7,19,20)  |
| TIMER_FLAG_MCH<br>30     | 多模式通道3捕获溢出标志, TIMERx(x=0,7,19,20)  |
| TIMER_FLAG_CH0<br>COMADD | 通道0附加比较标志, TIMERx(x=0,7,19,20)     |



|                                 |                                |
|---------------------------------|--------------------------------|
| <i>TIMER_FLAG_CH1</i><br>COMADD | 通道1附加比较标志, TIMERx(x=0,7,19,20) |
| <i>TIMER_FLAG_CH2</i><br>COMADD | 通道2附加比较标志, TIMERx(x=0,7,19,20) |
| <i>TIMER_FLAG_CH3</i><br>COMADD | 通道3附加比较标志, TIMERx(x=0,7,19,20) |
| 输出参数{out}                       |                                |
| -                               | -                              |
| 返回值                             |                                |
| -                               | -                              |

例如:

```
/* clear TIMER0 update flags */
```

```
timer_flag_clear(TIMER0, TIMER_FLAG_UP);
```

### 函数 timer\_interrupt\_enable

函数timer\_interrupt\_enable描述见下表:

表 3-744. 函数 timer\_interrupt\_enable

|                |   |
|----------------|---|
| 函数名称           | timer_interrupt_enable  |
| 函数原型           | void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt); |
| 功能描述           | 外设TIMERx中断使能  |
| 先决条件           | -   |
| 被调用函数          | -   |
| 输入参数{in}       |   |
| timer_periph   | TIMER外设   |
| TIMERx         | 参考具体参数  |
| 输入参数{in}       |   |
| interrupt      | 中断源   |
| TIMER_INT_UP   | 更新中断, TIMERx(x=0,1,5,6,7,19,20)   |
| TIMER_INT_CH0  | 通道0比较/捕获中断, TIMERx(x=0,1,7,19,20)                                       |
| TIMER_INT_CH1  | 通道1比较/捕获中断, TIMERx(x=0,1,7,19,20)                                       |
| TIMER_INT_CH2  | 通道2比较/捕获中断, TIMERx(x=0,1,7,19,20)                                       |
| TIMER_INT_CH3  | 通道3比较/捕获中断, TIMERx(x=0,1,7,19,20)                                       |
| TIMER_INT_CMT  | 换相更新中断, TIMERx(x=0,7,19,20)   |
| TIMER_INT_TRG  | 触发中断, TIMERx(x=0,1,7,19,20)   |
| TIMER_INT_BRK  | 中止中断, TIMERx(x=0,7,19,20)   |
| TIMER_INT_MCH0 | 多模式通道0比较/捕获中断, TIMERx(x=0,7,19,20)                                      |
| TIMER_INT_MCH1 | 多模式通道1比较/捕获中断, TIMERx(x=0,7,19,20)                                      |
| TIMER_INT_MCH2 | 多模式通道2比较/捕获中断, TIMERx(x=0,7,19,20)                                      |
| TIMER_INT_MCH3 | 多模式通道3比较/捕获中断, TIMERx(x=0,7,19,20)                                      |
| TIMER_INT_CH0C | 通道0附加比较中断, TIMERx(x=0,7,19,20)  |

|                         |                                |
|-------------------------|--------------------------------|
| OMADD                   |                                |
| TIMER_INT_CH0C<br>OMADD | 通道1附加比较中断, TIMERx(x=0,7,19,20) |
| TIMER_INT_CH0C<br>OMADD | 通道2附加比较中断, TIMERx(x=0,7,19,20) |
| TIMER_INT_CH0C<br>OMADD | 通道3附加比较中断, TIMERx(x=0,7,19,20) |
| 输出参数{out}               |                                |
| -                       | -                              |
| 返回值                     |                                |
| -                       | -                              |

例如:

```
/* enable the TIMER0 update interrupt */
```

```
timer_interrupt_enable (TIMER0, TIMER_INT_UP);
```

### 函数 timer\_interrupt\_disable

函数timer\_interrupt\_disable描述见下表:

表 3-745. 函数 timer\_interrupt\_disable

|                |   |
|----------------|---|
| 函数名称           | timer_interrupt_disable   |
| 函数原型           | void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt); |
| 功能描述           | 外设TIMERx中断禁能  |
| 先决条件           | -   |
| 被调用函数          | -   |
| 输入参数{in}       |   |
| timer_periph   | TIMER外设   |
| TIMERx         | 参考具体参数  |
| 输入参数{in}       |   |
| interrupt      | 中断源   |
| TIMER_INT_UP   | 更新中断, TIMERx(x=0,1,5,6,7,19,20)   |
| TIMER_INT_CH0  | 通道0比较/捕获中断, TIMERx(x=0,1,7,19,20)   |
| TIMER_INT_CH1  | 通道1比较/捕获中断, TIMERx(x=0,1,7,19,20)   |
| TIMER_INT_CH2  | 通道2比较/捕获中断, TIMERx(x=0,1,7,19,20)   |
| TIMER_INT_CH3  | 通道3比较/捕获中断, TIMERx(x=0,1,7,19,20)   |
| TIMER_INT_CMT  | 换相更新中断, TIMERx(x=0,7,19,20)   |
| TIMER_INT_TRG  | 触发中断, TIMERx(x=0,1,7,19,20)   |
| TIMER_INT_BRK  | 中止中断, TIMERx(x=0,7,19,20)   |
| TIMER_INT_MCH0 | 多模式通道0比较/捕获中断, TIMERx(x=0,7,19,20)  |
| TIMER_INT_MCH1 | 多模式通道1比较/捕获中断, TIMERx(x=0,7,19,20)  |
| TIMER_INT_MCH2 | 多模式通道2比较/捕获中断, TIMERx(x=0,7,19,20)  |
| TIMER_INT_MCH3 | 多模式通道3比较/捕获中断, TIMERx(x=0,7,19,20)  |

|                                       |   |
|---------------------------------------|---|
| <i>TIMER_INT_CH0C</i><br><i>OMADD</i> | 通道0附加比较中断, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_INT_CH0C</i><br><i>OMADD</i> | 通道1附加比较中断, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_INT_CH0C</i><br><i>OMADD</i> | 通道2附加比较中断, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_INT_CH0C</i><br><i>OMADD</i> | 通道3附加比较中断, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| 输出参数{out}                             |   |
| -                                     | -   |
| 返回值                                   |   |
| -                                     | -   |

例如:

```
/* disable the TIMER0 update interrupt */
```

```
timer_interrupt_disable(TIMER0, TIMER_INT_UP);
```

### 函数 timer\_interrupt\_flag\_get

函数timer\_interrupt\_flag\_get描述见下表:

表 3-746. 函数 timer\_interrupt\_flag\_get

|                           |  |
|---------------------------|--|
| 函数名称                      | timer_interrupt_flag_get   |
| 函数原型                      | FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t int_flag); |
| 功能描述                      | 获取外设 <i>TIMERx</i> 中断标志  |
| 先决条件                      | -  |
| 被调用函数                     | -  |
| 输入参数{in}                  |  |
| timer_periph              | <i>TIMER</i> 外设  |
| <i>TIMERx</i>             | 参考具体参数   |
| 输入参数{in}                  |  |
| int_flag                  | 中断源  |
| <i>TIMER_INT_FLAG_UP</i>  | 更新中断标志, <i>TIMERx</i> ( <i>x</i> =0,1,5,6,7,19,20)                             |
| <i>TIMER_INT_FLAG_CH0</i> | 通道0比较/捕获中断标志, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)                           |
| <i>TIMER_INT_FLAG_CH1</i> | 通道1比较/捕获中断标志, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)                           |
| <i>TIMER_INT_FLAG_CH2</i> | 通道2比较/捕获中断标志, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)                           |
| <i>TIMER_INT_FLAG_CH3</i> | 通道3比较/捕获中断标志, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)                           |
| <i>TIMER_INT_FLAG_</i>    | 换相更新中断标志, <i>TIMERx</i> ( <i>x</i> =0,7,19,20)                                 |

|                                 |   |
|---------------------------------|---|
| <i>CMT</i>                      |   |
| <i>TIMER_INT_FLAG_TRG</i>       | 触发中断标志, <i>TIMERx</i> ( <i>x</i> =0,1,7,19,20)        |
| <i>TIMER_INT_FLAG_BRK</i>       | 中止中断标志, <i>TIMERx</i> ( <i>x</i> =0,7,19,20)          |
| <i>TIMER_INT_FLAG_MCH0</i>      | 多模式通道0比较/捕获中断标志, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_INT_FLAG_MCH1</i>      | 多模式通道1比较/捕获中断标志, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_INT_FLAG_MCH2</i>      | 多模式通道2比较/捕获中断标志, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_INT_FLAG_MCH3</i>      | 多模式通道3比较/捕获中断标志, <i>TIMERx</i> ( <i>x</i> =0,7,19,20) |
| <i>TIMER_INT_FLAG_CH0COMADD</i> | 通道0附加比较中断标志, <i>TIMERx</i> ( <i>x</i> =0,7,19,20)     |
| <i>TIMER_INT_FLAG_CH1COMADD</i> | 通道1附加比较中断标志, <i>TIMERx</i> ( <i>x</i> =0,7,19,20)     |
| <i>TIMER_INT_FLAG_CH2COMADD</i> | 通道2附加比较中断标志, <i>TIMERx</i> ( <i>x</i> =0,7,19,20)     |
| <i>TIMER_INT_FLAG_CH3COMADD</i> | 通道3附加比较中断标志, <i>TIMERx</i> ( <i>x</i> =0,7,19,20)     |
| 输出参数{out}                       |   |
| -                               | -   |
| 返回值                             |   |
| <b>FlagStatus</b>               | SET或者RESET  |

例如:

```
/* get TIMER0 update interrupt flag */
```

```
FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = timer_interrupt_flag_get(TIMER0, TIMER_INT_FLAG_UP);
```

### 函数 timer\_interrupt\_flag\_clear

函数timer\_interrupt\_flag\_clear描述见下表:

表 3-747. 函数 timer\_interrupt\_flag\_clear

|                     |  |
|---------------------|--|
| 函数名称                | timer_interrupt_flag_clear   |
| 函数原型                | void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t int_flag); |
| 功能描述                | 清除外设 <i>TIMERx</i> 的中断标志   |
| 先决条件                | -  |
| 被调用函数               | -  |
| 输入参数{in}            |  |
| <b>timer_periph</b> | <i>TIMER</i> 外设  |

| TIMERx                   | 参考具体参数                               |
|--------------------------|--------------------------------------|
| 输入参数{in}                 |                                      |
| int_flag                 | 中断源                                  |
| TIMER_INT_FLAG_UP        | 更新中断标志, TIMERx(x=0,1,5,6,7,19,20)    |
| TIMER_INT_FLAG_CH0       | 通道0比较/捕获中断标志, TIMERx(x=0,1,7,19,20)  |
| TIMER_INT_FLAG_CH1       | 通道1比较/捕获中断标志, TIMERx(x=0,1,7,19,20)  |
| TIMER_INT_FLAG_CH2       | 通道2比较/捕获中断标志, TIMERx(x=0,1,7,19,20)  |
| TIMER_INT_FLAG_CH3       | 通道3比较/捕获中断标志, TIMERx(x=0,1,7,19,20)  |
| TIMER_INT_FLAG_CMT       | 换相更新中断标志, TIMERx(x=0,7,19,20)        |
| TIMER_INT_FLAG_TRG       | 触发中断标志, TIMERx(x=0,1,7,19,20)        |
| TIMER_INT_FLAG_BRK       | 中止中断标志, TIMERx(x=0,7,19,20)          |
| TIMER_INT_FLAG_MCH0      | 多模式通道0比较/捕获中断标志, TIMERx(x=0,7,19,20) |
| TIMER_INT_FLAG_MCH1      | 多模式通道1比较/捕获中断标志, TIMERx(x=0,7,19,20) |
| TIMER_INT_FLAG_MCH2      | 多模式通道2比较/捕获中断标志, TIMERx(x=0,7,19,20) |
| TIMER_INT_FLAG_MCH3      | 多模式通道3比较/捕获中断标志, TIMERx(x=0,7,19,20) |
| TIMER_INT_FLAG_CH0COMADD | 通道0附加比较中断标志, TIMERx(x=0,7,19,20)     |
| TIMER_INT_FLAG_CH1COMADD | 通道1附加比较中断标志, TIMERx(x=0,7,19,20)     |
| TIMER_INT_FLAG_CH2COMADD | 通道2附加比较中断标志, TIMERx(x=0,7,19,20)     |
| TIMER_INT_FLAG_CH3COMADD | 通道3附加比较中断标志, TIMERx(x=0,7,19,20)     |
| 输出参数{out}                |                                      |
| -                        | -                                    |
| 返回值                      |                                      |
| -                        | -                                    |

例如:

```
/* clear TIMER0 update interrupt flag */
```

```
timer_interrupt_flag_clear(TIMER0, TIMER_INT_FLAG_UP);
```

### 3.23. TRGSEL

TRGSEL 是 MCU 中的触发选择控制器。可通过软件配置的方式，为各种外设选择触发输入信号。章节 [3.23.1](#) 描述了 TRGSEL 的寄存器列表，章节 [3.23.2](#) 对 TRGSEL 库函数进行说明。

#### 3.23.1. 外设寄存器说明

TRGSEL 寄存器列表如下表所示：

**表 3-748. TRGSEL 寄存器**

| 寄存器名称               | 寄存器描述                        |
|---------------------|------------------------------|
| TRGSEL_EXTOUT0      | EXTOUT0 TRGSEL 触发选择寄存器       |
| TRGSEL_EXTOUT1      | EXTOUT1 TRGSEL 触发选择寄存器       |
| TRGSEL_ADC0         | ADC0 TRGSEL 触发选择寄存器          |
| TRGSEL_ADC1         | ADC1 TRGSEL 触发选择寄存器          |
| TRGSEL_DAC          | DAC TRGSEL 触发选择寄存器           |
| TRGSEL_TIMER0IN     | TIMER0_ITI TRGSEL 触发选择寄存器    |
| TRGSEL_TIMER0BRKIN  | TIMER0_BRKIN TRGSEL 触发选择寄存器  |
| TRGSEL_TIMER7IN     | TIMER7_ITI TRGSEL 触发选择寄存器    |
| TRGSEL_TIMER7BRKIN  | TIMER7_BRKIN TRGSEL 触发选择寄存器  |
| TRGSEL_TIMER19IN    | TIMER19_ITI TRGSEL 触发选择寄存器   |
| TRGSEL_TIMER19BRKIN | TIMER19_BRKIN TRGSEL 触发选择寄存器 |
| TRGSEL_TIMER20IN    | TIMER20_ITI TRGSEL 触发选择寄存器   |
| TRGSEL_TIMER20BRKIN | TIMER20_BRKIN TRGSEL 触发选择寄存器 |
| TRGSEL_TIMER1IN     | TIMER1_ITI TRGSEL 触发选择寄存器    |
| TRGSEL_MFCOM        | MFCOM TRGSEL 触发选择寄存器         |
| TRGSEL_CAN0         | CAN0 TRGSEL 触发选择寄存器          |
| TRGSEL_CAN1         | CAN1 TRGSEL 触发选择寄存器          |

#### 3.23.2. 外设库函数说明

TRGSEL 库函数列表如下表所示：

**表 3-749. TRGSEL 库函数**

| 函数名称                       | 功能描述        |
|----------------------------|-------------|
| trigsel_init               | 为外设选择触发输入源  |
| trigsel_trigger_source_get | 获取外设的触发输入源  |
| trigsel_register_lock_set  | 锁定触发寄存器     |
| trigsel_register_lock_get  | 获取触发寄存器锁定状态 |

## 枚举类型 `trigsel_source_enum`

表 3-750. 枚举类型 `trigsel_source_enum`

| 成员名称                       | 功能描述                 |
|----------------------------|----------------------|
| TRIGSEL_INPUT_0            | 触发输入源为0              |
| TRIGSEL_INPUT_1            | 触发输入源为1              |
| TRIGSEL_INPUT_TRIGSEL_IN0  | 触发输入源为TRIGSEL_IN0引脚  |
| TRIGSEL_INPUT_TRIGSEL_IN1  | 触发输入源为TRIGSEL_IN1引脚  |
| TRIGSEL_INPUT_TRIGSEL_IN2  | 触发输入源为TRIGSEL_IN2引脚  |
| TRIGSEL_INPUT_TRIGSEL_IN3  | 触发输入源为TRIGSEL_IN3引脚  |
| TRIGSEL_INPUT_TRIGSEL_IN4  | 触发输入源为TRIGSEL_IN4引脚  |
| TRIGSEL_INPUT_TRIGSEL_IN5  | 触发输入源为TRIGSEL_IN5引脚  |
| TRIGSEL_INPUT_TRIGSEL_IN6  | 触发输入源为TRIGSEL_IN6引脚  |
| TRIGSEL_INPUT_TRIGSEL_IN7  | 触发输入源为TRIGSEL_IN7引脚  |
| TRIGSEL_INPUT_TRIGSEL_IN8  | 触发输入源为TRIGSEL_IN8引脚  |
| TRIGSEL_INPUT_TRIGSEL_IN9  | 触发输入源为TRIGSEL_IN9引脚  |
| TRIGSEL_INPUT_TRIGSEL_IN10 | 触发输入源为TRIGSEL_IN10引脚 |
| TRIGSEL_INPUT_TRIGSEL_IN11 | 触发输入源为TRIGSEL_IN11引脚 |
| TRIGSEL_INPUT_CMP_OUT      | 触发输入源为CMP_OUT        |
| TRIGSEL_INPUT_LXTAL_TRG    | 触发输入源为LXTAL_TRG      |
| TRIGSEL_INPUT_TIMER1_CH0   | 触发输入源为timer1通道0      |
| TRIGSEL_INPUT_TIMER1_CH1   | 触发输入源为timer1通道1      |
| TRIGSEL_INPUT_TIMER1_CH2   | 触发输入源为timer1通道2      |
| TRIGSEL_INPUT_TIMER1_CH3   | 触发输入源为timer1通道3      |
| TRIGSEL_INPUT_TIMER1_TRGO  | 触发输入源为timer1 TRGO    |
| TRIGSEL_INPUT_TIMER0_CH0   | 触发输入源为timer0通道0      |
| TRIGSEL_INPUT_TIMER0_CH1   | 触发输入源为timer0通道1      |
| TRIGSEL_INPUT_TIMER0_CH2   | 触发输入源为timer0通道2      |
| TRIGSEL_INPUT_TIMER0_CH3   | 触发输入源为timer0通道3      |
| TRIGSEL_INPUT_TIMER0_MCH0  | 触发输入源为timer0多模式通道0   |
| TRIGSEL_INPUT_TIMER0_MCH1  | 触发输入源为timer0多模式通道1   |
| TRIGSEL_INPUT_TIMER0_MCH2  | 触发输入源为timer0多模式通道2   |
| TRIGSEL_INPUT_TIMER0_MCH3  | 触发输入源为timer0多模式通道3   |
| TRIGSEL_INPUT_TIMER0_TRGO  | 触发输入源为timer0 TRGO    |
| TRIGSEL_INPUT_TIMER7_CH0   | 触发输入源为timer7通道0      |
| TRIGSEL_INPUT_TIMER7_CH1   | 触发输入源为timer7通道1      |
| TRIGSEL_INPUT_TIMER7_CH2   | 触发输入源为timer7通道2      |
| TRIGSEL_INPUT_TIMER7_CH3   | 触发输入源为timer7通道3      |
| TRIGSEL_INPUT_TIMER7_MCH0  | 触发输入源为timer7多模式通道0   |
| TRIGSEL_INPUT_TIMER7_MCH1  | 触发输入源为timer7多模式通道1   |
| TRIGSEL_INPUT_TIMER7_MCH2  | 触发输入源为timer7多模式通道2   |
| TRIGSEL_INPUT_TIMER7_MCH3  | 触发输入源为timer7多模式通道3   |

| 成员名称                       | 功能描述                   |
|----------------------------|------------------------|
| TRIGSEL_INPUT_TIMER7_TRGO  | 触发输入源为timer7 TRGO      |
| TRIGSEL_INPUT_TIMER19_CH0  | 触发输入源为timer19通道0       |
| TRIGSEL_INPUT_TIMER19_CH1  | 触发输入源为timer19通道1       |
| TRIGSEL_INPUT_TIMER19_CH2  | 触发输入源为timer19通道2       |
| TRIGSEL_INPUT_TIMER19_CH3  | 触发输入源为timer19通道3       |
| TRIGSEL_INPUT_TIMER19_MCH0 | 触发输入源为timer19多模式通道0    |
| TRIGSEL_INPUT_TIMER19_MCH1 | 触发输入源为timer19多模式通道1    |
| TRIGSEL_INPUT_TIMER19_MCH2 | 触发输入源为timer19多模式通道2    |
| TRIGSEL_INPUT_TIMER19_MCH3 | 触发输入源为timer19多模式通道3    |
| TRIGSEL_INPUT_TIMER19_TRGO | 触发输入源为timer19 TRGO     |
| TRIGSEL_INPUT_TIMER20_CH0  | 触发输入源为timer20通道0       |
| TRIGSEL_INPUT_TIMER20_CH1  | 触发输入源为timer20通道1       |
| TRIGSEL_INPUT_TIMER20_CH2  | 触发输入源为timer20通道2       |
| TRIGSEL_INPUT_TIMER20_CH3  | 触发输入源为timer20通道3       |
| TRIGSEL_INPUT_TIMER20_MCH0 | 触发输入源为timer20多模式通道0    |
| TRIGSEL_INPUT_TIMER20_MCH1 | 触发输入源为timer20多模式通道1    |
| TRIGSEL_INPUT_TIMER20_MCH2 | 触发输入源为timer20多模式通道2    |
| TRIGSEL_INPUT_TIMER20_MCH3 | 触发输入源为timer20多模式通道3    |
| TRIGSEL_INPUT_TIMER20_TRGO | 触发输入源为timer20 TRGO     |
| TRIGSEL_INPUT_TIMER5_TRGO  | 触发输入源为timer5 TRGO      |
| TRIGSEL_INPUT_TIMER6_TRGO  | 触发输入源为timer6 TRGO      |
| TRIGSEL_INPUT_MFCOM_TRIG0  | 触发输入源为MFCOM TRIG0      |
| TRIGSEL_INPUT_MFCOM_TRIG1  | 触发输入源为MFCOM TRIG1      |
| TRIGSEL_INPUT_MFCOM_TRIG2  | 触发输入源为MFCOM TRIG2      |
| TRIGSEL_INPUT_MFCOM_TRIG3  | 触发输入源为MFCOM TRIG3      |
| TRIGSEL_INPUT_RTC_ALARM    | 触发输入源为RTC alarm        |
| TRIGSEL_INPUT_RTC_SECOND   | 触发输入源为RTC second       |
| TRIGSEL_INPUT_TRIGSEL_IN12 | 触发输入源为TRIGSEL_IN12 pin |
| TRIGSEL_INPUT_TRIGSEL_IN13 | 触发输入源为TRIGSEL_IN13 pin |

### 枚举类型 `trigsel_periph_enum`

表 3-751. 枚举类型 `trigsel_periph_enum`

| 成员名称                        | 功能描述                  |
|-----------------------------|-----------------------|
| TRIGSEL_OUTPUT_TRIGSEL_OUT0 | 输出到目标外设TRIGSEL_OUT0引脚 |
| TRIGSEL_OUTPUT_TRIGSEL_OUT1 | 输出到目标外设TRIGSEL_OUT1引脚 |
| TRIGSEL_OUTPUT_TRIGSEL_OUT2 | 输出到目标外设TRIGSEL_OUT2引脚 |
| TRIGSEL_OUTPUT_TRIGSEL_OUT3 | 输出到目标外设TRIGSEL_OUT3引脚 |
| TRIGSEL_OUTPUT_TRIGSEL_OUT4 | 输出到目标外设TRIGSEL_OUT4引脚 |
| TRIGSEL_OUTPUT_TRIGSEL_OUT5 | 输出到目标外设TRIGSEL_OUT5引脚 |
| TRIGSEL_OUTPUT_TRIGSEL_OUT6 | 输出到目标外设TRIGSEL_OUT6引脚 |
| TRIGSEL_OUTPUT_TRIGSEL_OUT7 | 输出到目标外设TRIGSEL_OUT7引脚 |



| 成员名称                          | 功能描述                  |
|-------------------------------|-----------------------|
| TRIGSEL_OUTPUT_ADC0_RTTRG     | 输出到目标外设ADC0_RTTRG     |
| TRIGSEL_OUTPUT_ADC0_INSTRG    | 输出到目标外设ADC0_INSTRG    |
| TRIGSEL_OUTPUT_ADC1_RTTRG     | 输出到目标外设ADC1_RTTRG     |
| TRIGSEL_OUTPUT_ADC1_INSTRG    | 输出到目标外设ADC1_INSTRG    |
| TRIGSEL_OUTPUT_DAC_EXTRIG     | 输出到目标外设DAC_EXTRIG     |
| TRIGSEL_OUTPUT_TIMER0_ITI0    | 输出到目标外设TIMER0_ITI0    |
| TRIGSEL_OUTPUT_TIMER0_ITI1    | 输出到目标外设TIMER0_ITI1    |
| TRIGSEL_OUTPUT_TIMER0_ITI2    | 输出到目标外设TIMER0_ITI2    |
| TRIGSEL_OUTPUT_TIMER0_ITI3    | 输出到目标外设TIMER0_ITI3    |
| TRIGSEL_OUTPUT_TIMER0_BRKIN0  | 输出到目标外设TIMER0_BRKIN0  |
| TRIGSEL_OUTPUT_TIMER0_BRKIN1  | 输出到目标外设TIMER0_BRKIN1  |
| TRIGSEL_OUTPUT_TIMER0_BRKIN2  | 输出到目标外设TIMER0_BRKIN2  |
| TRIGSEL_OUTPUT_TIMER0_BRKIN3  | 输出到目标外设TIMER0_BRKIN3  |
| TRIGSEL_OUTPUT_TIMER7_ITI0    | 输出到目标外设TIMER7_ITI0    |
| TRIGSEL_OUTPUT_TIMER7_ITI1    | 输出到目标外设TIMER7_ITI1    |
| TRIGSEL_OUTPUT_TIMER7_ITI2    | 输出到目标外设TIMER7_ITI2    |
| TRIGSEL_OUTPUT_TIMER7_ITI3    | 输出到目标外设TIMER7_ITI3    |
| TRIGSEL_OUTPUT_TIMER7_BRKIN0  | 输出到目标外设TIMER7_BRKIN0  |
| TRIGSEL_OUTPUT_TIMER7_BRKIN1  | 输出到目标外设TIMER7_BRKIN1  |
| TRIGSEL_OUTPUT_TIMER7_BRKIN2  | 输出到目标外设TIMER7_BRKIN2  |
| TRIGSEL_OUTPUT_TIMER7_BRKIN3  | 输出到目标外设TIMER7_BRKIN3  |
| TRIGSEL_OUTPUT_TIMER19_ITI0   | 输出到目标外设TIMER19_ITI0   |
| TRIGSEL_OUTPUT_TIMER19_ITI1   | 输出到目标外设TIMER19_ITI1   |
| TRIGSEL_OUTPUT_TIMER19_ITI2   | 输出到目标外设TIMER19_ITI2   |
| TRIGSEL_OUTPUT_TIMER19_ITI3   | 输出到目标外设TIMER19_ITI3   |
| TRIGSEL_OUTPUT_TIMER19_BRKIN0 | 输出到目标外设TIMER19_BRKIN0 |
| TRIGSEL_OUTPUT_TIMER19_BRKIN1 | 输出到目标外设TIMER19_BRKIN1 |
| TRIGSEL_OUTPUT_TIMER19_BRKIN2 | 输出到目标外设TIMER19_BRKIN2 |
| TRIGSEL_OUTPUT_TIMER19_BRKIN3 | 输出到目标外设TIMER19_BRKIN3 |
| TRIGSEL_OUTPUT_TIMER20_ITI0   | 输出到目标外设TIMER20_ITI0   |
| TRIGSEL_OUTPUT_TIMER20_ITI1   | 输出到目标外设TIMER20_ITI1   |
| TRIGSEL_OUTPUT_TIMER20_ITI2   | 输出到目标外设TIMER20_ITI2   |
| TRIGSEL_OUTPUT_TIMER20_ITI3   | 输出到目标外设TIMER20_ITI3   |
| TRIGSEL_OUTPUT_TIMER20_BRKIN0 | 输出到目标外设TIMER20_BRKIN0 |
| TRIGSEL_OUTPUT_TIMER20_BRKIN1 | 输出到目标外设TIMER20_BRKIN1 |
| TRIGSEL_OUTPUT_TIMER20_BRKIN2 | 输出到目标外设TIMER20_BRKIN2 |
| TRIGSEL_OUTPUT_TIMER20_BRKIN3 | 输出到目标外设TIMER20_BRKIN3 |
| TRIGSEL_OUTPUT_TIMER1_ITI0    | 输出到目标外设TIMER1_ITI0    |
| TRIGSEL_OUTPUT_TIMER1_ITI1    | 输出到目标外设TIMER1_ITI1    |
| TRIGSEL_OUTPUT_TIMER1_ITI2    | 输出到目标外设TIMER1_ITI2    |
| TRIGSEL_OUTPUT_TIMER1_ITI3    | 输出到目标外设TIMER1_ITI3    |

| 成员名称                                 | 功能描述                     |
|--------------------------------------|--------------------------|
| TRIGSEL_OUTPUT_MFCOM_TRG_TIME<br>R0  | 输出到目标外设MFCOM_TRG_TIMER0  |
| TRIGSEL_OUTPUT_MFCOM_TRG_TIME<br>R1  | 输出到目标外设MFCOM_TRG_TIMER1  |
| TRIGSEL_OUTPUT_MFCOM_TRG_TIME<br>R2  | 输出到目标外设MFCOM_TRG_TIMER2  |
| TRIGSEL_OUTPUT_MFCOM_TRG_TIME<br>R3  | 输出到目标外设MFCOM_TRG_TIMER3  |
| TRIGSEL_OUTPUT_CAN0_EX_TIME_TIC<br>K | 输出到目标外设CAN0_EX_TIME_TICK |
| TRIGSEL_OUTPUT_CAN1_EX_TIME_TIC<br>K | 输出到目标外设CAN1_EX_TIME_TICK |

### 函数 trigsel\_init

函数trigsel\_init描述见下表:

表 3-752. 函数 trigsel\_init

|                |   |
|----------------|---|
| 函数名称           | trigsel_init  |
| 函数原型           | void trigsel_init(trigsel_periph_enum target_periph, trigsel_source_enum trigger_source); |
| 功能描述           | 为外设选择触发输入源  |
| 先决条件           | -   |
| 被调用函数          | -   |
| 输入参数{in}       |   |
| target_periph  | 目标外设, 参考 <a href="#">表 3-751. 枚举类型 trigsel_periph_enum</a>                                |
| 输入参数{in}       |   |
| trigger_source | 触发源, 参考 <a href="#">表 3-750. 枚举类型 trigsel_source_enum</a>                                 |
| 输出参数{out}      |   |
| -              | -   |
| 返回值            |   |
| -              | -   |

例如:

```
/* select TIMER0_CH2 to trigger ADC0 */
trigsel_init(TRIGSEL_OUTPUT_ADC0_RTTRG, TRIGSEL_INPUT_TIMER0_CH2);
```

### 函数 trigsel\_trigger\_source\_get

函数trigsel\_trigger\_source\_get描述见下表:

表 3-753. 函数 trigsel\_trigger\_source\_get

|      |                            |
|------|----------------------------|
| 函数名称 | trigsel_trigger_source_get |
|------|----------------------------|

|                |  |
|----------------|--|
| 函数原型           | uint8_t trigsels_trigger_source_get(trigsels_periph_enum target_periph); |
| 功能描述           | 获取外设的触发输入源   |
| 先决条件           | -  |
| 被调用函数          | -  |
| 输入参数{in}       |  |
| target_periph  | 目标外设，参考 <a href="#">表 3-751. 枚举类型 trigsels_periph_enum</a>               |
| 输出参数{out}      |  |
| -              | -  |
| 返回值            |  |
| trigger_source | 触发源，值范围应为 0-67   |

例如：

```
/* get the trigger input signal for ADC0 */
```

```
uint8_t input_signal;
```

```
input_signal = trigsels_trigger_source_get(TRIGSELS_OUTPUT_ADC0_RTTRG);
```

### 函数 trigsels\_register\_lock\_set

函数trigsels\_register\_lock\_set描述见下表：

**表 3-754. 函数 trigsels\_trigger\_source\_set**

|               |  |
|---------------|--|
| 函数名称          | trigsels_register_lock_set   |
| 函数原型          | void trigsels_register_lock_set(trigsels_periph_enum target_periph); |
| 功能描述          | 锁定触发寄存器  |
| 先决条件          | -  |
| 被调用函数         | -  |
| 输入参数{in}      |  |
| target_periph | 目标外设，参考 <a href="#">表 3-751. 枚举类型 trigsels_periph_enum</a>           |
| 输出参数{out}     |  |
| -             | -  |
| 返回值           |  |
| -             | -  |

例如：

```
/* lock the trigger register for ADC0 */
```

```
trigsels_register_lock_set(TRIGSELS_OUTPUT_ADC0_RTTRG);
```

### 函数 trigsels\_register\_lock\_get

函数trigsels\_register\_lock\_get描述见下表：

**表 3-755. 函数 trigsels\_trigger\_lock\_get**

|      |                            |
|------|----------------------------|
| 函数名称 | trigsels_register_lock_get |
|------|----------------------------|

|               |  |
|---------------|--|
| 函数原型          | FlagStatus trigsels_register_lock_get(trigsels_periph_enum target_periph); |
| 功能描述          | 获取触发寄存器锁定状态  |
| 先决条件          | -  |
| 被调用函数         | -  |
| 输入参数{in}      |  |
| target_periph | 目标外设，参考 <a href="#">表 3-751. 枚举类型 trigsels_periph_enum</a>                 |
| 输出参数{out}     |  |
| -             | -  |
| 返回值           |  |
| FlagStatus    | SET 或 RESET  |

例如：

```
/* get the trigger register lock status of ADC0 */
```

```
FlagStatus status;
```

```
status = trigsels_register_lock_get(TRIGSELS_OUTPUT_ADC0_RTTRG);
```

## 3.24. USART

通用同步异步收发器（USART）提供了一个灵活方便的串行数据交换接口，章节[3.24.1](#)描述了USART的寄存器列表，章节[3.24.2](#)对USART库函数进行说明。

### 3.24.1. 外设寄存器说明

USART寄存器列表如下表所示：

表 3-756. USART 寄存器

| 寄存器名称       | 寄存器描述          |
|-------------|----------------|
| USART_CTL0  | 控制寄存器0         |
| USART_CTL1  | 控制寄存器1         |
| USART_CTL2  | 控制寄存器2         |
| USART_BAUD  | 波特率寄存器         |
| USART_GP    | 保护时间和预分频器寄存器   |
| USART_RT    | 接收超时寄存器        |
| USART_CMD   | 请求寄存器          |
| USART_STAT  | 状态寄存器          |
| USART_INTC  | 中断标志清除寄存器      |
| USART_RDATA | 数据接收寄存器        |
| USART_TDATA | 数据发送寄存器        |
| USART_CHC   | 兼容性控制寄存器       |
| USART_RFCS  | 接收FIFO控制和状态寄存器 |

### 3.24.2. 外设库函数说明

USART库函数列表如下表所示：

表 3-757. USART 库函数

| 库函数名称                   | 库函数描述            |
|-------------------------|------------------|
| usart_deinit            | 复位外设USART        |
| usart_baudrate_set      | 配置USART波特率       |
| usart_parity_config     | 配置USART奇偶校验      |
| usart_word_length_set   | 配置USART字长        |
| usart_stop_bit_set      | 配置USART停止位       |
| usart_enable            | 使能USART          |
| usart_disable           | 失能USART          |
| usart_transmit_config   | USART发送配置        |
| usart_receive_config    | USART接收配置        |
| usart_data_first_config | 配置数据传输时低位在前或高位在前 |
| usart_invert_config     | 配置USART反转功能      |
| usart_overrun_enable    | 使能USART溢出禁止功能    |

| 库函数名称                                   | 库函数描述                       |
|---|-----------------------------|
| usart_overrun_disable                   | 失能USART溢出禁止功能               |
| usart_oversample_config                 | 配置USART过采样模式                |
| usart_sample_bit_config                 | 配置USART单次采样方式               |
| usart_receiver_timeout_enable           | 使能USART接收超时                 |
| usart_receiver_timeout_disable          | 失能USART接收超时                 |
| usart_receiver_timeout_threshold_config | 设置USART接收超时阈值               |
| usart_data_transmit                     | USART发送数据功能                 |
| usart_data_receive                      | USART接收数据功能                 |
| usart_command_enable                    | 使能USART请求                   |
| usart_address_config                    | 配置USART地址                   |
| usart_address_detection_mode_config     | 配置USART地址检测模式               |
| usart_mute_mode_enable                  | 使能USART静默模式                 |
| usart_mute_mode_disable                 | 失能USART静默模式                 |
| usart_mute_mode_wakeup_config           | 配置USART静默模式唤醒方式             |
| usart_lin_mode_enable                   | 使能USART LIN模式               |
| usart_lin_mode_disable                  | 失能USART LIN模式               |
| usart_lin_break_detection_length_config | 配置USART LIN模式中斷帧长度          |
| usart_halfduplex_enable                 | 使能USART半双工模式                |
| usart_halfduplex_disable                | 失能USART半双工模式                |
| usart_clock_enable                      | 使能USART CK引脚                |
| usart_clock_disable                     | 失能USART CK引脚                |
| usart_synchronous_clock_config          | 配置USART同步通讯模式参数             |
| usart_guard_time_config                 | 在USART智能卡模式下配置保护时间值         |
| usart_smartcard_mode_enable             | 使能USART智能卡模式                |
| usart_smartcard_mode_disable            | 失能USART智能卡模式                |
| usart_smartcard_mode_nack_enable        | 在USART智能卡模式下使能NACK          |
| usart_smartcard_mode_nack_disable       | 在USART智能卡模式下失能NACK          |
| usart_smartcard_mode_early_nack_enable  | 使能USART智能卡模式提前NACK          |
| usart_smartcard_mode_early_nack_disable | 失能USART智能卡模式提前NACK          |
| usart_smartcard_autoretry_config        | 配置智能卡自动重试次数                 |
| usart_block_length_config               | 配置智能卡T=1的接收时块的长度            |
| usart_irda_mode_enable                  | 使能USART串行红外编解码功能模块          |
| usart_irda_mode_disable                 | 失能USART串行红外编解码功能模块          |
| usart_prescaler_config                  | 在USART IrDA低功耗模式下配置外设时钟分频系数 |
| usart_irda_lowpower_config              | 配置USART IrDA低功耗模式           |
| usart_hardware_flow_rts_config          | 配置USART RTS硬件控制流            |

| 库函数名称                                | 库函数描述              |
|--------------------------------------|--------------------|
| usart_hardware_flow_cts_config       | 配置USART CTS硬件控制流   |
| usart_hardware_flow_coherence_config | 配置硬件流控兼容模式         |
| usart_rs485_driver_enable            | 使能USART rs485驱动    |
| usart_rs485_driver_disable           | 失能USART rs485驱动    |
| usart_driver_asserttime_config       | 配置USART驱动使能置位时间    |
| usart_driver_deasserttime_config     | 配置USART驱动使能置低时间    |
| usart_depolarity_config              | 配置USART驱动使能极性模式    |
| usart_dma_receive_config             | 配置USART DMA接收      |
| usart_dma_transmit_config            | 配置USART DMA发送      |
| usart_reception_error_dma_disable    | USART接收错误时失能DMA    |
| usart_reception_error_dma_enable     | USART接收错误时使能DMA    |
| usart_wakeup_enable                  | 使能USART唤醒          |
| usart_wakeup_disable                 | 失能USART唤醒          |
| usart_wakeup_mode_config             | 配置USART唤醒模式        |
| usart_receive_fifo_enable            | 使能接收FIFO           |
| usart_receive_fifo_disable           | 失能接收FIFO           |
| usart_receive_fifo_counter_number    | 读取接收FIFO计数器的值      |
| usart_flag_get                       | 得到STAT/RFCR寄存器中的标志 |
| usart_flag_clear                     | 清除USART状态          |
| usart_interrupt_enable               | 使能USART中断          |
| usart_interrupt_disable              | 失能USART中断          |
| usart_interrupt_flag_get             | 得到USART中断和标志状态     |
| usart_interrupt_flag_clear           | 清除USART中断标志位       |

### 枚举类型 usart\_flag\_enum

表 3-758. 枚举类型 usart\_flag\_enum

| 成员名称            | 功能描述        |
|-----------------|-------------|
| USART_FLAG_REA  | 接收使能通知标志    |
| USART_FLAG_TEA  | 发送使能通知标志    |
| USART_FLAG_WU   | 从深度睡眠模式唤醒标志 |
| USART_FLAG_RWU  | 接收器从静默模式唤醒  |
| USART_FLAG_SB   | 断开信号发送标志    |
| USART_FLAG_AM   | 地址匹配标志      |
| USART_FLAG_BSY  | 忙标志         |
| USART_FLAG_EB   | 块结束标志       |
| USART_FLAG_RT   | 接收超时标志      |
| USART_FLAG_CTS  | CTS电平       |
| USART_FLAG_CTSF | CTS变化标志     |
| USART_FLAG_LBD  | LIN断开检测标志   |
| USART_FLAG_TBE  | 发送数据寄存器空    |

| 成员名称              | 功能描述        |
|-------------------|-------------|
| USART_FLAG_TC     | 发送完成        |
| USART_FLAG_RBNE   | 读数据缓冲区非空    |
| USART_FLAG_IDLE   | 空闲线检测标志     |
| USART_FLAG_ORERR  | 溢出错误        |
| USART_FLAG_NERR   | 噪声错误标志      |
| USART_FLAG_FERR   | 帧错误         |
| USART_FLAG_PERR   | 校验错误        |
| USART_FLAG_EPERR  | 校验错误超前检测标志  |
| USART_FLAG_RFFINT | 接收FIFO满中断标志 |
| USART_FLAG_RFF    | 接收FIFO满标志   |
| USART_FLAG_RFE    | 接收FIFO空标志   |

### 枚举类型 `usart_interrupt_flag_enum`

表 3-759. 枚举类型 `usart_interrupt_flag_enum`

| 成员名称                      | 功能描述          |
|---------------------------|---------------|
| USART_INT_FLAG_EB         | 块结束中断标志       |
| USART_INT_FLAG_RT         | 接收超时中断标志      |
| USART_INT_FLAG_AM         | 地址匹配中断标志      |
| USART_INT_FLAG_PERR       | 奇偶校验错误中断标志    |
| USART_INT_FLAG_TBE        | 发送寄存器空中断标志    |
| USART_INT_FLAG_TC         | 发送完成中断标志      |
| USART_INT_FLAG_RBNE       | 读缓冲区非空中断标志    |
| USART_INT_FLAG_RBNE_ORERR | 读缓冲区非空和溢出中断标志 |
| USART_INT_FLAG_IDLE       | 空闲线检测中断标志     |
| USART_INT_FLAG_LBD        | LIN断开检测中断标志   |
| USART_INT_FLAG_WU         | 从深度睡眠模式唤醒中断标志 |
| USART_INT_FLAG_CTS        | CTS中断标志       |
| USART_INT_FLAG_ERR_NERR   | 噪声错误中断标志      |
| USART_INT_FLAG_ERR_ORERR  | 溢出错误中断标志      |
| USART_INT_FLAG_ERR_FERR   | 帧错误中断标志       |
| USART_INT_FLAG_RFF        | 接收FIFO满中断标志   |

### 枚举类型 `usart_interrupt_enum`

表 3-760. 枚举类型 `usart_interrupt_enum`

| 成员名称         | 功能描述     |
|--------------|----------|
| USART_INT_EB | 块结束中断使能  |
| USART_INT_RT | 接收超时中断使能 |
| USART_INT_AM | 地址匹配中断使能 |



| 成员名称           | 功能描述              |
|----------------|-------------------|
| USART_INT_PERR | 奇偶校验错误中断使能        |
| USART_INT_TBE  | 发送寄存器空中断使能        |
| USART_INT_TC   | 发送完成中断使能          |
| USART_INT_RBNE | 读缓冲区非空中断和溢出错误中断使能 |
| USART_INT_IDLE | 空闲线检测中断使能         |
| USART_INT_LBD  | LIN断开检测中断使能       |
| USART_INT_WU   | 从深度睡眠模式唤醒中断使能     |
| USART_INT_CTS  | CTS中断使能           |
| USART_INT_ERR  | 错误中断使能            |
| USART_INT_RFF  | 接收FIFO满中断使能       |

### 枚举类型 `usart_invert_enum`

表 3-761. 枚举类型 `usart_invert_enum`

| 成员名称                | 功能描述       |
|---------------------|------------|
| USART_DINV_ENABLE   | 数据位反转      |
| USART_DINV_DISABLE  | 数据位不反转     |
| USART_TXPIN_ENABLE  | TX管脚电平反转   |
| USART_TXPIN_DISABLE | TX管脚电平不反转  |
| USART_RXPIN_ENABLE  | RX管脚电平反转   |
| USART_RXPIN_DISABLE | RX管脚电平不反转  |
| USART_SWAP_ENABLE   | 交换TX/RX管脚  |
| USART_SWAP_DISABLE  | 不交换TX/RX管脚 |

### 函数 `usart_deinit`

函数`usart_deinit`描述见下表：

表 3-762. 函数 `usart_deinit`

| 函数名称                | <code>usart_deinit</code>  |
|---------------------|--|
| 函数原型                | <code>void usart_deinit(uint32_t usart_periph);</code>                       |
| 功能描述                | 复位外设USARTx   |
| 先决条件                | -  |
| 被调用函数               | <code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code> |
| 输入参数{in}            |  |
| <b>usart_periph</b> | 外设USARTx   |
| USARTx              | x=0,1,2  |
| 输出参数{out}           |  |
| -                   | -  |
| 返回值                 |  |
| -                   | -  |

例如：

```
/* reset USART0 */
```

```
usart_deinit(USART0);
```

### 函数 usart\_baudrate\_set

函数usart\_baudrate\_set描述见下表：

表 3-763. 函数 usart\_baudrate\_set

|              |   |
|--------------|---|
| 函数名称         | usart_baudrate_set  |
| 函数原型         | void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval); |
| 功能描述         | 配置USART波特率  |
| 先决条件         | -   |
| 被调用函数        | rcu_clock_freq_get  |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输入参数{in}     |   |
| baudval      | 波特率值  |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如：

```
/* configure USART0 baud rate value */
```

```
usart_baudrate_set(USART0, 115200);
```

### 函数 usart\_parity\_config

函数usart\_parity\_config描述见下表：

表 3-764. 函数 usart\_parity\_config

|               |  |
|---------------|--|
| 函数名称          | usart_parity_config  |
| 函数原型          | void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg); |
| 功能描述          | 配置USART奇偶校验  |
| 先决条件          | -  |
| 被调用函数         | -  |
| 输入参数{in}      |  |
| usart_periph  | 外设USARTx   |
| USARTx        | x=0,1,2  |
| 输入参数{in}      |  |
| paritycfg     | 配置USART奇偶校验  |
| USART_PM_NONE | 无校验  |

|               |     |
|---------------|-----|
| USART_PM_ODD  | 奇校验 |
| USART_PM_EVEN | 偶校验 |
| 输出参数{out}     |     |
| -             | -   |
| 返回值           |     |
| -             | -   |

例如：

```
/* configure USART0 parity */
```

```
usart_parity_config(USART0, USART_PM_EVEN);
```

### 函数 usart\_word\_length\_set

函数usart\_word\_length\_set描述见下表：

表 3-765. 函数 usart\_word\_length\_set

|               |   |
|---------------|---|
| 函数名称          | usart_word_length_set   |
| 函数原型          | void usart_word_length_set(uint32_t usart_periph, uint32_t wlen); |
| 功能描述          | 配置USART字长   |
| 先决条件          | -   |
| 被调用函数         | -   |
| 输入参数{in}      |   |
| usart_periph  | 外设USARTx  |
| USARTx        | x=0,1,2   |
| 输入参数{in}      |   |
| wlen          | 配置USART字长   |
| USART_WL_8BIT | 8位  |
| USART_WL_9BIT | 9位  |
| 输出参数{out}     |   |
| -             | -   |
| 返回值           |   |
| -             | -   |

例如：

```
/* configure USART0 word length */
```

```
usart_word_length_set(USART0, USART_WL_9BIT);
```

### 函数 usart\_stop\_bit\_set

函数usart\_stop\_bit\_set描述见下表：

表 3-766. 函数 usart\_stop\_bit\_set

|      |                    |
|------|--------------------|
| 函数名称 | usart_stop_bit_set |
|------|--------------------|

|                  |  |
|------------------|--|
| 函数原型             | void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen); |
| 功能描述             | 配置USART停止位   |
| 先决条件             | -  |
| 被调用函数            | -  |
| 输入参数{in}         |  |
| usart_periph     | 外设USARTx   |
| USARTx           | x=0,1,2  |
| 输入参数{in}         |  |
| stblen           | 配置USART停止位   |
| USART_STB_1BIT   | 1位   |
| USART_STB_0_5BIT | 0.5位   |
| USART_STB_2BIT   | 2位   |
| USART_STB_1_5BIT | 1.5位   |
| 输出参数{out}        |  |
| -                | -  |
| 返回值              |  |
| -                | -  |

例如:

```
/* configure USART0 stop bit length */
```

```
usart_stop_bit_set(USART0, USART_STB_1_5BIT);
```

### 函数 usart\_enable

函数usart\_enable描述见下表:

表 3-767. 函数 usart\_enable

|              |   |
|--------------|---|
| 函数名称         | usart_enable                              |
| 函数原型         | void usart_enable(uint32_t usart_periph); |
| 功能描述         | 使能USART                                   |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx                                  |
| USARTx       | x=0,1,2                                   |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如:

```
/* enable USART0 */
```

```
usart_enable(USART0);
```

### 函数 usart\_disable

函数usart\_disable描述见下表：

**表 3-768. 函数 usart\_disable**

|              |  |
|--------------|--|
| 函数名称         | usart_disable                              |
| 函数原型         | void usart_disable(uint32_t usart_periph); |
| 功能描述         | 失能USART                                    |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| usart_periph | 外设USARTx                                   |
| USARTx       | x=0,1,2                                    |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| -            | -  |

例如：

```
/* disable USART0 */
```

```
usart_disable(USART0);
```

### 函数 usart\_transmit\_config

函数usart\_transmit\_config描述见下表：

**表 3-769. 函数 usart\_transmit\_config**

|                        |   |
|------------------------|---|
| 函数名称                   | usart_transmit_config   |
| 函数原型                   | void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig); |
| 功能描述                   | USART发送器配置  |
| 先决条件                   | -   |
| 被调用函数                  | -   |
| 输入参数{in}               |   |
| usart_periph           | 外设USARTx  |
| USARTx                 | x=0,1,2   |
| 输入参数{in}               |   |
| txconfig               | 使能/失能USART发送器   |
| USART_TRANSMIT_ENABLE  | 使能USART发送   |
| USART_TRANSMIT_DISABLE | 失能USART发送   |

|                       |   |
|-----------------------|---|
| <code>_DISABLE</code> |   |
| 输出参数{out}             |   |
| -                     | - |
| 返回值                   |   |
| -                     | - |

例如:

```
/* configure USART0 transmitter */

usart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

### 函数 usart\_receive\_config

函数usart\_receive\_config描述见下表:

表 3-770. 函数 usart\_receive\_config

|                       |  |
|-----------------------|--|
| 函数名称                  | usart_receive_config   |
| 函数原型                  | void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig); |
| 功能描述                  | USART接收器配置   |
| 先决条件                  | -  |
| 被调用函数                 | -  |
| 输入参数{in}              |  |
| usart_periph          | 外设USARTx   |
| USARTx                | x=0,1,2  |
| 输入参数{in}              |  |
| rxconfig              | 使能/失能USART接收器  |
| USART_RECEIVE_ENABLE  | 使能USART接收  |
| USART_RECEIVE_DISABLE | 失能USART接收  |
| 输出参数{out}             |  |
| -                     | -  |
| 返回值                   |  |
| -                     | -  |

例如:

```
/* configure USART0 receiver */

usart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

### 函数 usart\_data\_first\_config

函数usart\_data\_first\_config描述见下表:

表 3-771. 函数 usart\_data\_first\_config

|                     |   |
|---------------------|---|
| 函数名称                | usart_data_first_config   |
| 函数原型                | void usart_data_first_config(uint32_t usart_periph, uint32_t msbf); |
| 功能描述                | 配置数据传输时低位在前或高位在前  |
| 先决条件                | -   |
| 被调用函数               | -   |
| 输入参数{in}            |   |
| usart_periph        | 外设USARTx  |
| USARTx              | x=0,1,2   |
| 输入参数{in}            |   |
| msbf                | 数据传输时低位在前/高位在前  |
| USART_MSBF_LSB<br>B | 数据传输时低位在前   |
| USART_MSBF_MS<br>B  | 数据传输时高位在前   |
| 输出参数{out}           |   |
| -                   | -   |
| 返回值                 |   |
| -                   | -   |

例如:

```
/* configure LSB of data first */
```

```
usart_data_first_config(USART0, USART_MSBF_LSB);
```

### 函数 usart\_invert\_config

函数usart\_invert\_config描述见下表:

表 3-772. 函数 usart\_invert\_config

|              |  |
|--------------|--|
| 函数名称         | usart_invert_config  |
| 函数原型         | void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara); |
| 功能描述         | 配置USART反转功能  |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| usart_periph | 外设USARTx   |
| USARTx       | x=0,1,2  |
| 输入参数{in}     |  |
| invertpara   | 参考 <a href="#">表3-761. 枚举类型usart_invert_enum</a>                               |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| -            | -  |

例如：

```
/* configure USART0 inversion */
```

```
usart_invert_config(USART0, USART_DINV_ENABLE);
```

### 函数 usart\_oversize\_enable

函数usart\_oversize\_enable描述见下表：

表 3-773. 函数 usart\_oversize\_enable

|              |  |
|--------------|--|
| 函数名称         | usart_oversize_enable                              |
| 函数原型         | void usart_oversize_enable(uint32_t usart_periph); |
| 功能描述         | 使能USART溢出禁止功能                                      |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| usart_periph | 外设USARTx   |
| USARTx       | x=0,1,2  |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| -            | -  |

例如：

```
/* enable USART0 overrun */
```

```
usart_oversize_enable(USART0);
```

### 函数 usart\_oversize\_disable

函数usart\_oversize\_disable描述见下表：

表 3-774. 函数 usart\_oversize\_disable

|              |   |
|--------------|---|
| 函数名称         | usart_oversize_disable                              |
| 函数原型         | void usart_oversize_disable(uint32_t usart_periph); |
| 功能描述         | 失能USART溢出禁止功能                                       |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |



例如:

```
/* disable USART0 overrun */
usart_oversample_disable(USART0);
```

### 函数 usart\_oversample\_config

函数usart\_oversample\_config描述见下表:

表 3-775. 函数 usart\_oversample\_config

|                 |   |
|-----------------|---|
| 函数名称            | usart_oversample_config   |
| 函数原型            | void usart_oversample_config(uint32_t usart_periph, uint32_t oversamp); |
| 功能描述            | 配置USART过采样模式  |
| 先决条件            | -   |
| 被调用函数           | -   |
| 输入参数{in}        |   |
| usart_periph    | 外设USARTx  |
| USARTx          | x=0,1,2   |
| 输入参数{in}        |   |
| oversamp        | 过采样值  |
| USART_OVSMOD_8  | 8倍过采样   |
| USART_OVSMOD_16 | 16倍过采样  |
| 输出参数{out}       |   |
| -               | -   |
| 返回值             |   |
| -               | -   |

例如:

```
/* config USART0 oversampling by 8 */
usart_oversample_config(USART0, USART_OVSMOD_8);
```

### 函数 usart\_sample\_bit\_config

函数usart\_sample\_bit\_config描述见下表:

表 3-776. 函数 usart\_sample\_bit\_config

|          |  |
|----------|--|
| 函数名称     | usart_sample_bit_config  |
| 函数原型     | void usart_sample_bit_config(uint32_t usart_periph, uint32_t osb); |
| 功能描述     | 配置USART单次采样方式  |
| 先决条件     | -  |
| 被调用函数    | -  |
| 输入参数{in} |  |

|                       |          |
|-----------------------|----------|
| <b>usart_periph</b>   | 外设USARTx |
| <i>USARTx</i>         | x=0,1,2  |
| <b>输入参数{in}</b>       |          |
| <b>osb</b>            | 单次采样方式   |
| <i>USART_OSB_1BIT</i> | 1次采样方法   |
| <i>USART_OSB_3BIT</i> | 3次采样方法   |
| <b>输出参数{out}</b>      |          |
| -                     | -        |
| <b>返回值</b>            |          |
| -                     | -        |

例如：

```
/* config USART0 1 bit sample mode */
```

```
usart_sample_bit_config(USART0, USART_OSB_1BIT);
```

### 函数 usart\_receiver\_timeout\_enable

函数usart\_receiver\_timeout\_enable描述见下表：

表 3-777. 函数 usart\_receiver\_timeout\_enable

|                     |  |
|---------------------|--|
| <b>函数名称</b>         | usart_receiver_timeout_enable                              |
| <b>函数原型</b>         | void usart_receiver_timeout_enable(uint32_t usart_periph); |
| <b>功能描述</b>         | 使能USART接收超时  |
| <b>先决条件</b>         | -  |
| <b>被调用函数</b>        | -  |
| <b>输入参数{in}</b>     |  |
| <b>usart_periph</b> | 外设USARTx   |
| <i>USARTx</i>       | x=0,1,2  |
| <b>输出参数{out}</b>    |  |
| -                   | -  |
| <b>返回值</b>          |  |
| -                   | -  |

例如：

```
/* enable USART0 receiver timeout */
```

```
usart_receiver_timeout_enable(USART0);
```

### 函数 usart\_receiver\_timeout\_disable

函数usart\_receiver\_timeout\_disable描述见下表：

表 3-778. 函数 usart\_receiver\_timeout\_disable

|             |                                |
|-------------|--------------------------------|
| <b>函数名称</b> | usart_receiver_timeout_disable |
|-------------|--------------------------------|

|              |   |
|--------------|---|
| 函数原型         | void usart_receiver_timeout_disable(uint32_t usart_periph); |
| 功能描述         | 失能USART接收超时   |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如：

```
/* disable USART0 receiver timeout */
usart_receiver_timeout_disable(USART0);
```

### 函数 usart\_receiver\_timeout\_threshold\_config

函数usart\_receiver\_timeout\_threshold\_config描述见下表：

表 3-779. 函数 usart\_receiver\_timeout\_threshold\_config

|              |   |
|--------------|---|
| 函数名称         | usart_receiver_timeout_threshold_config   |
| 函数原型         | void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout); |
| 功能描述         | 设置USART接收超时阈值   |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输入参数{in}     |   |
| rtimeout     | 超时时间（0x00000000-0x00FFFFFF）   |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如：

```
/* set the receiver timeout threshold of USART0*/
usart_receiver_timeout_threshold_config(USART0, 115200*3);
```

**函数 usart\_data\_transmit**

函数usart\_data\_transmit描述见下表：

**表 3-780. 函数 usart\_data\_transmit**

|              |   |
|--------------|---|
| 函数名称         | usart_data_transmit   |
| 函数原型         | void usart_data_transmit(uint32_t usart_periph, uint16_t data); |
| 功能描述         | USART发送数据功能   |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输入参数{in}     |   |
| data         | 发送的数据（0x0000-0x01FF）  |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如：

```
/* USART0 transmit data */
usart_data_transmit(USART0, 0x00AA);
```

**函数 usart\_data\_receive**

函数usart\_data\_receive描述见下表：

**表 3-781. 函数 usart\_data\_receive**

|              |   |
|--------------|---|
| 函数名称         | usart_data_receive                                  |
| 函数原型         | uint16_t usart_data_receive(uint32_t usart_periph); |
| 功能描述         | USART接收数据功能   |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| uint16_t     | 接收到的数据（0x0000-0x01FF）                               |

例如：

```
/* USART0 receive data */
```

```
uint16_t temp;
```

```
temp = usart_data_receive(USART0);
```

### 函数 usart\_command\_enable

函数usart\_command\_enable描述见下表：

**表 3-782. 函数 usart\_command\_enable**

|                      |   |
|----------------------|---|
| 函数名称                 | usart_command_enable  |
| 函数原型                 | void usart_command_enable(uint32_t usart_periph, uint32_t cmdtype); |
| 功能描述                 | 使能USART请求   |
| 先决条件                 | -   |
| 被调用函数                | -   |
| 输入参数{in}             |   |
| usart_periph         | 外设USARTx  |
| USARTx               | x=0,1,2   |
| 输入参数{in}             |   |
| cmdtype              | 请求类型  |
| USART_CMD_SBK<br>CMD | 发送断开帧请求   |
| USART_CMD_MM<br>CMD  | 静模式请求   |
| USART_CMD_RXF<br>CMD | 接收数据清空请求  |
| USART_CMD_TXF<br>CMD | 发送数据清空请求  |
| 输出参数{out}            |   |
| -                    | -   |
| 返回值                  |   |
| -                    | -   |

例如：

```
/* enable USART0 command */
```

```
usart_command_enable(USART0, USART_CMD_SBKCMD);
```

### 函数 usart\_address\_config

函数usart\_address\_config描述见下表：

**表 3-783. 函数 usart\_address\_config**

|      |   |
|------|---|
| 函数名称 | usart_address_config  |
| 函数原型 | void usart_address_config(uint32_t usart_periph, uint8_t addr); |
| 功能描述 | 配置USART地址   |

|                     |                    |
|---------------------|--------------------|
| 先决条件                | -                  |
| 被调用函数               | -                  |
| 输入参数{in}            |                    |
| <b>usart_periph</b> | 外设USARTx           |
| <i>USARTx</i>       | x=0,1,2            |
| 输入参数{in}            |                    |
| <b>addr</b>         | USART地址（0x00-0xFF） |
| 输出参数{out}           |                    |
| -                   | -                  |
| 返回值                 |                    |
| -                   | -                  |

例如：

```
/* configure address of the USART0 */
```

```
usart_address_config(USART0, 0x00);
```

### 函数 usart\_address\_detection\_mode\_config

函数usart\_address\_detection\_mode\_config描述见下表：

表 3-784. 函数 usart\_address\_detection\_mode\_config

|                           |   |
|---------------------------|---|
| 函数名称                      | usart_address_detection_mode_config   |
| 函数原型                      | void usart_address_detection_mode_config(uint32_t usart_periph, uint32_t addmod); |
| 功能描述                      | 配置USART地址检测模式   |
| 先决条件                      | -   |
| 被调用函数                     | -   |
| 输入参数{in}                  |   |
| <b>usart_periph</b>       | 外设USARTx  |
| <i>USARTx</i>             | x=0,1,2   |
| 输入参数{in}                  |   |
| <b>addmod</b>             | 地址检测模式  |
| <i>USART_ADDM_4BIT</i>    | 4位地址检测  |
| <i>USART_ADDM_FULLBIT</i> | 全位地址检测  |
| 输出参数{out}                 |   |
| -                         | -   |
| 返回值                       |   |
| -                         | -   |

例如：

```
/*configure address detection mode */
```

```
usart_address_config(USART0, USART_ADDDM_4BIT);
```

### 函数 usart\_mute\_mode\_enable

函数usart\_mute\_mode\_enable描述见下表：

**表 3-785. 函数 usart\_mute\_mode\_enable**

|              |   |
|--------------|---|
| 函数名称         | usart_mute_mode_enable                              |
| 函数原型         | void usart_mute_mode_enable(uint32_t usart_periph); |
| 功能描述         | 使能USART静默模式   |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如：

```
/* enable USART0 receiver in mute mode */
```

```
usart_mute_mode_enable(USART0);
```

### 函数 usart\_mute\_mode\_disable

函数usart\_mute\_mode\_disable描述见下表：

**表 3-786. 函数 usart\_mute\_mode\_disable**

|              |  |
|--------------|--|
| 函数名称         | usart_mute_mode_disable                              |
| 函数原型         | void usart_mute_mode_disable(uint32_t usart_periph); |
| 功能描述         | 失能USART静默模式  |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| usart_periph | 外设USARTx   |
| USARTx       | x=0,1,2  |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| -            | -  |

例如：

```
/* disable USART0 receiver in mute mode */
```

```
usart_mute_mode_disable(USART0);
```

### 函数 usart\_mute\_mode\_wakeup\_config

函数usart\_mute\_mode\_wakeup\_config描述见下表：

**表 3-787. 函数 usart\_mute\_mode\_wakeup\_config**

|               |  |
|---------------|--|
| 函数名称          | usart_mute_mode_wakeup_config  |
| 函数原型          | void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod); |
| 功能描述          | 配置USART静默模式唤醒方式  |
| 先决条件          | -  |
| 被调用函数         | -  |
| 输入参数{in}      |  |
| usart_periph  | 外设USARTx   |
| USARTx        | x=0,1,2  |
| 输入参数{in}      |  |
| wmethod       | 两种方法用于进入或退出静默模式  |
| USART_WM_IDLE | 空闲线唤醒  |
| USART_WM_ADDR | 地址匹配唤醒   |
| 输出参数{out}     |  |
| -             | -  |
| 返回值           |  |
| -             | -  |

例如：

```
/* configure USART0 wakeup method in mute mode */
```

```
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

### 函数 usart\_lin\_mode\_enable

函数usart\_lin\_mode\_enable描述见下表：

**表 3-788. 函数 usart\_lin\_mode\_enable**

|              |  |
|--------------|--|
| 函数名称         | usart_lin_mode_enable                              |
| 函数原型         | void usart_lin_mode_enable(uint32_t usart_periph); |
| 功能描述         | 使能USART LIN模式                                      |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| usart_periph | 外设USARTx   |
| USARTx       | x=0,1,2  |
| 输出参数{out}    |  |
| -            | -  |



| 返回值 |   |
|-----|---|
| -   | - |

例如：

```
/* USART0 LIN mode enable */
usart_lin_mode_enable(USART0);
```

### 函数 usart\_lin\_mode\_disable

函数usart\_lin\_mode\_disable描述见下表：

表 3-789. 函数 usart\_lin\_mode\_disable

|              |   |
|--------------|---|
| 函数名称         | usart_lin_mode_disable                              |
| 函数原型         | void usart_lin_mode_disable(uint32_t usart_periph); |
| 功能描述         | 失能USART LIN模式                                       |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如：

```
/* USART0 LIN mode disable */
usart_lin_mode_disable(USART0);
```

### 函数 usart\_lin\_break\_dection\_length\_config

函数usart\_lin\_break\_dection\_length\_config描述见下表：

表 3-790. 函数 usart\_lin\_break\_dection\_length\_config

|              |  |
|--------------|--|
| 函数名称         | usart_lin_break_dection_length_config  |
| 函数原型         | void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lflen); |
| 功能描述         | 配置USART LIN模式中断帧长度   |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| usart_periph | 外设USARTx   |
| USARTx       | x=0,1,2  |

| 输入参数{in}                          |            |
|-----------------------------------|------------|
| <b>lblen</b>                      | LIN模式中断帧长度 |
| <i>USART_LBLEN_10</i><br><i>B</i> | 断开帧长度为10位  |
| <i>USART_LBLEN_11</i><br><i>B</i> | 断开帧长度为11位  |
| 输出参数{out}                         |            |
| -                                 | -          |
| 返回值                               |            |
| -                                 | -          |

例如:

```
/* configure LIN break frame length */
```

```
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

### 函数 usart\_halfduplex\_enable

函数usart\_halfduplex\_enable描述见下表:

表 3-791. 函数 usart\_halfduplex\_enable

| <b>函数名称</b>         | usart_halfduplex_enable                              |
|---------------------|--|
| <b>函数原型</b>         | void usart_halfduplex_enable(uint32_t usart_periph); |
| <b>功能描述</b>         | 使能USART半双工模式   |
| <b>先决条件</b>         | -  |
| <b>被调用函数</b>        | -  |
| 输入参数{in}            |  |
| <b>usart_periph</b> | 外设USARTx   |
| <i>USARTx</i>       | x=0,1,2  |
| 输出参数{out}           |  |
| -                   | -  |
| 返回值                 |  |
| -                   | -  |

例如:

```
/* enable USART0 half duplex mode */
```

```
usart_halfduplex_enable(USART0);
```

### 函数 usart\_halfduplex\_disable

函数usart\_halfduplex\_disable描述见下表:

表 3-792. 函数 usart\_halfduplex\_disable

|             |                          |
|-------------|--------------------------|
| <b>函数名称</b> | usart_halfduplex_disable |
|-------------|--------------------------|

|              |   |
|--------------|---|
| 函数原型         | void usart_halfduplex_disable(uint32_t usart_periph); |
| 功能描述         | 失能USART半双工模式  |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如:

```
/* disable USART0 half duplex mode */
usart_halfduplex_disable(USART0);
```

### 函数 usart\_clock\_enable

函数usart\_clock\_enable描述见下表:

表 3-793. 函数 usart\_clock\_enable

|              |   |
|--------------|---|
| 函数名称         | usart_clock_enable                              |
| 函数原型         | void usart_clock_enable(uint32_t usart_periph); |
| 功能描述         | 使能USART CK引脚                                    |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如:

```
/* enable clock */
usart_clock_enable(USART0);
```

### 函数 usart\_clock\_disable

函数usart\_clock\_disable描述见下表:

表 3-794. 函数 usart\_clock\_disable

|              |  |
|--------------|--|
| 函数名称         | usart_clock_disable                              |
| 函数原型         | void usart_clock_disable(uint32_t usart_periph); |
| 功能描述         | 失能USART CK引脚                                     |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| usart_periph | 外设USARTx   |
| USARTx       | x=0,1,2  |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| -            | -  |

例如:

```
/* disable clock */
```

```
usart_clock_disable(USART0);
```

### 函数 usart\_synchronous\_clock\_config

函数usart\_synchronous\_clock\_config描述见下表:

表 3-795. 函数 usart\_synchronous\_clock\_config

|                     |  |
|---------------------|--|
| 函数名称                | usart_synchronous_clock_config   |
| 函数原型                | void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl); |
| 功能描述                | 配置USART同步通讯模式参数  |
| 先决条件                | -  |
| 被调用函数               | -  |
| 输入参数{in}            |  |
| usart_periph        | 外设USARTx   |
| USARTx              | x=0,1,2  |
| 输入参数{in}            |  |
| clen                | CK信号长度   |
| USART_CLEN_NO<br>NE | 8位数据帧中有7个CK脉冲, 9位数据帧中有8个CK脉冲   |
| USART_CLEN_EN       | 8位数据帧中有8个CK脉冲, 9位数据帧中有9个CK脉冲   |
| 输入参数{in}            |  |
| cph                 | 时钟相位   |
| USART_CPH_1CK       | 在首个时钟边沿采样第一个数据   |
| USART_CPH_2CK       | 在第二个时钟边沿采样第一个数据  |
| 输入参数{in}            |  |
| cpl                 | 时钟极性   |

|                |                  |
|----------------|------------------|
| USART_CPL_LOW  | CK引脚不对外发送时保持为低电平 |
| USART_CPL_HIGH | CK引脚不对外发送时保持为高电平 |
| 输出参数{out}      |                  |
| -              | -                |
| 返回值            |                  |
| -              | -                |

例如：

```
/* configure USART0 synchronous mode parameters */
```

```
usart_synchronous_clock_config(USART0,          USART_CLEN_EN,USART_CPH_2CK,
USART_CPL_HIGH);
```

### 函数 usart\_guard\_time\_config

函数usart\_guard\_time\_config描述见下表：

表 3-796. 函数 usart\_guard\_time\_config

|              |  |
|--------------|--|
| 函数名称         | usart_guard_time_config  |
| 函数原型         | void usart_guard_time_config(uint32_t usart_periph,uint32_t guat); |
| 功能描述         | 在USART智能卡模式下配置保护时间值  |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| usart_periph | 外设USARTx   |
| USARTx       | x=0,1,2  |
| 输入参数{in}     |  |
| guat         | 保护时间值（0x00000000-0x000000FF）                                       |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| -            | -  |

例如：

```
/* configure USART0 guard time value in smartcard mode */
```

```
usart_guard_time_config(USART0, 0x00000055);
```

### 函数 usart\_smartcard\_mode\_enable

函数usart\_smartcard\_mode\_enable描述见下表：

表 3-797. 函数 usart\_smartcard\_mode\_enable

|      |  |
|------|--|
| 函数名称 | usart_smartcard_mode_enable                              |
| 函数原型 | void usart_smartcard_mode_enable(uint32_t usart_periph); |

|              |              |
|--------------|--------------|
| 功能描述         | 使能USART智能卡模式 |
| 先决条件         | -            |
| 被调用函数        | -            |
| 输入参数{in}     |              |
| usart_periph | 外设USARTx     |
| USARTx       | x=0,1,2      |
| 输出参数{out}    |              |
| -            | -            |
| 返回值          |              |
| -            | -            |

例如：

```
/* USART0 smartcard mode enable */
usart_smartcard_mode_enable(USART0);
```

### 函数 usart\_smartcard\_mode\_disable

函数usart\_smartcard\_mode\_disable描述见下表：

表 3-798. 函数 usart\_smartcard\_mode\_disable

|              |   |
|--------------|---|
| 函数名称         | usart_smartcard_mode_disable                              |
| 函数原型         | void usart_smartcard_mode_disable(uint32_t usart_periph); |
| 功能描述         | 失能USART智能卡模式  |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如：

```
/* USART0 smartcard mode disable */
usart_smartcard_mode_disable(USART0);
```

### 函数 usart\_smartcard\_mode\_nack\_enable

函数usart\_smartcard\_mode\_nack\_enable描述见下表：

表 3-799. 函数 usart\_smartcard\_mode\_nack\_enable

|      |                                  |
|------|----------------------------------|
| 函数名称 | usart_smartcard_mode_nack_enable |
|------|----------------------------------|

|              |   |
|--------------|---|
| 函数原型         | void usart_smartcard_mode_nack_enable(uint32_t usart_periph); |
| 功能描述         | 在USART智能卡模式下使能NACK  |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如：

```
/* enable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_enable(USART0);
```

### 函数 usart\_smartcard\_mode\_nack\_disable

函数usart\_smartcard\_mode\_nack\_disable描述见下表：

表 3-800. 函数 usart\_smartcard\_mode\_nack\_disable

|              |  |
|--------------|--|
| 函数名称         | usart_smartcard_mode_nack_disable                              |
| 函数原型         | void usart_smartcard_mode_nack_disable(uint32_t usart_periph); |
| 功能描述         | 在USART智能卡模式下失能NACK   |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| usart_periph | 外设USARTx   |
| USARTx       | x=0,1,2  |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| -            | -  |

例如：

```
/* disable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_disable(USART0);
```

### 函数 usart\_smartcard\_mode\_early\_nack\_enable

函数usart\_smartcard\_mode\_early\_nack\_enable描述见下表：

表 3-801. 函数 `usart_smartcard_mode_early_nack_enable`

|                           |  |
|---------------------------|--|
| 函数名称                      | <code>usart_smartcard_mode_early_nack_enable</code>                              |
| 函数原型                      | <code>void usart_smartcard_mode_early_nack_enable(uint32_t usart_periph);</code> |
| 功能描述                      | 使能USART智能卡模式提前NACK   |
| 先决条件                      | -  |
| 被调用函数                     | -  |
| 输入参数{in}                  |  |
| <code>usart_periph</code> | 外设USARTx   |
| <code>USARTx</code>       | x=0,1,2  |
| 输出参数{out}                 |  |
| -                         | -  |
| 返回值                       |  |
| -                         | -  |

例如：

```
/* enable USART0 early NACK in smartcard mode */
usart_smartcard_mode_early_nack_enable(USART0);
```

### 函数 `usart_smartcard_mode_early_nack_disable`

函数`usart_smartcard_mode_early_nack_disable`描述见下表：

表 3-802. 函数 `usart_smartcard_mode_early_nack_disable`

|                           |   |
|---------------------------|---|
| 函数名称                      | <code>usart_smartcard_mode_early_nack_disable</code>                              |
| 函数原型                      | <code>void usart_smartcard_mode_early_nack_disable(uint32_t usart_periph);</code> |
| 功能描述                      | 失能USART智能卡模式提前NACK  |
| 先决条件                      | -   |
| 被调用函数                     | -   |
| 输入参数{in}                  |   |
| <code>usart_periph</code> | 外设USARTx  |
| <code>USARTx</code>       | x=0,1,2   |
| 输出参数{out}                 |   |
| -                         | -   |
| 返回值                       |   |
| -                         | -   |

例如：

```
/* disable USART0 early NACK in smartcard mode */
usart_smartcard_mode_early_nack_disable(USART0);
```

### 函数 `usart_smartcard_autoretry_config`

函数`usart_smartcard_autoretry_config`描述见下表：



表 3-803. 函数 `usart_smartcard_autoretry_config`

|                           |  |
|---------------------------|--|
| 函数名称                      | <code>usart_smartcard_autoretry_config</code>  |
| 函数原型                      | <code>void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrtnum);</code> |
| 功能描述                      | 配置智能卡自动重试次数  |
| 先决条件                      | -  |
| 被调用函数                     | -  |
| 输入参数{in}                  |  |
| <code>usart_periph</code> | 外设USARTx   |
| <code>USARTx</code>       | x=0,1,2  |
| 输入参数{in}                  |  |
| <code>scrtnum</code>      | 智能卡自动重试次数 (0x00000000-0x00000007)  |
| 输出参数{out}                 |  |
| -                         | -  |
| 返回值                       |  |
| -                         | -  |

例如:

```
/* configure smartcard auto-retry number */
usart_smartcard_autoretry_config(USART0, 0x00000007);
```

### 函数 `usart_block_length_config`

函数`usart_block_length_config`描述见下表:

表 3-804. 函数 `usart_block_length_config`

|                           |  |
|---------------------------|--|
| 函数名称                      | <code>usart_block_length_config</code>   |
| 函数原型                      | <code>void usart_block_length_config(uint32_t usart_periph, uint32_t bl);</code> |
| 功能描述                      | 配置智能卡T=1的接收时块的长度   |
| 先决条件                      | -  |
| 被调用函数                     | -  |
| 输入参数{in}                  |  |
| <code>usart_periph</code> | 外设USARTx   |
| <code>USARTx</code>       | x=0,1,2  |
| 输入参数{in}                  |  |
| <code>bl</code>           | 块长度 (0x00000000-0x000000FF)  |
| 输出参数{out}                 |  |
| -                         | -  |
| 返回值                       |  |
| -                         | -  |

例如:

```
/* configure block length in Smartcard T=1 reception */
```

```
usart_block_length_config(USART0, 0x000000FF);
```

### 函数 usart\_irda\_mode\_enable

函数usart\_irda\_mode\_enable描述见下表：

**表 3-805. 函数 usart\_irda\_mode\_enable**

|              |   |
|--------------|---|
| 函数名称         | usart_irda_mode_enable                              |
| 函数原型         | void usart_irda_mode_enable(uint32_t usart_periph); |
| 功能描述         | 使能USART串行红外编解码功能模块                                  |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如：

```
/* enable USART0 IrDA mode */
```

```
usart_irda_mode_enable(USART0);
```

### 函数 usart\_irda\_mode\_disable

函数usart\_irda\_mode\_disable描述见下表：

**表 3-806. 函数 usart\_irda\_mode\_disable**

|              |  |
|--------------|--|
| 函数名称         | usart_irda_mode_disable                              |
| 函数原型         | void usart_irda_mode_disable(uint32_t usart_periph); |
| 功能描述         | 失能USART串行红外编解码功能模块                                   |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| usart_periph | 外设USARTx   |
| USARTx       | x=0,1,2  |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| -            | -  |

例如：

```
/* disable USART0 IrDA mode */
```

```
usart_irda_mode_disable(USART0);
```

### 函数 usart\_prescaler\_config

函数usart\_prescaler\_config描述见下表：

表 3-807. 函数 usart\_prescaler\_config

|              |   |
|--------------|---|
| 函数名称         | usart_prescaler_config  |
| 函数原型         | void usart_prescaler_config(uint32_t usart_periph, uint32_t psc); |
| 功能描述         | 在USART IrDA低功耗模式下配置外设时钟分频系数                                       |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输入参数{in}     |   |
| psc          | 时钟分频系数（0x00000000-0x000000FF）                                     |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如：

```
/* configure the USART0 peripheral clock prescaler in USART IrDA low-power mode */
usart_prescaler_config(USART0, 0x00000001);
```

### 函数 usart\_irda\_lowpower\_config

函数usart\_irda\_lowpower\_config描述见下表：

表 3-808. 函数 usart\_irda\_lowpower\_config

|                   |  |
|-------------------|--|
| 函数名称              | usart_irda_lowpower_config   |
| 函数原型              | void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp); |
| 功能描述              | 配置USART IrDA低功耗模式  |
| 先决条件              | -  |
| 被调用函数             | -  |
| 输入参数{in}          |  |
| usart_periph      | 外设USARTx   |
| USARTx            | x=0,1,2  |
| 输入参数{in}          |  |
| irlp              | IrDA低功耗模式或正常模式   |
| USART_IRLP_LOW    | 低功耗模式  |
| USART_IRLP_NORMAL | 正常模式   |

| 输出参数{out} |   |
|-----------|---|
| -         | - |
| 返回值       |   |
| -         | - |

例如:

```
/* configure USART0 IrDA low-power */
```

```
usart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

### 函数 usart\_hardware\_flow\_rts\_config

函数usart\_hardware\_flow\_rts\_config描述见下表:

表 3-809. 函数 usart\_hardware\_flow\_rts\_config

| 函数名称                  | usart_hardware_flow_rts_config  |
|-----------------------|---|
| 函数原型                  | void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig); |
| 功能描述                  | 配置USART RTS硬件控制流  |
| 先决条件                  | -   |
| 被调用函数                 | -   |
| 输入参数{in}              |   |
| usart_periph          | 外设USARTx  |
| USARTx                | x=0,1,2   |
| 输入参数{in}              |   |
| rtsconfig             | 使能/失能RTS  |
| USART_RTS_ENA<br>BLE  | 使能RTS   |
| USART_RTS_DISA<br>BLE | 失能RTS   |
| 输出参数{out}             |   |
| -                     | -   |
| 返回值                   |   |
| -                     | -   |

例如:

```
/* configure USART0 hardware flow control RTS */
```

```
usart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);
```

### 函数 usart\_hardware\_flow\_cts\_config

函数usart\_hardware\_flow\_cts\_config描述见下表:

表 3-810. 函数 usart\_hardware\_flow\_cts\_config

| 函数名称 | usart_hardware_flow_cts_config |
|------|--------------------------------|
|------|--------------------------------|

|                       |   |
|-----------------------|---|
| 函数原型                  | void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig); |
| 功能描述                  | 配置USART CTS硬件控制流  |
| 先决条件                  | -   |
| 被调用函数                 | -   |
| 输入参数{in}              |   |
| usart_periph          | 外设USARTx  |
| USARTx                | x=0,1,2   |
| 输入参数{in}              |   |
| ctsconfig             | 使能/失能CTS  |
| USART_CTS_ENA<br>BLE  | 使能CTS   |
| USART_CTS_DISA<br>BLE | 失能CTS   |
| 输出参数{out}             |   |
| -                     | -   |
| 返回值                   |   |
| -                     | -   |

例如:

```
/* configure USART0 hardware flow control CTS */
```

```
usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

### 函数 usart\_hardware\_flow\_coherence\_config

函数usart\_hardware\_flow\_coherence\_config描述见下表:

表 3-811. 函数 usart\_hardware\_flow\_coherence\_config

|                    |   |
|--------------------|---|
| 函数名称               | usart_hardware_flow_coherence_config  |
| 函数原型               | void usart_hardware_flow_coherence_config(uint32_t usart_periph, uint32_t hcm); |
| 功能描述               | 配置硬件流控兼容模式  |
| 先决条件               | -   |
| 被调用函数              | -   |
| 输入参数{in}           |   |
| usart_periph       | 外设USARTx  |
| USARTx             | x=0,1,2   |
| 输入参数{in}           |   |
| hcm                | 硬件流控制兼容模式   |
| USART_HCM_NON<br>E | nRTS信号与USART_STAT0寄存器中RBNE位相同   |
| USART_HCM_EN       | nRTS信号在最后一个数据位被采样后被置位   |
| 输出参数{out}          |   |
| -                  | -   |

| 返回值 |   |
|-----|---|
| -   | - |

例如：

```
/* configure hardware flow control coherence mode */
```

```
usart_hardware_flow_coherence_config(USART0, USART_HCM_NONE);
```

### 函数 usart\_rs485\_driver\_enable

函数usart\_rs485\_driver\_enable描述见下表：

表 3-812. 函数 usart\_rs485\_driver\_enable

| 函数名称         | usart_rs485_driver_enable                              |
|--------------|--|
| 函数原型         | void usart_rs485_driver_enable(uint32_t usart_periph); |
| 功能描述         | 使能USART rs485驱动  |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| usart_periph | 外设USARTx   |
| USARTx       | x=0,1,2  |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| -            | -  |

例如：

```
/* enable USART0 RS485 driver */
```

```
usart_rs485_driver_enable(USART0);
```

### 函数 usart\_rs485\_driver\_disable

函数usart\_rs485\_driver\_disable描述见下表：

表 3-813. 函数 usart\_rs485\_driver\_disable

| 函数名称         | usart_rs485_driver_disable                              |
|--------------|---|
| 函数原型         | void usart_rs485_driver_disable(uint32_t usart_periph); |
| 功能描述         | 失能USART rs485驱动   |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输出参数{out}    |   |

|     |   |
|-----|---|
| -   | - |
| 返回值 |   |
| -   | - |

例如：

```
/* disable USART0 RS485 driver */
```

```
usart_rs485_driver_disable(USART0);
```

### 函数 usart\_driver\_assertime\_config

函数usart\_driver\_assertime\_config描述见下表：

表 3-814. 函数 usart\_driver\_assertime\_config

|              |  |
|--------------|--|
| 函数名称         | usart_driver_assertime_config  |
| 函数原型         | void usart_driver_assertime_config(uint32_t usart_periph, uint32_t deatime); |
| 功能描述         | 配置USART驱动使能置位时间  |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| usart_periph | 外设USARTx   |
| USARTx       | x=0,1,2  |
| 输入参数{in}     |  |
| deatime      | 驱动使能置位时间（0x00000000-0x0000001F）  |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| -            | -  |

例如：

```
/* set USART0 driver assertime */
```

```
usart_driver_assertime_config(USART0, 0x0000001F);
```

### 函数 usart\_driver\_deassertime\_config

函数usart\_driver\_deassertime\_config描述见下表：

表 3-815. 函数 usart\_driver\_deassertime\_config

|          |  |
|----------|--|
| 函数名称     | usart_driver_deassertime_config  |
| 函数原型     | void usart_driver_deassertime_config(uint32_t usart_periph, uint32_t dedtime); |
| 功能描述     | 配置USART驱动使能置低时间  |
| 先决条件     | -  |
| 被调用函数    | -  |
| 输入参数{in} |  |

|                     |                                 |
|---------------------|---------------------------------|
| <b>usart_periph</b> | 外设USARTx                        |
| <i>USARTx</i>       | x=0,1,2                         |
| <b>输入参数{in}</b>     |                                 |
| <b>dedtime</b>      | 驱动使能置低时间（0x00000000-0x0000001F） |
| <b>输出参数{out}</b>    |                                 |
| -                   | -                               |
| <b>返回值</b>          |                                 |
| -                   | -                               |

例如：

```
/* set USART0 driver deasserttime */
```

```
usart_driver_deasserttime_config(USART0, 0x0000001F);
```

### 函数 usart\_depolarity\_config

函数usart\_depolarity\_config描述见下表：

**表 3-816. 函数 usart\_depolarity\_config**

|                       |  |
|-----------------------|--|
| <b>函数名称</b>           | usart_depolarity_config  |
| <b>函数原型</b>           | void usart_depolarity_config(uint32_t usart_periph, uint32_t dep); |
| <b>功能描述</b>           | 配置USART驱动使能极性模式  |
| <b>先决条件</b>           | -  |
| <b>被调用函数</b>          | -  |
| <b>输入参数{in}</b>       |  |
| <b>usart_periph</b>   | 外设USARTx   |
| <i>USARTx</i>         | x=0,1,2  |
| <b>输入参数{in}</b>       |  |
| <b>dep</b>            | 驱动使能的极性选择模式  |
| <i>USART_DEP_HIGH</i> | DE信号高有效  |
| <i>USART_DEP_LOW</i>  | DE信号低有效  |
| <b>输出参数{out}</b>      |  |
| -                     | -  |
| <b>返回值</b>            |  |
| -                     | -  |

例如：

```
/* configure driver enable polarity mode */
```

```
usart_driver_depolarity_config(USART0, USART_DEP_HIGH);
```

### 函数 usart\_dma\_receive\_config

函数usart\_dma\_receive\_config描述见下表：



表 3-817. 函数 usart\_dma\_receive\_config

|                           |   |
|---------------------------|---|
| 函数名称                      | usart_dma_receive_config  |
| 函数原型                      | void usart_dma_receive_config(uint32_t usart_periph, uint8_t dmacmd); |
| 功能描述                      | 配置USART DMA接收   |
| 先决条件                      | -   |
| 被调用函数                     | -   |
| 输入参数{in}                  |   |
| usart_periph              | 外设USARTx  |
| USARTx                    | x=0,1,2   |
| 输入参数{in}                  |   |
| dmacmd                    | USART DMA模式   |
| USART_RECEIVE_DMA_ENABLE  | 使能USART DMA接收   |
| USART_RECEIVE_DMA_DISABLE | 失能USART DMA发送   |
| 输出参数{out}                 |   |
| -                         | -   |
| 返回值                       |   |
| -                         | -   |

例如:

```
/* enable USART0 DMA for reception */
```

```
usart_dma_receive_config(USART0, USART_RECEIVE_DMA_ENABLE);
```

### 函数 usart\_dma\_transmit\_config

函数usart\_dma\_transmit\_config描述见下表:

表 3-818. 函数 usart\_dma\_transmit\_config

|                            |  |
|----------------------------|--|
| 函数名称                       | usart_dma_transmit_config  |
| 函数原型                       | void usart_dma_transmit_config(uint32_t usart_periph, uint8_t dmacmd); |
| 功能描述                       | 配置USART DMA发送  |
| 先决条件                       | -  |
| 被调用函数                      | -  |
| 输入参数{in}                   |  |
| usart_periph               | 外设USARTx   |
| USARTx                     | x=0,1,2  |
| 输入参数{in}                   |  |
| dmacmd                     | USART DMA模式  |
| USART_TRANSMIT_DMA_ENABLE  | USART DMA接收  |
| USART_TRANSMIT_DMA_DISABLE | USART DMA发送  |

| 输出参数{out} |   |
|-----------|---|
| -         | - |
| 返回值       |   |
| -         | - |

例如：

```
/* enable USART0 DMA for transmission */
```

```
usart_dma_transmit_config(USART0, USART_TRANSMIT_DMA_ENABLE);
```

### 函数 usart\_reception\_error\_dma\_disable

函数usart\_reception\_error\_dma\_disable描述见下表：

表 3-819. 函数 usart\_reception\_error\_dma\_disable

| 函数名称         | usart_reception_error_dma_disable                              |
|--------------|--|
| 函数原型         | void usart_reception_error_dma_disable(uint32_t usart_periph); |
| 功能描述         | USART接收错误时失能DMA  |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| usart_periph | 外设USARTx   |
| USARTx       | x=0,1,2  |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| -            | -  |

例如：

```
/* disable DMA on reception error */
```

```
usart_reception_error_dma_disable(USART0);
```

### 函数 usart\_reception\_error\_dma\_enable

函数usart\_reception\_error\_dma\_enable描述见下表：

表 3-820. 函数 usart\_reception\_error\_dma\_enable

| 函数名称         | usart_reception_error_dma_enable                              |
|--------------|---|
| 函数原型         | void usart_reception_error_dma_enable(uint32_t usart_periph); |
| 功能描述         | USART接收错误时使能DMA   |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |

|           |         |
|-----------|---------|
| USARTx    | x=0,1,2 |
| 输出参数{out} |         |
| -         | -       |
| 返回值       |         |
| -         | -       |

例如:

```
/* enable DMA on reception error */
usart_reception_error_dma_enable(USART0);
```

### 函数 usart\_wakeup\_enable

函数usart\_wakeup\_enable描述见下表:

表 3-821. 函数 usart\_wakeup\_enable

|              |  |
|--------------|--|
| 函数名称         | usart_wakeup_enable                              |
| 函数原型         | void usart_wakeup_enable(uint32_t usart_periph); |
| 功能描述         | 使能USART唤醒  |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| usart_periph | 外设USARTx   |
| USARTx       | x=0,1,2  |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| -            | -  |

例如:

```
/* USART0 wake up enable */
usart_wakeup_enable(USART0);
```

### 函数 usart\_wakeup\_disable

函数usart\_wakeup\_disable描述见下表:

表 3-822. 函数 usart\_wakeup\_disable

|          |   |
|----------|---|
| 函数名称     | usart_wakeup_disable                              |
| 函数原型     | void usart_wakeup_disable(uint32_t usart_periph); |
| 功能描述     | 失能USART唤醒   |
| 先决条件     | -   |
| 被调用函数    | -   |
| 输入参数{in} |   |

|                     |          |
|---------------------|----------|
| <b>usart_periph</b> | 外设USARTx |
| <i>USARTx</i>       | x=0,1,2  |
| <b>输出参数{out}</b>    |          |
| -                   | -        |
| <b>返回值</b>          |          |
| -                   | -        |

例如：

```
/* USART0 wake up disable */
```

```
usart_wakeup_disable(USART0);
```

### 函数 usart\_wakeup\_mode\_config

函数usart\_reception\_mode\_config描述见下表：

**表 3-823. 函数 usart\_wakeup\_mode\_config**

|                              |   |
|------------------------------|---|
| <b>函数名称</b>                  | usart_wakeup_mode_config  |
| <b>函数原型</b>                  | void usart_wakeup_mode_config(uint32_t usart_periph, uint32_t wum); |
| <b>功能描述</b>                  | 配置USART唤醒模式   |
| <b>先决条件</b>                  | -   |
| <b>被调用函数</b>                 | -   |
| <b>输入参数{in}</b>              |   |
| <b>usart_periph</b>          | 外设USARTx  |
| <i>USARTx</i>                | x=0,1,2   |
| <b>输入参数{in}</b>              |   |
| <b>wum</b>                   | 唤醒模式  |
| <i>USART_WUM_ADD<br/>R</i>   | WUF在地址匹配时置位   |
| <i>USART_WUM_STA<br/>RTB</i> | WUF在检测到起始位时置位   |
| <i>USART_WUM_RBN<br/>E</i>   | WUF在检测到RBNE时置位  |
| <b>输出参数{out}</b>             |   |
| -                            | -   |
| <b>返回值</b>                   |   |
| -                            | -   |

例如：

```
/* configure USART0 wake up mode */
```

```
usart_wakeup_mode_config(USART0, USART_WUM_ADDR);
```

**函数 usart\_receive\_fifo\_enable**

函数usart\_receive\_fifo\_enable描述见下表:

**表 3-824. 函数 usart\_receive\_fifo\_enable**

|              |  |
|--------------|--|
| 函数名称         | usart_receive_fifo_enable                              |
| 函数原型         | void usart_receive_fifo_enable(uint32_t usart_periph); |
| 功能描述         | 使能接收FIFO   |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| usart_periph | 外设USARTx   |
| USARTx       | x=0,1,2  |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| -            | -  |

例如:

```
/* enable receive FIFO */  
  
usart_receive_fifo_enable(USART0);
```

**函数 usart\_receive\_fifo\_disable**

函数usart\_receive\_fifo\_disable描述见下表:

**表 3-825. 函数 usart\_receive\_fifo\_disable**

|              |   |
|--------------|---|
| 函数名称         | usart_receive_fifo_disable                              |
| 函数原型         | void usart_receive_fifo_disable(uint32_t usart_periph); |
| 功能描述         | 失能接收FIFO  |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如:

```
/* disable receive FIFO */  
  
usart_receive_fifo_disable(USART0);
```

**函数 usart\_receive\_fifo\_counter\_number**

函数usart\_receive\_fifo\_counter\_number描述见下表：

**表 3-826. 函数 usart\_receive\_fifo\_counter\_number**

|              |   |
|--------------|---|
| 函数名称         | usart_receive_fifo_counter_number                                 |
| 函数原型         | uint8_t usart_receive_fifo_counter_number(uint32_t usart_periph); |
| 功能描述         | 读取接收FIFO计数器的值   |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| uint8_t      | 接收FIFO计数器的值   |

例如：

```
/* read receive FIFO counter number */

uint8_t temp;

temp = usart_receive_fifo_counter_number(USART0);
```

**函数 usart\_flag\_get**

函数usart\_flag\_get描述见下表：

**表 3-827. 函数 usart\_flag\_get**

|              |   |
|--------------|---|
| 函数名称         | usart_flag_get  |
| 函数原型         | FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag); |
| 功能描述         | 获取USART STAT/CHC/RFCS寄存器标志位   |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输入参数{in}     |   |
| flag         | USART标志位，参考 <a href="#">表3-758. 枚举类型usart_flag_enum</a><br>只能选择一个参数     |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| FlagStatus   | SET或RESET   |

例如:

```
/* get flag USART0 state */
```

```
FlagStatus status;
```

```
status = usart_flag_get(USART0, USART_FLAG_TBE);
```

### 函数 usart\_flag\_clear

函数usart\_flag\_clear描述见下表:

表 3-828. 函数 usart\_flag\_clear

| 函数名称                 | usart_flag_clear   |
|----------------------|--|
| 函数原型                 | void usart_flag_clear(uint32_t usart_periph, usart_flag_enum flag);  |
| 功能描述                 | 清除USART状态寄存器标志位  |
| 先决条件                 | -  |
| 被调用函数                | -  |
| 输入参数{in}             |  |
| usart_periph         | 外设USARTx   |
| USARTx               | x=0,1,2  |
| 输入参数{in}             |  |
| flag                 | USART标志位, 参考 <a href="#">表3-758. 枚举类型usart_flag_enum</a><br>只能选择一个参数 |
| USART_FLAG_WU        | 从深度睡眠模式唤醒标志  |
| USART_FLAG_AM        | 地址匹配标志   |
| USART_FLAG_EB        | 块结束标志  |
| USART_FLAG_RT        | 接收超时标志   |
| USART_FLAG_CTS<br>F  | CTS变化标志  |
| USART_FLAG_TC        | 发送完成   |
| USART_FLAG_RB<br>NE  | 读数据缓冲区非空   |
| USART_FLAG_IDL<br>E  | 空闲线检测标志  |
| USART_FLAG_OR<br>ERR | 溢出错误   |
| USART_FLAG_NE<br>RR  | 噪声错误标志   |
| USART_FLAG_FER<br>R  | 帧错误  |
| USART_FLAG_PE<br>RR  | 校验错误   |
| USART_FLAG_EPE<br>RR | 校验错误超前检测标志   |

| 输出参数{out} |   |
|-----------|---|
| -         | - |
| 返回值       |   |
| -         | - |

例如：

```
/* clear USART0 flag */
```

```
usart_flag_clear(USART0, USART_FLAG_TC);
```

### 函数 usart\_interrupt\_enable

函数usart\_interrupt\_enable描述见下表：

表 3-829. 函数 usart\_interrupt\_enable

| 函数名称         | usart_interrupt_enable  |
|--------------|---|
| 函数原型         | void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt); |
| 功能描述         | 使能USART中断   |
| 先决条件         | -   |
| 被调用函数        | -   |
| 输入参数{in}     |   |
| usart_periph | 外设USARTx  |
| USARTx       | x=0,1,2   |
| 输入参数{in}     |   |
| interrupt    | USART中断，参考 <a href="#">表3-760. 枚举类型usart_interrupt_enum</a><br>只能选择一个参数             |
| 输出参数{out}    |   |
| -            | -   |
| 返回值          |   |
| -            | -   |

例如：

```
/* enable USART0 TBE interrupt */
```

```
usart_interrupt_enable(USART0, USART_INT_TBE);
```

### 函数 usart\_interrupt\_disable

函数usart\_interrupt\_disable描述见下表：

表 3-830. 函数 usart\_interrupt\_disable

| 函数名称 | usart_interrupt_disable  |
|------|--|
| 函数原型 | void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt); |



|              |  |
|--------------|--|
| 功能描述         | 失能USART中断  |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| usart_periph | 外设USARTx   |
| USARTx       | x=0,1,2  |
| 输入参数{in}     |  |
| interrupt    | USART中断, 参考 <a href="#">表3-760. 枚举类型usart_interrupt_enum</a><br>只能选择一个参数 |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| -            | -  |

例如:

```
/* disable USART0 TBE interrupt */
```

```
usart_interrupt_disable(USART0, USART_INT_TBE);
```

### 函数 usart\_interrupt\_flag\_get

函数usart\_interrupt\_flag\_get描述见下表:

表 3-831. 函数 usart\_interrupt\_flag\_get

|              |  |
|--------------|--|
| 函数名称         | usart_interrupt_flag_get   |
| 函数原型         | FlagStatus usart_interrupt_flag_get(uint32_t usart_periph,<br>usart_interrupt_flag_enum int_flag); |
| 功能描述         | 获取USART中断标志位状态   |
| 先决条件         | -  |
| 被调用函数        | -  |
| 输入参数{in}     |  |
| usart_periph | 外设USARTx   |
| USARTx       | x=0,1,2  |
| 输入参数{in}     |  |
| int_flag     | USART中断标志, 参考 <a href="#">表3-759. 枚举类型usart_interrupt_flag_enum</a><br>只能选择一个参数                    |
| 输出参数{out}    |  |
| -            | -  |
| 返回值          |  |
| FlagStatus   | SET或RESET  |

例如:

```
/* get the USART0 interrupt flag status */
```

```
FlagStatus status;
```

```
status = usart_interrupt_flag_get(USART0, USART_INT_FLAG_RBNE);
```

## 函数 usart\_interrupt\_flag\_clear

函数usart\_interrupt\_flag\_clear描述见下表：

**表 3-832. 函数 usart\_interrupt\_flag\_clear**

|                               |  |
|-------------------------------|--|
| 函数名称                          | usart_interrupt_flag_clear   |
| 函数原型                          | void usart_interrupt_flag_clear(uint32_t usart_periph,<br>usart_interrupt_flag_enum int_flag); |
| 功能描述                          | 清除USART中断标志位状态   |
| 先决条件                          | -  |
| 被调用函数                         | -  |
| 输入参数{in}                      |  |
| usart_periph                  | 外设USARTx   |
| USARTx                        | x=0,1,2  |
| 输入参数{in}                      |  |
| int_flag                      | USART中断标志，参考 <a href="#">表3-759. 枚举类型usart_interrupt_flag_enum</a><br>只能选择一个参数                 |
| USART_INT_FLAG<br>_EB         | 块结束中断标志  |
| USART_INT_FLAG<br>_RT         | 接收超时中断标志   |
| USART_INT_FLAG<br>_AM         | 地址匹配中断标志   |
| USART_INT_FLAG<br>_PERR       | 奇偶校验错误中断标志   |
| USART_INT_FLAG<br>_TC         | 发送完成中断标志   |
| USART_INT_FLAG<br>_RBNE_ORERR | 读缓冲区非空和溢出中断标志  |
| USART_INT_FLAG<br>_IDLE       | 空闲线检测中断标志  |
| USART_INT_FLAG<br>_LBD        | LIN断开检测中断标志  |
| USART_INT_FLAG<br>_WU         | 从深度睡眠模式唤醒中断标志  |
| USART_INT_FLAG<br>_CTS        | CTS中断标志  |
| USART_INT_FLAG<br>_ERR_NERR   | 噪声错误中断标志   |
| USART_INT_FLAG<br>_ERR_ORERR  | 溢出错误中断标志   |
| USART_INT_FLAG                | 帧错误中断标志  |

|                        |             |
|------------------------|-------------|
| _ERR_FERR              |             |
| USART_INT_FLAG<br>_RFF | 接收FIFO满中断标志 |
| 输出参数{out}              |             |
| -                      | -           |
| 返回值                    |             |
| -                      | -           |

例如:

```
/* clear the USART0 interrupt flag */
```

```
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_TC);
```

## 3.25. WWDGT

窗口看门狗定时器(WWDGT)用来监测由软件故障导致的系统故障。章节[3.25.1](#)描述了WWDGT的寄存器列表，章节[3.25.2](#)对WWDGT库函数进行说明。

### 3.25.1. 外设寄存器说明

WWDGT寄存器列表如下表所示:

表 3-833. WWDGT 寄存器

| 寄存器名称      | 寄存器描述 |
|------------|-------|
| WWDGT_CTL  | 控制寄存器 |
| WWDGT_CFG  | 配置寄存器 |
| WWDGT_STAT | 状态寄存器 |

### 3.25.2. 外设库函数说明

WWDGT库函数列表如下表所示:

表 3-834. WWDGT 库函数

| 库函数名称                  | 库函数说明                |
|------------------------|----------------------|
| wwdgt_deinit           | 将WWDGT寄存器重设为缺省值      |
| wwdgt_enable           | 使能WWDGT              |
| wwdgt_counter_update   | 设置WWDGT计数器更新值        |
| wwdgt_config           | 设置WWDGT计数器值、窗口值和预分频值 |
| wwdgt_interrupt_enable | 使能WWDGT提前唤醒中断        |
| wwdgt_flag_get         | 检查WWDGT提前唤醒中断标志位是否置位 |
| wwdgt_flag_clear       | 清除WWDGT提前唤醒中断标志位状态   |

## 函数 wwdgt\_deinit

函数wwdgt\_deinit描述见下表:

**表 3-835. 函数 wwdgt\_deinit**

|           |                          |
|-----------|--------------------------|
| 函数名称      | wwdgt_deinit             |
| 函数原型      | void wwdgt_deinit(void); |
| 功能描述      | 将WWDGT寄存器重设为缺省值          |
| 先决条件      | -                        |
| 被调用函数     | -                        |
| 输入参数{in}  |                          |
| -         | -                        |
| 输出参数{out} |                          |
| -         | -                        |
| 返回值       |                          |
| -         | -                        |

例如:

```
/* reset the WWDGT configuration */
```

```
wwdgt_deinit();
```

## 函数 wwdgt\_enable

函数wwdgt\_enable描述见下表:

**表 3-836. 函数 wwdgt\_enable**

|           |                          |
|-----------|--------------------------|
| 函数名称      | wwdgt_enable             |
| 函数原型      | void wwdgt_enable(void); |
| 功能描述      | 使能WWDGT                  |
| 先决条件      | -                        |
| 被调用函数     | -                        |
| 输入参数{in}  |                          |
| -         | -                        |
| 输出参数{out} |                          |
| -         | -                        |
| 返回值       |                          |
| -         | -                        |

例如:

```
/* start the WWDGT counter */
```

```
wwdgt_enable();
```

函数 **wwdgt\_counter\_update**

函数 **wwdgt\_counter\_update** 描述见下表：

表 3-837. 函数 **wwdgt\_counter\_update**

|               |  |
|---------------|--|
| 函数名称          | wwdgt_counter_update                               |
| 函数原型          | void wwdgt_counter_update(uint16_t counter_value); |
| 功能描述          | 设置WWDGT计数器更新值                                      |
| 先决条件          | -  |
| 被调用函数         | -  |
| 输入参数{in}      |  |
| counter_value | 计数器值，数值范围为0x00000000 - 0x0000007F                  |
| 输出参数{out}     |  |
| -             | -  |
| 返回值           |  |
| -             | -  |

例如：

```
/* update WWDGT counter to 0x7F */
```

```
wwdgt_counter_update(127);
```

函数 **wwdgt\_config**

函数 **wwdgt\_config** 描述见下表：

表 3-838. 函数 **wwdgt\_config**

|                    |   |
|--------------------|---|
| 函数名称               | wwdgt_config  |
| 函数原型               | void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler); |
| 功能描述               | 设置WWDGT计数器值、窗口值和预分频值  |
| 先决条件               | -   |
| 被调用函数              | -   |
| 输入参数{in}           |   |
| counter            | 定时器计数值，数值范围0x00000000 - 0x0000007F  |
| 输入参数{in}           |   |
| window             | 窗口值，数值范围0x00000000 - 0x0000007F   |
| 输入参数{in}           |   |
| prescaler          | WWDGT预分频值   |
| WWDGT_CFG_PSC_DIV1 | WWDGT计数器时钟为 (PCLK/4096) /1  |
| WWDGT_CFG_PSC_DIV2 | WWDGT计数器时钟为 (PCLK/4096) /2  |
| WWDGT_CFG_PSC_DIV4 | WWDGT计数器时钟为 (PCLK/4096) /4  |
| WWDGT_CFG_PSC      | WWDGT计数器时钟为 (PCLK/4096) /8  |

|                                    |                               |
|------------------------------------|-------------------------------|
| <code>_DIV8</code>                 |                               |
| <code>WWDGT_CFG_PSC_DIV16</code>   | WWDGT计数器时钟为 (PCLK/4096) /16   |
| <code>WWDGT_CFG_PSC_DIV32</code>   | WWDGT计数器时钟为 (PCLK/4096) /32   |
| <code>WWDGT_CFG_PSC_DIV64</code>   | WWDGT计数器时钟为 (PCLK/4096) /64   |
| <code>WWDGT_CFG_PSC_DIV128</code>  | WWDGT计数器时钟为 (PCLK/4096) /128  |
| <code>WWDGT_CFG_PSC_DIV256</code>  | WWDGT计数器时钟为 (PCLK/4096) /256  |
| <code>WWDGT_CFG_PSC_DIV512</code>  | WWDGT计数器时钟为 (PCLK/4096) /512  |
| <code>WWDGT_CFG_PSC_DIV1024</code> | WWDGT计数器时钟为 (PCLK/4096) /1024 |
| <code>WWDGT_CFG_PSC_DIV2048</code> | WWDGT计数器时钟为 (PCLK/4096) /2048 |
| <code>WWDGT_CFG_PSC_DIV4096</code> | WWDGT计数器时钟为 (PCLK/4096) /4096 |
| <code>WWDGT_CFG_PSC_DIV8192</code> | WWDGT计数器时钟为 (PCLK/4096) /8192 |
| 输出参数{out}                          |                               |
| -                                  | -                             |
| Return value                       |                               |
| -                                  | -                             |

例如:

```
/* configure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
```

```
wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

### 函数 `wwdgt_interrupt_enable`

函数 `wwdgt_interrupt_enable` 描述见下表:

表 3-839. 函数 `wwdgt_interrupt_enable`

|          |   |
|----------|---|
| 函数名称     | <code>wwdgt_interrupt_enable</code>             |
| 函数原型     | <code>void wwdgt_interrupt_enable(void);</code> |
| 功能描述     | 使能WWDGT提前唤醒中断                                   |
| 先决条件     | -   |
| 被调用函数    | -   |
| 输入参数{in} |   |
| -        | -   |

| 输出参数{out} |   |
|-----------|---|
| -         | - |
| 返回值       |   |
| -         | - |

例如:

```
/* enable early wakeup interrupt of WWDGT */
```

```
wwdgt_interrupt_enable();
```

### 函数 wwdgt\_flag\_get

函数wwdgt\_flag\_get描述见下表:

表 3-840. 函数 wwdgt\_flag\_get

| 函数名称       | wwdgt_flag_get                   |
|------------|----------------------------------|
| 函数原型       | FlagStatus wwdgt_flag_get(void); |
| 功能描述       | 检查WWDGT提前唤醒中断标志位是否置位             |
| 先决条件       | -                                |
| 被调用函数      | -                                |
| 输入参数{in}   |                                  |
| -          | -                                |
| 输出参数{out}  |                                  |
| -          | -                                |
| 返回值        |                                  |
| FlagStatus | SET or RESET                     |

例如:

```
/* test if the counter value update has reached the 0x40 */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get();
```

### 函数 wwdgt\_flag\_clear

函数wwdgt\_flag\_clear描述见下表:

表 3-841. 函数 wwdgt\_flag\_clear

| 函数名称     | wwdgt_flag_clear             |
|----------|------------------------------|
| 函数原型     | void wwdgt_flag_clear(void); |
| 功能描述     | 清除WWDGT提前唤醒中断标志位状态           |
| 先决条件     | -                            |
| 被调用函数    | -                            |
| 输入参数{in} |                              |
| -        | -                            |

| 输出参数{out} |   |
|-----------|---|
| -         | - |
| 返回值       |   |
| -         | - |

例如:

```
/* clear early wakeup interrupt state of WWDGT */
```

```
wwdgt_flag_clear();
```



## 4. 版本历史

表 4-1. 版本历史

| 版本号. | 说明  | 日期               |
|------|---|------------------|
| 1.0  | 初稿发布  | 2022 年 4 月 7 日   |
| 1.1  | 1.在 3.20.2 章节中增加函数接口 spi_i2s_format_error_clear。<br>2. 修改 3.24.2 章节中函数接口 usart_dma_enable / usart_dma_disable 为 usart_dma_receive_config / usart_dma_transmit_config。 | 2022 年 12 月 29 日 |
| 1.2  | 1.更新 <b>CMP</b> 章节。<br>2.更新 <b>DAC</b> 章节。  | 2024 年 1 月 13 日  |
| 1.3  | 1.更新 <b>DMA</b> 章节。   | 2024 年 8 月 20 日  |
| 1.4  | 1. 删除 <b>FMC</b> 章节中与硬件 EEPROM 相关的内容。   | 2024 年 12 月 6 日  |
| 1.5  | 1. 删除 FMC 章节中函数 ob1_parameter_config 的入参 OB1CS_DF_16K。  | 2025 年 2 月 6 日   |

## 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.