

**GigaDevice Semiconductor Inc.**

**GD32E51x**

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

**固件库  
使用指南**

1.0 版本

(2024 年 6 月)

# 目录

目录 .....	2
图索引 .....	5
表索引 .....	6
<b>1. 介绍 .....</b>	<b>31</b>
<b>1.1. 文档和固件库规则 .....</b>	<b>31</b>
1.1.1. 外设缩写 .....	31
1.1.2. 命名规则 .....	32
<b>2. 固件库概述 .....</b>	<b>33</b>
<b>2.1. 文件组织结构 .....</b>	<b>33</b>
2.1.1. Examples 文件夹 .....	33
2.1.2. Firmware 文件夹 .....	34
2.1.3. Template 文件夹 .....	34
2.1.4. Utilities 文件夹 .....	37
<b>2.2. 固件库文件描述 .....</b>	<b>38</b>
<b>3. 外设固件库 .....</b>	<b>39</b>
<b>3.1. 外设固件库概述 .....</b>	<b>39</b>
<b>3.2. ADC .....</b>	<b>39</b>
3.2.1. 外设寄存器描述 .....	39
3.2.2. 外设库函数说明 .....	40
<b>3.3. BKP .....</b>	<b>75</b>
3.3.1. 外设寄存器说明 .....	75
3.3.2. 外设库函数说明 .....	75
<b>3.4. CAN .....</b>	<b>88</b>
3.4.1. 外设寄存器说明 .....	88
3.4.2. 外设库函数说明 .....	89
<b>3.5. CRC .....</b>	<b>114</b>
3.5.1. 外设寄存器说明 .....	114
3.5.2. 外设库函数说明 .....	114
<b>3.6. CTC .....</b>	<b>121</b>
3.6.1. 外设寄存器说明 .....	122
3.6.2. 外设库函数说明 .....	122
<b>3.7. CMP .....</b>	<b>135</b>
3.7.1. 外设寄存器说明 .....	135
3.7.2. 外设库函数说明 .....	135

<b>3.8. DAC .....</b>	<b>141</b>
3.8.1. 外设寄存器说明 .....	142
3.8.2. 外设库函数说明 .....	142
<b>3.9. DBG .....</b>	<b>164</b>
3.9.1. 外设寄存器说明 .....	164
3.9.2. 外设库函数说明 .....	164
<b>3.10. DMA .....</b>	<b>170</b>
3.10.1. 外设寄存器说明 .....	170
3.10.2. 外设库函数说明 .....	170
<b>3.11. ENET .....</b>	<b>191</b>
3.11.1. 外设寄存器描述 .....	191
3.11.2. 外设库函数说明 .....	193
<b>3.12. EXMC .....</b>	<b>282</b>
3.12.1. 外设寄存器说明 .....	282
3.12.2. 外设库函数说明 .....	282
<b>3.13. EXTI.....</b>	<b>302</b>
3.13.1. 外设寄存器说明 .....	302
3.13.2. 外设库函数说明 .....	302
<b>3.14. FMC .....</b>	<b>310</b>
3.14.1. 外设寄存器说明 .....	310
3.14.2. 外设库函数说明 .....	310
<b>3.15. FWDGT.....</b>	<b>330</b>
3.15.1. 外设寄存器说明 .....	330
3.15.2. 外设库函数说明 .....	330
<b>3.16. GPIO .....</b>	<b>335</b>
3.16.1. 外设寄存器说明 .....	335
3.16.2. 外设库函数说明 .....	336
<b>3.17. SHRTIMER .....</b>	<b>354</b>
3.17.1. 外设寄存器说明 .....	354
3.17.2. 外设库函数说明 .....	356
<b>3.18. I2C .....</b>	<b>422</b>
3.18.1. 外设寄存器说明 .....	422
3.18.2. 外设库函数说明 .....	423
<b>3.19. MISC .....</b>	<b>482</b>
3.19.1. 外设寄存器说明 .....	482
3.19.2. 外设库函数说明 .....	483
<b>3.20. PMU.....</b>	<b>491</b>
3.20.1. 外设寄存器说明 .....	491
3.20.2. 外设库函数说明 .....	491

<b>3.21.</b>	<b>RCU.....</b>	<b>505</b>
3.21.1.	外设寄存器说明 .....	505
3.21.2.	外设库函数说明 .....	505
<b>3.22.</b>	<b>RTC .....</b>	<b>543</b>
3.22.1.	外设寄存器描述 .....	543
3.22.2.	外设库函数描述 .....	543
<b>3.23.</b>	<b>SDIO.....</b>	<b>551</b>
3.23.1.	外设寄存器说明 .....	551
3.23.2.	外设库函数说明 .....	552
<b>3.24.</b>	<b>SPI.....</b>	<b>583</b>
3.24.1.	外设寄存器说明 .....	583
3.24.2.	外设库函数说明 .....	584
<b>3.25.</b>	<b>SQPI .....</b>	<b>611</b>
3.25.1.	外设寄存器说明 .....	611
3.25.2.	外设库函数说明 .....	612
<b>3.26.</b>	<b>TIMER .....</b>	<b>618</b>
3.26.1.	外设寄存器说明 .....	618
3.26.2.	外设库函数说明 .....	618
<b>3.27.</b>	<b>TMU.....</b>	<b>676</b>
3.27.1.	外设寄存器说明 .....	676
3.27.2.	外设库函数说明 .....	676
<b>3.28.</b>	<b>USART.....</b>	<b>682</b>
3.28.1.	外设寄存器说明 .....	682
3.28.2.	外设库函数说明 .....	683
<b>3.29.</b>	<b>WWDGT .....</b>	<b>738</b>
3.29.1.	外设寄存器说明 .....	738
3.29.2.	外设库函数说明 .....	739
<b>4.</b>	<b>版本历史.....</b>	<b>744</b>

## 图索引

图 2-1. GD32E51x 固件库文件组织结构.....	33
图 2-2. 选择外设例程文件 .....	35
图 2-3. 拷贝外设例程文件 .....	36
图 2-4. 打开工程文件.....	36
图 2-5. 配置工程文件.....	37
图 2-6. 编译调试下载.....	37

## 表索引

表 1-1. 外设缩写 .....	31
表 2-1. 固件函数库文件描述 .....	38
表 3-1. 外设固件库函数描述格式 .....	39
表 3-2. ADC 寄存器 .....	39
表 3-3. ADC 库函数 .....	40
表 3-4. 函数 adc_deinit .....	41
表 3-5. 函数 adc_enable .....	42
表 3-6. 函数 adc_disable .....	42
表 3-7. 函数 adc_calibration_enable .....	43
表 3-8. 函数 adc_calibration_number .....	43
表 3-9. 函数 adc_dma_mode_enable .....	44
表 3-10. 函数 adc_dma_mode_disable .....	45
表 3-11. 函数 adc_tempsensor_vrefint_enable .....	45
表 3-12. 函数 adc_tempsensor_vrefint_disable .....	46
表 3-13. 函数 adc_discontinuous_mode_config .....	46
表 3-14. 函数 adc_mode_config .....	47
表 3-15. 函数 adc_special_function_config .....	48
表 3-16. 函数 adc_data_alignment_config .....	49
表 3-17. 函数 adc_channel_length_config .....	50
表 3-18. 函数 adc_regular_channel_config .....	50
表 3-19. 函数 adc_inserted_channel_config .....	51
表 3-20. 函数 adc_inserted_channel_offset_config .....	52
表 3-21. 函数 adc_channel_differential_mode_config .....	53
表 3-22. 函数 adc_external_trigger_config .....	54
表 3-23. 函数 adc_external_trigger_source_config .....	55
表 3-24. 函数 adc_software_trigger_enable .....	58
表 3-25. 函数 adc_regular_data_read .....	58
表 3-26. 函数 adc_inserted_data_read .....	59
表 3-27. 函数 adc_sync_mode_convert_value_read .....	60
表 3-28. 函数 adc_watchdog0_single_channel_enable .....	60
表 3-29. 函数 adc_watchdog0_group_channel_enable .....	61
表 3-30. 函数 adc_watchdog0_disable .....	62
表 3-31. 函数 adc_watchdog1_channel_config .....	62
表 3-32. 函数 adc_watchdog2_channel_config .....	63
表 3-33. 函数 adc_watchdog1_disable .....	64
表 3-34. 函数 adc_watchdog2_disable .....	64
表 3-35. 函数 adc_watchdog0_threshold_config .....	65
表 3-36. 函数 adc_watchdog1_threshold_config .....	65
表 3-37. 函数 adc_watchdog2_threshold_config .....	66
表 3-38. 函数 adc_resolution_config .....	67

表 3-39. 函数 <code>adc_oversample_mode_config</code> .....	67
表 3-40. 函数 <code>adc_oversample_mode_enable</code> .....	69
表 3-41. 函数 <code>adc_oversample_mode_disable</code> .....	70
表 3-42. 函数 <code>adc_flag_get</code> .....	70
表 3-43. 函数 <code>adc_flag_clear</code> .....	71
表 3-44. 函数 <code>adc_interrupt_enable</code> .....	72
表 3-45. 函数 <code>adc_interrupt_disable</code> .....	73
表 3-46. 函数 <code>adc_interrupt_flag_get</code> .....	73
表 3-47. 函数 <code>adc_interrupt_flag_clear</code> .....	74
表 3-48. BKP 寄存器.....	75
表 3-49. BKP 库函数.....	75
表 3-50. 枚举类型 <code>bkp_data_register_enum</code> .....	76
表 3-51. 函数 <code>bkp_deinit</code> .....	77
表 3-52. 函数 <code>bkp_write_data</code> .....	77
表 3-53. 函数 <code>bkp_read_data</code> .....	78
表 3-54. 函数 <code>bkp_rtc_calibration_output_enable</code> .....	79
表 3-55. 函数 <code>bkp_rtc_calibration_output_disable</code> .....	79
表 3-56. 函数 <code>bkp_rtc_signal_output_enable</code> .....	80
表 3-57. 函数 <code>bkp_rtc_signal_output_disable</code> .....	80
表 3-58. 函数 <code>bkp_rtc_output_select</code> .....	81
表 3-59. 函数 <code>bkp_rtc_clock_output_select</code> .....	81
表 3-60. 函数 <code>bkp_rtc_clock_calibration_direction</code> .....	82
表 3-61. 函数 <code>bkp_rtc_calibration_value_set</code> .....	82
表 3-62. 函数 <code>bkp_tamper_detection_enable</code> .....	83
表 3-63. 函数 <code>bkp_tamper_detection_disable</code> .....	83
表 3-64. 函数 <code>bkp_tamper_active_level_set</code> .....	84
表 3-65. 函数 <code>bkp_tamper_interrupt_enable</code> .....	84
表 3-66. 函数 <code>bkp_tamper_interrupt_disable</code> .....	85
表 3-67. 函数 <code>bkp_flag_get</code> .....	85
表 3-68. 函数 <code>bkp_flag_clear</code> .....	86
表 3-69. 函数 <code>bkp_interrupt_flag_get</code> .....	86
表 3-70. 函数 <code>bkp_interrupt_flag_clear</code> .....	87
表 3-71. CAN 寄存器.....	88
表 3-72. CAN-FD 寄存器.....	88
表 3-73. CAN 库函数.....	89
表 3-74. 结构体 <code>can_parameter_struct</code> .....	90
表 3-75. 结构体 <code>can_transmit_message_struct</code> .....	90
表 3-76. 结构体 <code>can_transmit_message_struct</code> (用于 CAN-FD).....	91
表 3-77. 结构体 <code>can_receive_message_struct</code> .....	91
表 3-78. 结构体 <code>can_receive_message_struct</code> (用于 CAN-FD).....	91
表 3-79. 结构体 <code>can_filter_parameter_struct</code> .....	91
表 3-80. 结构体 <code>can_fd_tdc_struct</code> (用于 CAN-FD).....	92
表 3-81. 结构体 <code>can_fdframe_struct</code> (用于 CAN-FD).....	92
表 3-82. 枚举 <code>can_interrupt_flag_enum</code> .....	92

表 3-83. 枚举 Enum can_flag_enum .....	93
表 3-84. 枚举 can_error_enum .....	94
表 3-85. 枚举 can_transmit_state_enum .....	94
表 3-86. 枚举 can_format_fifo_enum .....	94
表 3-87. 枚举 can_struct_type_enum .....	95
表 3-88. 函数 can_deinit .....	95
表 3-89. 函数 can_struct_para_init .....	95
表 3-90. 函数 can_init .....	96
表 3-91. 函数 can_filter_init .....	96
表 3-92. 函数 can_filter_mask_mode_init .....	97
表 3-93. 函数 can_monitor_mode_set .....	98
表 3-94. 函数 can_fd_init .....	98
表 3-95. 函数 can_fd_function_enable .....	99
表 3-96. 函数 can_fd_function_disable .....	100
表 3-97. 函数 can1_filter_start_bank .....	100
表 3-98. 函数 can_debug_freeze_enable .....	101
表 3-99. 函数 can_debug_freeze_disable .....	101
表 3-100. 函数 can_time_trigger_mode_enable .....	102
表 3-101. 函数 can_time_trigger_mode_disable .....	102
表 3-102. 函数 can_message_transmit .....	103
表 3-103. 函数 can_transmit_states .....	103
表 3-104. 函数 can_transmission_stop .....	104
表 3-105. 函数 can_message_receive .....	105
表 3-106. 函数 can_fifo_release .....	105
表 3-107. 函数 can_receive_message_length_get .....	106
表 3-108. 函数 can_working_mode_set .....	106
表 3-109. 函数 can_wakeup .....	107
表 3-110. 函数 can_error_get .....	108
表 3-111. 函数 can_receive_error_number_get .....	108
表 3-112. 函数 can_transmit_error_number_get .....	109
表 3-113. 函数 can_interrupt_enable .....	109
表 3-114. 函数 can_interrupt_disable .....	110
表 3-115. 函数 can_flag_get .....	111
表 3-116. 函数 can_flag_clear .....	112
表 3-117. 函数 can_interrupt_flag_get .....	112
表 3-118. 函数 can_interrupt_flag_clear .....	113
表 3-119. CRC 寄存器 .....	114
表 3-120. CRC 库函数 .....	114
表 3-121. 函数 crc_deinit .....	114
表 3-122. 函数 crc_data_register_reset .....	115
表 3-123. 函数 crc_reverse_output_data_enable .....	115
表 3-124. 函数 crc_reverse_output_data_disable .....	116
表 3-125. 函数 crc_input_data_reverse_config .....	116
表 3-126. 函数 crc_data_register_read .....	117

表 3-127. 函数 crc_free_data_register_read .....	118
表 3-128. 函数 crc_free_data_register_write .....	118
表 3-129. 函数 crc_init_data_register_write .....	119
表 3-130. 函数 crc_polynomial_size_set .....	119
表 3-131. 函数 crc_polynomial_set .....	120
表 3-132. 函数 crc_single_data_calculate .....	120
表 3-133. 函数 crc_block_data_calculate .....	121
表 3-134. CTC 寄存器 .....	122
表 3-135. CTC 库函数 .....	122
表 3-136. 函数 ctc_deinit .....	123
表 3-137. 函数 ctc_counter_enable .....	123
表 3-138. 函数 ctc_counter_disable .....	124
表 3-139. 函数 ctc_irc48m_trim_value_config .....	124
表 3-140. 函数 ctc_software_refsource_pulse_generate .....	125
表 3-141. 函数 ctc_hardware_trim_mode_config .....	125
表 3-142. 函数 ctc_refsource_polarity_config .....	126
表 3-143. 函数 ctc_refsource_signal_select .....	126
表 3-144. 函数 ctc_refsource_prescaler_config .....	127
表 3-145. 函数 ctc_clock_limit_value_config .....	128
表 3-146. 函数 ctc_counter_reload_value_config .....	128
表 3-147. 函数 ctc_counter_capture_value_read .....	129
表 3-148. 函数 ctc_counter_direction_read .....	129
表 3-149. 函数 ctc_counter_reload_value_read .....	130
表 3-150. 函数 ctc_irc48m_trim_value_read .....	130
表 3-151. 函数 ctc_flag_get .....	131
表 3-152. 函数 ctc_flag_clear .....	132
表 3-153. 函数 ctc_interrupt_enable .....	132
表 3-154. 函数 ctc_interrupt_disable .....	133
表 3-155. 函数 ctc_interrupt_flag_get .....	133
表 3-156. 函数 ctc_interrupt_flag_clear .....	134
表 3-157. CMP 寄存器 .....	135
表 3-158. CMP 库函数 .....	135
表 3-159. 枚举类型 cmp_enum .....	136
表 3-160. 函数 cmp_deinit .....	136
表 3-161. 函数 cmp_mode_init .....	136
表 3-162. 函数 cmp_output_init .....	138
表 3-163. 函数 cmp_blanking_init .....	139
表 3-164. 函数 cmp_enable .....	139
表 3-165. 函数 cmp_disable .....	140
表 3-166. 函数 cmp_lock_enable .....	140
表 3-167. 函数 cmp_output_level_get .....	141
表 3-168. DAC 寄存器 .....	142
表 3-169. DAC 库函数 .....	142
表 3-170. 函数 dac_deinit .....	143

表 3-170. 函数 dac_enable .....	144
表 3-170. 函数 dac_disable .....	144
表 3-170. 函数 dac_dma_enable .....	145
表 3-170. 函数 dac_dma_disable .....	145
表 3-170. 函数 dac_gpio_connect_config .....	146
表 3-170. 函数 dac_output_buffer_enable.....	147
表 3-170. 函数 dac_output_buffer_disable.....	147
表 3-170. 函数 dac_output_value_get .....	148
表 3-170. 函数 dac_data_set .....	148
表 3-170. 函数 dac_trigger_enable .....	149
表 3-170. 函数 dac_trigger_disable .....	150
表 3-170. 函数 dac_trigger_source_config .....	150
表 3-170. 函数 dac_software_trigger_enable.....	152
表 3-170. 函数 dac_wave_mode_config .....	152
表 3-170. 函数 dac_lfsr_noise_config .....	153
表 3-170. 函数 dac_triangle_noise_config .....	154
表 3-170. 函数 dac_concurrent_enable .....	154
表 3-170. 函数 dac_concurrent_disable .....	155
表 3-170. 函数 dac_concurrent_software_trigger_enable.....	155
表 3-170. 函数 dac_concurrent_output_buffer_enable.....	156
表 3-170. 函数 dac_concurrent_output_buffer_disable.....	156
表 3-170. 函数 dac_concurrent_data_set .....	157
表 3-170. 函数 dac_output_fifo_enable .....	158
表 3-170. 函数 dac_output_fifo_disable .....	158
表 3-170. 函数 dac_output_fifo_number_get.....	159
表 3-170. 函数 dac_flag_get.....	159
表 3-170. 函数 dac_flag_clear.....	160
表 3-170. 函数 dac_interrupt_enable.....	161
表 3-170. 函数 dac_interrupt_disable.....	162
表 3-170. 函数 dac_interrupt_flag_get .....	162
表 3-170. 函数 dac_interrupt_flag_clear .....	163
表 3-170. DBG 寄存器.....	164
表 3-171. DBG 库函数.....	164
表 3-172. 枚举类型 dbg_periph_enum.....	165
表 3-173. 函数 dbg_deinit.....	165
表 3-174. 函数 dbg_id_get.....	166
表 3-175. 函数 dbg_low_power_enable .....	166
表 3-176. 函数 dbg_low_power_disable .....	167
表 3-177. 函数 dbg_periph_enable .....	168
表 3-178. 函数 dbg_periph_disable .....	168
表 3-179. 函数 dbg_trace_pin_enable .....	169
表 3-180. 函数 dbg_trace_pin_disable .....	169
表 3-181. DMA 寄存器 .....	170
表 3-182. DMA 库函数 .....	170

表 3-183. 枚举类型 dma_channel_enum .....	171
表 3-184. 结构体 dma_parameter_struct .....	172
表 3-185. 函数 dma_deinit .....	172
表 3-186. 函数 dma_struct_para_init .....	173
表 3-187. 函数 dma_init .....	173
表 3-188. 函数 dma_circulation_enable .....	174
表 3-189. 函数 dma_circulation_disable .....	175
表 3-190. 函数 dma_memory_to_memory_enable .....	175
表 3-191. 函数 dma_memory_to_memory_disable .....	176
表 3-192. 函数 dma_channel_enable .....	177
表 3-193. 函数 dma_channel_disable .....	177
表 3-194. 函数 dma_periph_address_config .....	178
表 3-195. 函数 dma_memory_address_config .....	178
表 3-196. 函数 dma_transfer_number_config .....	179
表 3-197. 函数 dma_transfer_number_get .....	180
表 3-198. 函数 dma_priority_config .....	181
表 3-199. 函数 dma_memory_width_config .....	181
表 3-200. 函数 dma_periph_width_config .....	182
表 3-201. 函数 dma_memory_increase_enable .....	183
表 3-202. 函数 dma_memory_increase_disable .....	184
表 3-203. 函数 dma_periph_increase_enable .....	184
表 3-204. 函数 dma_periph_increase_disable .....	185
表 3-205. 函数 dma_transfer_direction_config .....	185
表 3-206. 函数 dma_flag_get .....	186
表 3-207. 函数 dma_flag_clear .....	187
表 3-208. 函数 dma_interrupt_enable .....	188
表 3-209. 函数 dma_interrupt_disable .....	188
表 3-210. 函数 dma_interrupt_flag_get .....	189
表 3-211. 函数 dma_interrupt_flag_clear .....	190
表 3-212. ENET 寄存器 .....	191
表 3-213. ENET 库函数 .....	193
表 3-214. 结构体 enet_initpara_struct .....	197
表 3-215. 结构体 enet_descriptors_struct .....	197
表 3-216. 结构体 enet_ptp_systime_struct .....	197
表 3-217. 枚举类型 enet_flag_enum .....	198
表 3-218. 枚举类型 enet_flag_clear_enum .....	199
表 3-219. 枚举类型 enet_int_enum .....	200
表 3-220. 枚举类型 enet_int_flag_enum .....	201
表 3-221. 枚举类型 enet_int_flag_clear_enum .....	203
表 3-222. 枚举类型 enet_desc_reg_enum .....	204
表 3-223. 枚举类型 enet_msc_counter_enum .....	204
表 3-224. 枚举类型 enet_option_enum .....	205
表 3-225. 枚举类型 enet_mediamode_enum .....	205
表 3-226. 枚举类型 enet_chksumconf_enum .....	206

表 3-227. 枚举类型 enet_frmrecept_enum .....	206
表 3-228. 枚举类型 enet_registers_type_enum .....	206
表 3-229. 枚举类型 enet_dmadirection_enum .....	206
表 3-230. 枚举类型 enet_phydirection_enum .....	207
表 3-231. 枚举类型 enet_regdirection_enum .....	207
表 3-232. 枚举类型 enet_macaddress_enum .....	207
表 3-233. 枚举类型 enet_descstate_enum .....	207
表 3-234. 枚举类型 enet_msc_preset_enum .....	208
表 3-235. 函数 enet_deinit .....	208
表 3-236. 函数 enet_initpara_config .....	208
表 3-237. 函数 enet_init .....	212
表 3-238. 函数 enet_software_reset .....	213
表 3-239. 函数 enet_rxframe_size_get .....	213
表 3-240. 函数 enet_descriptors_chain_init .....	214
表 3-241. 函数 enet_descriptors_ring_init .....	215
表 3-242. 函数 enet_frame_receive .....	215
表 3-243. 函数 enet_frame_transmit .....	216
表 3-244. 函数 enet_transmit_checksum_config .....	216
表 3-245. 函数 enet_enable .....	217
表 3-246. 函数 enet_disable .....	218
表 3-247. 函数 enet_mac_address_set .....	218
表 3-248. 函数 enet_mac_address_get .....	219
表 3-249. 函数 enet_flag_get .....	220
表 3-250. 函数 enet_flag_clear .....	222
表 3-251. 函数 enet_interrupt_enable .....	223
表 3-252. 函数 enet_interrupt_disable .....	225
表 3-253. 函数 enet_interrupt_flag_get .....	226
表 3-254. 函数 enet_interrupt_flag_clear .....	228
表 3-255. 函数 enet_tx_enable .....	230
表 3-256. 函数 enet_tx_disable .....	230
表 3-257. 函数 enet_rx_enable .....	230
表 3-258. 函数 enet_rx_disable .....	231
表 3-259. 函数 enet_registers_get .....	231
表 3-260. 函数 enet_debug_status_get .....	232
表 3-261. 函数 enet_address_filter_enable .....	233
表 3-262. 函数 enet_address_filter_disable .....	234
表 3-263. 函数 enet_address_filter_config .....	235
表 3-264. 函数 enet_phy_config .....	236
表 3-265. 函数 enet_phy_write_read .....	236
表 3-266. 函数 enet_phyloopback_enable .....	237
表 3-267. 函数 enet_phyloopback_disable .....	237
表 3-268. 函数 enet_forward_feature_enable .....	238
表 3-269. 函数 enet_forward_feature_disable .....	239
表 3-270. 函数 enet_fliter_feature_enable .....	239

表 3-271. 函数 enet_fliter_feature_disable .....	240
表 3-272. 函数 enet_pauseframe_generate .....	241
表 3-273. 函数 enet_pauseframe_detect_config .....	241
表 3-274. 函数 enet_pauseframe_config .....	242
表 3-275. 函数 enet_flowcontrol_threshold_config .....	243
表 3-276. 函数 enet_flowcontrol_feature_enable .....	244
表 3-277. 函数 enet_flowcontrol_feature_disable .....	245
表 3-278. 函数 enet_dmaprocess_state_get .....	246
表 3-279. 函数 enet_dmaprocess_resume .....	247
表 3-280. 函数 enet_rxprocess_check_recovery .....	247
表 3-281. 函数 enet_txfifo_flush .....	248
表 3-282. 函数 enet_current_desc_address_get .....	248
表 3-283. 函数 enet_desc_information_get .....	249
表 3-284. 函数 enet_missed_frame_counter_get .....	250
表 3-285. 函数 enet_desc_flag_get .....	250
表 3-286. 函数 enet_desc_flag_set .....	253
表 3-287. 函数 enet_desc_flag_clear .....	254
表 3-288. 函数 enet_rx_desc_immediate_receive_complete_interrupt .....	255
表 3-289. 函数 enet_rx_desc_delay_receive_complete_interrupt .....	255
表 3-290. 函数 enet_rxframe_drop .....	256
表 3-291. 函数 enet_dma_feature_enable .....	256
表 3-292. 函数 enet_dma_feature_disable .....	257
表 3-293. 函数 enet_rx_desc_enhanced_status_get .....	257
表 3-294. 函数 enet_desc_select_enhanced_mode .....	258
表 3-295. 函数 enet_ptp_enhanced_descriptors_chain_init .....	259
表 3-296. 函数 enet_ptp_enhanced_descriptors_ring_init .....	259
表 3-297. 函数 enet_ptpframe_receive_enhanced_mode .....	260
表 3-298. 函数 enet_ptpframe_transmit_enhanced_mode .....	261
表 3-299. 函数 enet_desc_select_normal_mode .....	261
表 3-300. 函数 enet_ptp_normal_descriptors_chain_init .....	262
表 3-301. 函数 enet_ptp_normal_descriptors_ring_init .....	263
表 3-302. 函数 enet_ptpframe_receive_normal_mode .....	263
表 3-303. 函数 enet_ptpframe_transmit_normal_mode .....	264
表 3-304. 函数 enet_wum_filter_register_pointer_reset .....	265
表 3-305. 函数 enet_wum_filter_config .....	265
表 3-306. 函数 enet_wum_feature_enable .....	266
表 3-307. 函数 enet_wum_feature_disable .....	266
表 3-308. 函数 enet_msc_counters_reset .....	267
表 3-309. 函数 enet_msc_feature_enable .....	267
表 3-310. 函数 enet_msc_feature_disable .....	268
表 3-311. 函数 enet_msc_counters_preset_config .....	269
表 3-312. 函数 enet_msc_counters_get .....	269
表 3-313. 函数 enet_ptp_subsecond_2_nanosecond .....	270
表 3-314. 函数 enet_ptp_nanosecond_2_subsecond .....	271

表 3-315. 函数 enet_ptp_feature_enable .....	271
表 3-316. 函数 enet_ptp_feature_disable .....	272
表 3-317. 函数 enet_ptp_timestamp_function_config .....	273
表 3-318. 函数 enet_ptp_subsecond_increment_config .....	274
表 3-319. 函数 enet_ptp_timestamp_addend_config .....	275
表 3-320. 函数 enet_ptp_timestamp_update_config .....	275
表 3-321. 函数 enet_ptp_expected_time_config .....	276
表 3-322. 函数 enet_ptp_system_time_get .....	276
表 3-323. 函数 enet_ptp_pps_output_frequency_config .....	277
表 3-324. 函数 enet_ptp_start .....	278
表 3-325. 函数 enet_ptp_finecorrection_adjfreq .....	279
表 3-326. 函数 enet_ptp_coarsecorrection_systime_update .....	279
表 3-327. 函数 enet_ptp_finecorrection_settime .....	280
表 3-328. 函数 enet_ptp_flag_get .....	281
表 3-329. 函数 enet_initpara_reset .....	281
表 3-330. EXMC 寄存器 .....	282
表 3-331. EXMC 库函数 .....	282
表 3-332. 结构体 exmc_norsram_timing_parameter_struct .....	283
表 3-333. 结构体 exmc_norsram_parameter_struct .....	283
表 3-334. 结构体 exmc_nand_pccard_timing_parameter_struct .....	284
表 3-335. 结构体 exmc_nand_parameter_struct .....	284
表 3-336. 结构体 exmc_pccard_parameter_struct .....	284
表 3-337. 函数 exmc_norsram_deinit .....	285
表 3-338. 函数 exmc_norsram_struct_para_init .....	285
表 3-339. 函数 exmc_norsram_init .....	286
表 3-340. 函数 exmc_norsram_enable .....	287
表 3-341. 函数 exmc_norsram_disable .....	288
表 3-342. 函数 exmc_norsram_page_size_config .....	288
表 3-343. 函数 exmc_nand_deinit .....	289
表 3-344. 函数 exmc_nand_struct_para_init .....	290
表 3-345. 函数 exmc_nand_init .....	290
表 3-346. 函数 exmc_nand_enable .....	291
表 3-347. 函数 exmc_nand_disable .....	292
表 3-348. 函数 exmc_nand_ecc_config .....	292
表 3-349. 函数 exmc_ecc_get .....	293
表 3-350. 函数 exmc_pccard_deinit .....	293
表 3-351. 函数 exmc_pccard_struct_para_init .....	294
表 3-352. 函数 exmc_pccard_init .....	294
表 3-353. 函数 exmc_pccard_enable .....	295
表 3-354. 函数 exmc_pccard_disable .....	296
表 3-355. 函数 exmc_interrupt_enable .....	296
表 3-356. 函数 exmc_interrupt_disable .....	297
表 3-357. 函数 exmc_flag_get .....	298
表 3-358. 函数 exmc_flag_clear .....	299

表 3-359. 函数 exmc_interrupt_flag_get .....	300
表 3-360. 函数 exmc_interrupt_flag_clear .....	301
表 3-361. EXTI 寄存器 .....	302
表 3-362. EXTI 库函数 .....	302
表 3-363. 枚举类型 exti_line_enum .....	302
表 3-364. 枚举类型 exti_mode_enum .....	303
表 3-365. 枚举类型 exti_trig_type_enum .....	303
表 3-366. 函数 exti_deinit .....	303
表 3-367. 函数 exti_init .....	304
表 3-368. 函数 exti_interrupt_enable .....	305
表 3-369. 函数 exti_interrupt_disable .....	305
表 3-370. 函数 exti_event_enable .....	306
表 3-371. 函数 exti_event_disable .....	306
表 3-372. 函数 exti_software_interrupt_enable .....	307
表 3-373. 函数 exti_software_interrupt_disable .....	307
表 3-374. 函数 exti_flag_get .....	308
表 3-375. 函数 exti_flag_clear .....	308
表 3-376. 函数 exti_interrupt_flag_get .....	309
表 3-377. 函数 exti_interrupt_flag_clear .....	309
表 3-378. FMC 寄存器 .....	310
表 3-379. FMC 固件库函数 .....	310
表 3-380. 枚举类型 fmc_state_enum .....	311
表 3-381. 枚举类型 fmc_flag_enum .....	311
表 3-382. 枚举类型 fmc_interrupt_flag_enum .....	312
表 3-383. 枚举类型 fmc_interrupt_enum .....	312
表 3-384. 函数 fmc_unlock .....	313
表 3-385. 函数 fmc_lock .....	313
表 3-386. 函数 fmc_wscnt_set .....	314
表 3-387. 函数 fmc_prefetch_enable .....	314
表 3-388. 函数 fmc_prefetch_disable .....	315
表 3-389. 函数 fmc_ibus_enable .....	315
表 3-390. 函数 fmc_ibus_disable .....	316
表 3-391. 函数 fmc_dbus_enable .....	316
表 3-392. 函数 fmc_dbus_disable .....	317
表 3-393. 函数 fmc_ibus_reset .....	317
表 3-394. 函数 fmc_dbus_reset .....	318
表 3-395. 函数 fmc_page_erase .....	318
表 3-396. 函数 fmc_mass_erase .....	319
表 3-397. 函数 fmc_doubleword_program .....	319
表 3-398. 函数 ob_unlock .....	320
表 3-399. 函数 ob_lock .....	320
表 3-400. 函数 ob_erase .....	321
表 3-401. 函数 ob_write_protection_enable .....	321
表 3-402. 函数 ob_security_protection_config .....	322

表 3-403. 函数 ob_user_write .....	323
表 3-404. 函数 ob_data_program .....	324
表 3-405. 函数 ob_user_get .....	324
表 3-406. 函数 ob_data_get.....	325
表 3-407. 函数 ob_write_protection_get.....	325
表 3-408. 函数 ob_security_protection_flag_get .....	326
表 3-409. 函数 fmc_ecc_address_get .....	326
表 3-410. 函数 fmc_flag_get.....	327
表 3-411. 函数 fmc_flag_clear.....	327
表 3-412. 函数 fmc_interrupt_enable .....	328
表 3-413. 函数 fmc_interrupt_disable .....	328
表 3-414. 函数 fmc_interrupt_flag_get.....	329
表 3-415. 函数 fmc_interrupt_flag_clear.....	329
表 3-416. FWDGT 寄存器.....	330
表 3-417. FWDGT 库函数.....	330
表 3-418. 函数 fwdgt_write_enable.....	330
表 3-419. 函数 fwdgt_write_disable.....	331
表 3-420. 函数 fwdgt_enable.....	331
表 3-421. 函数 fwdgt_prescaler_value_config.....	332
表 3-422. 函数 fwdgt_reload_value_config .....	332
表 3-423. 函数 fwdgt_config .....	333
表 3-424. 函数 fwdgt_counter_reload .....	334
表 3-425. 函数 fwdgt_flag_get .....	334
表 3-426. GPIO 寄存器.....	335
表 3-427. GPIO 库函数.....	336
表 3-428. 函数 gpio_deinit.....	337
表 3-429. 函数 gpio_afio_deinit.....	337
表 3-430. 函数 gpio_init .....	338
表 3-431. 函数 gpio_bit_set.....	339
表 3-432. 函数 gpio_bit_reset.....	339
表 3-433. 函数 gpio_bit_write .....	340
表 3-434. 函数 gpio_port_write .....	341
表 3-435. 函数 gpio_input_bit_get .....	341
表 3-436. 函数 gpio_input_port_get .....	342
表 3-437. 函数 gpio_output_bit_get.....	342
表 3-438. 函数 gpio_output_port_get.....	343
表 3-439. 函数 gpio_pin_remap_config .....	343
表 3-440. 函数 gpio_afio_port_config .....	345
表 3-441. 函数 gpio_ethernet_phy_select .....	350
表 3-442. 函数 gpio_exti_source_select.....	350
表 3-443. 函数 gpio_event_output_config.....	351
表 3-444. 函数 gpio_event_output_enable .....	352
表 3-445. 函数 gpio_event_output_disable .....	352
表 3-446. 函数 gpio_pin_lock .....	353

表 3-447. 函数 gpio_compensation_config.....	353
表 3-448. 函数 gpio_compensation_flag_get.....	354
表 3-449. SHRTIMER 寄存器 .....	355
表 3-450. SHRTIMER 库函数 .....	356
表 3-451. 结构体 shrtimer_baseinit_parameter_struct.....	359
表 3-452. 结构体 shrtimer_timerinit_parameter_struct.....	359
表 3-453. 结构体 shrtimer_timercfg_parameter_struct.....	359
表 3-454. 结构体 shrtimer_comparecfg_parameter_struct .....	360
表 3-455. 结构体 shrtimer_exevfilter_parameter_struct .....	360
表 3-456. 结构体 shrtimer_deadtimecfg_parameter_struct .....	360
表 3-457. 结构体 shrtimer_carriersignalcfg_parameter_struct .....	360
表 3-458. 结构体 shrtimer_synccfg_parameter_struct.....	361
表 3-459. 结构体 shrtimer_bunchmode_parameter_struct .....	361
表 3-460. 结构体 shrtimer_exeventcfg_parameter_struct .....	361
表 3-461. 结构体 shrtimer_faultcfg_parameter_struct .....	361
表 3-462. 结构体 shrtimer_adctrigcfg_parameter_struct.....	362
表 3-463. 结构体 shrtimer_channel_outputcfg_parameter_struct .....	362
表 3-464. 函数 shrtimer_deinit.....	362
表 3-465. 函数 shrtimer_dll_calibration_start.....	363
表 3-466. 函数 shrtimer_baseinit_struct_para_init.....	363
表 3-467. 函数 shrtimer_timers_base_init.....	364
表 3-468. 函数 shrtimer_timers_counter_enable .....	365
表 3-469. 函数 shrtimer_timers_counter_enable .....	366
表 3-470. 函数 shrtimer_timers_update_event_enable.....	366
表 3-471. 函数 shrtimer_timers_update_event_enable.....	367
表 3-472. 函数 shrtimer_software_update.....	368
表 3-473. 函数 shrtimer_software_counter_reset.....	368
表 3-474. 函数 shrtimer_timerinit_struct_para_init .....	369
表 3-475. 函数 shrtimer_timers_waveform_init.....	369
表 3-476. 函数 shrtimer_timercfg_struct_para_init .....	371
表 3-477. 函数 shrtimer_slavetimer_waveform_config.....	371
表 3-478. 函数 shrtimer_comparecfg_struct_para_init.....	372
表 3-479. 函数 shrtimer_slavetimer_waveform_compare_config.....	373
表 3-480. 函数 shrtimer_channel_outputcfg_struct_para_init .....	374
表 3-481. 函数 shrtimer_slavetimer_waveform_channel_config .....	374
表 3-482. 函数 shrtimer_slavetimer_waveform_channel_software_request .....	376
表 3-483. 函数 shrtimer_slavetimer_waveform_channel_output_level_get.....	377
表 3-484. 函数 shrtimer_slavetimer_waveform_channel_state_get .....	377
表 3-485. 函数 shrtimer_channel_outputcfg_struct_para_init .....	378
表 3-486. 函数 shrtimer_slavetimer_waveform_channel_software_request .....	379
表 3-487. 函数 shrtimer_channel_outputcfg_struct_para_init .....	380
表 3-488. 函数 shrtimer_slavetimer_carriersignal_config.....	381
表 3-489. 函数 shrtimer_output_channel_enable .....	381
表 3-490. 函数 shrtimer_output_channel_disable .....	382

表 3-491. 函数 shrtimer_slavetimer_waveform_compare_config .....	383
表 3-492. 函数 shrtimer_slavetimer_compare_value_get .....	384
表 3-493. 函数 shrtimer_mastertimer_compare_value_config .....	384
表 3-494. 函数 shrtimer_slavetimer_compare_value_get .....	385
表 3-495. 函数 shrtimer_timers_counter_value_config .....	386
表 3-496. 函数 shrtimer_timers_counter_value_get .....	386
表 3-497. 函数 shrtimer_timers_autoreload_value_config .....	387
表 3-498. 函数 shrtimer_timers_autoreload_value_get .....	388
表 3-499. 函数 shrtimer_timers_repetition_value_config .....	389
表 3-500. 函数 shrtimer_timers_repetition_value_get .....	389
表 3-501. 函数 shrtimer_exevcfg_struct_para_init .....	390
表 3-502. 函数 shrtimer_slavetimer_exevcfg_filtering_config .....	391
表 3-503. 函数 shrtimer_exevcfg_struct_para_init .....	392
表 3-504. 函数 shrtimer_exevcfg_config .....	392
表 3-505. 函数 shrtimer_exevcfg_prescaler .....	393
表 3-506. 函数 shrtimer_synccfg_struct_para_init .....	394
表 3-507. 函数 shrtimer_synchronization_config .....	394
表 3-508. 函数 shrtimer_faultcfg_struct_para_init .....	395
表 3-509. 函数 shrtimer_fault_config .....	396
表 3-510. 函数 shrtimer_fault_prescaler_config .....	397
表 3-511. 函数 shrtimer_fault_input_enable .....	397
表 3-512. 函数 shrtimer_fault_input_disable .....	398
表 3-513. 函数 shrtimer_timers_dma_enable .....	398
表 3-514. 函数 shrtimer_timers_dma_disable .....	400
表 3-515. 函数 shrtimer_dmamode_config .....	401
表 3-516. 函数 shrtimer_bunchmode_struct_para_init .....	402
表 3-517. 函数 shrtimer_bunchmode_config .....	403
表 3-518. 函数 shrtimer_bunchmode_enable .....	404
表 3-519. 函数 shrtimer_bunchmode_disable .....	404
表 3-520. 函数 shrtimer_bunchmode_flag_get .....	405
表 3-521. 函数 shrtimer_bunchmode_software_start .....	405
表 3-522. 函数 shrtimer_slavetimer_capture_config .....	406
表 3-523. 函数 shrtimer_slavetimer_capture_software .....	407
表 3-524. 函数 shrtimer_slavetimer_capture_value_read .....	408
表 3-525. 函数 shrtimer_adctrigcfg_struct_para_init .....	409
表 3-526. 函数 shrtimer_adc_trigger_config .....	409
表 3-527. 函数 shrtimer_timers_flag_get .....	410
表 3-528. 函数 shrtimer_timers_flag_clear .....	411
表 3-529. 函数 shrtimer_common_flag_get .....	413
表 3-530. 函数 shrtimer_common_flag_clear .....	413
表 3-531. 函数 shrtimer_timers_interrupt_enable .....	414
表 3-532. 函数 shrtimer_timers_interrupt_disable .....	415
表 3-533. 函数 shrtimer_timers_interrupt_flag_get .....	417
表 3-534. 函数 shrtimer_timers_interrupt_flag_clear .....	418

表 3-535. 函数 shrtimer_common_interrupt_enable .....	419
表 3-536. 函数 shrtimer_common_interrupt_disable .....	420
表 3-537. 函数 shrtimer_common_interrupt_flag_get.....	420
表 3-538. 函数 shrtimer_common_interrupt_flag_clear .....	421
表 3-539. I2C 寄存器 .....	422
表 3-540. I2C 库函数 .....	423
表 3-541. 枚举类型 i2c_flag_enum .....	425
表 3-542. 枚举类型 i2c_interrupt_enum .....	426
表 3-543. 枚举类型 i2c_interrupt_flag_enum .....	426
表 3-544. 枚举类型 i2c2_interrupt_flag_enum .....	427
表 3-545. 函数 i2c_deinit .....	428
表 3-546. 函数 i2c_enable.....	428
表 3-547. 函数 i2c_disable.....	429
表 3-548. 函数 i2c_start_on_bus.....	429
表 3-549. 函数 i2c_stop_on_bus .....	430
表 3-550. 函数 i2c_slave_response_to_gcall_enable.....	430
表 3-551. 函数 i2c_slave_response_to_gcall_disable.....	431
表 3-552. 函数 i2c_stretch_scl_low_enable .....	431
表 3-553. 函数 i2c_stretch_scl_low_disable .....	432
表 3-554. 函数 i2c_data_transmit.....	432
表 3-555. 函数 i2c_data_receive.....	433
表 3-556. 函数 i2c_pec_transfer.....	433
表 3-557. 函数 i2c_pec_enable .....	434
表 3-558. 函数 i2c_pec_disable .....	434
表 3-559. 函数 i2c_pec_value_get .....	435
表 3-560. 函数 i2c_clock_config .....	435
表 3-561. 函数 i2c_mode_addr_config.....	436
表 3-562. 函数 i2c_smbus_type_config .....	437
表 3-563. 函数 i2c_ack_config.....	437
表 3-564. 函数 i2c_ackpos_config .....	438
表 3-565. 函数 i2c_master_addressing.....	438
表 3-566. 函数 i2c_dualaddr_enable .....	439
表 3-567. 函数 i2c_dualaddr_disable .....	440
表 3-568. 函数 i2c_dma_config .....	440
表 3-569. 函数 i2c_dma_last_transfer_config .....	441
表 3-570. 函数 i2c_software_reset_config .....	441
表 3-571. 函数 i2c_smbus_alert_config.....	442
表 3-572. 函数 i2c_smbus_arp_config.....	443
表 3-573. 函数 i2c_sam_enable.....	443
表 3-574. 函数 i2c_sam_disable.....	444
表 3-575. 函数 i2c_sam_timeout_enable .....	444
表 3-576. 函数 i2c_sam_timeout_disable .....	445
表 3-577. 函数 i2c_start_early_termination_mode_config .....	445
表 3-578. 函数 i2c_timeout_calculation_enable .....	446

表 3-579. 函数 i2c_timeout_calculation_disable .....	446
表 3-580. 函数 i2c_record_received_slave_address_enable .....	447
表 3-581. 函数 i2c_record_received_slave_address_disable .....	447
表 3-582. 函数 i2c_address_bit_compare_config .....	448
表 3-583. 函数 i2c_status_clear_enable .....	449
表 3-584. 函数 i2c_status_clear_disable .....	449
表 3-585. 函数 i2c_start_early_termination_mode_config .....	450
表 3-586. 函数 i2c_flag_get .....	450
表 3-587. 函数 i2c_flag_clear .....	451
表 3-588. 函数 i2c_interrupt_enable .....	452
表 3-589. 函数 i2c_interrupt_disable .....	452
表 3-590. 函数 i2c_interrupt_flag_get .....	453
表 3-591. 函数 i2c_interrupt_flag_clear .....	454
表 3-592. 函数 i2c_timing_config .....	455
表 3-593. 函数 i2c_digital_noise_filter_config .....	455
表 3-594. 函数 i2c_analog_noise_filter_enable .....	456
表 3-595. 函数 i2c_analog_noise_filter_disable .....	457
表 3-596. 函数 i2c_wakeup_from_deepsleep_enable .....	457
表 3-597. 函数 i2c_wakeup_from_deepsleep_disable .....	458
表 3-598. 函数 i2c_master_clock_config .....	458
表 3-599. 函数 i2c2_master_transfer_direction_config .....	459
表 3-600. 函数 i2c_address10_header_enable .....	460
表 3-601. 函数 i2c_address10_header_disable .....	460
表 3-602. 函数 i2c_address10_enable .....	461
表 3-603. 函数 i2c_address10_disable .....	461
表 3-604. 函数 i2c_automatic_end_enable .....	462
表 3-605. 函数 i2c_automatic_end_disable .....	462
表 3-606. 函数 i2c_address_config .....	463
表 3-607. 函数 i2c_address_disable .....	463
表 3-608. 函数 i2c_second_address_config .....	464
表 3-609. 函数 i2c_second_address_disable .....	465
表 3-610. 函数 i2c_receivied_address_get .....	465
表 3-611. 函数 i2c_slave_byte_control_enable .....	466
表 3-612. 函数 i2c_slave_byte_control_disable .....	466
表 3-613. 函数 i2c_nack_enable .....	467
表 3-614. 函数 i2c_nack_disable .....	467
表 3-615. 函数 i2c_reload_enable .....	468
表 3-616. 函数 i2c_reload_disable .....	468
表 3-617. 函数 i2c_transfer_byte_number_config .....	469
表 3-618. 函数 i2c2_dma_enable .....	470
表 3-619. 函数 i2c2_dma_disable .....	470
表 3-620. 函数 i2c_smbus_alert_enable .....	471
表 3-621. 函数 i2c_smbus_alert_disable .....	471
表 3-622. 函数 i2c_smbus_default_addr_enable .....	472

表 3-623. 函数 i2c_smbus_default_addr_disable .....	472
表 3-624. 函数 i2c_smbus_host_addr_enable.....	473
表 3-625. 函数 i2c_smbus_host_addr_disable.....	473
表 3-626. 函数 i2c_extented_clock_timeout_enable .....	474
表 3-627. 函数 i2c_extented_clock_timeout_disable .....	474
表 3-628. 函数 i2c_clock_timeout_enable .....	475
表 3-629. 函数 i2c_clock_timeout_disable.....	475
表 3-630. 函数 i2c_bus_timeout_b_config .....	476
表 3-631. 函数 i2c_bus_timeout_a_config .....	476
表 3-632. 函数 i2c_idle_clock_timeout_config.....	477
表 3-633. 函数 i2c2_flag_get.....	477
表 3-634. 函数 i2c2_flag_clear.....	478
表 3-635. 函数 i2c2_interrupt_enable.....	479
表 3-636. 函数 i2c2_interrupt_disable.....	480
表 3-637. 函数 i2c2_interrupt_flag_get .....	481
表 3-638. 函数 i2c2_interrupt_flag_clear .....	481
表 3-639. NVIC 寄存器.....	482
表 3-640. SysTick 寄存器.....	483
表 3-641. 枚举类型 IRQn_Type.....	483
表 3-642. MISC 库函数.....	485
表 3-643. 函数 nvic_priority_group_set.....	486
表 3-644. 函数 nvic_irq_enable .....	487
表 3-645. 函数 nvic_irq_disable .....	487
表 3-646. 函数 nvic_system_reset .....	488
表 3-647. 函数 nvic_vector_table_set .....	488
表 3-648. 函数 system_lowpower_set.....	489
表 3-649. 函数 system_lowpower_reset.....	489
表 3-650. 函数 systick_clksource_set.....	490
表 3-651. PMU 寄存器.....	491
表 3-652. PMU 库函数.....	491
表 3-653. 函数 pmu_deinit.....	492
表 3-654. 函数 pmu_lvd_select.....	492
表 3-655. 函数 pmu_lvd_disable .....	493
表 3-656. 函数 pmu_highdriver_switch_select.....	493
表 3-657. 函数 pmu_highdriver_mode_enable .....	494
表 3-658. 函数 pmu_highdriver_mode_disable .....	494
表 3-659. 函数 pmu_lowdriver_mode_enable.....	495
表 3-660. 函数 pmu_lowdriver_mode_disable.....	495
表 3-661. 函数 pmu_lowpower_driver_config .....	496
表 3-662. 函数 pmu_normalpower_driver_config.....	496
表 3-663. 函数 pmu_to_sleepmode .....	497
表 3-664. 函数 pmu_to_deepsleepmode .....	497
表 3-665. 函数 pmu_to_deepsleepmode_1 .....	498
表 3-666. 函数 pmu_to_deepsleepmode_2 .....	499

表 3-667. 函数 pmu_to_standbymode.....	500
表 3-668. 函数 pmu_backup_write_enable.....	501
表 3-669. 函数 pmu_backup_write_disable .....	501
表 3-670. 函数 pmu_wakeup_pin_enable .....	501
表 3-671. 函数 pmu_wakeup_pin_disable .....	502
表 3-672. 函数 pmu_flag_get .....	503
表 3-673. 函数 pmu_flag_clear .....	504
表 3-674. RCU 寄存器.....	505
表 3-675. RCU 库函数.....	506
表 3-676. 枚举类型 rcu_periph_enum.....	507
表 3-677. 枚举类型 rcu_periph_sleep_enum.....	509
表 3-678. 枚举类型 rcu_periph_reset_enum .....	509
表 3-679. 枚举类型 rcu_flag_enum .....	510
表 3-680. 枚举类型 rcu_int_flag_enum.....	511
表 3-681. 枚举类型 rcu_int_flag_clear_enum.....	511
表 3-682. 枚举类型 rcu_int_enum .....	512
表 3-683. 枚举类型 rcu_osci_type_enum.....	512
表 3-684. 枚举类型 rcu_clock_freq_enum .....	512
表 3-685. 函数 rcu_deinit.....	512
表 3-686. 函数 rcu_periph_clock_enable .....	513
表 3-687. 函数 rcu_periph_clock_disable .....	513
表 3-688. 函数 rcu_periph_clock_sleep_enable.....	514
表 3-689. 函数 rcu_periph_clock_sleep_disable.....	514
表 3-690. 函数 rcu_periph_reset_enable .....	515
表 3-691. 函数 rcu_periph_reset_disable .....	515
表 3-692. 函数 rcu_bkp_reset_enable .....	516
表 3-693. 函数 rcu_bkp_reset_disable .....	516
表 3-694. 函数 rcu_system_clock_source_config .....	517
表 3-695. 函数 rcu_system_clock_source_get.....	517
表 3-696. 函数 rcu_ahb_clock_config .....	518
表 3-697. 函数 rcu_apb1_clock_config .....	518
表 3-698. 函数 rcu_apb2_clock_config .....	519
表 3-699. 函数 rcu_ckout0_config .....	520
表 3-700. 函数 rcu_pll_config .....	521
表 3-701. 函数 rcu_pllpresel_config .....	522
表 3-702. 函数 rcu_predv0_config .....	522
表 3-703. 函数 rcu_predv0_config .....	523
表 3-704. 函数 rcu_predv1_config .....	523
表 3-705. 函数 rcu_pll1_config .....	524
表 3-706. 函数 rcu_pll2_config .....	524
表 3-707. 函数 rcu_pllusbpresel_config .....	525
表 3-708. 函数 rcu_pllusbpredv_config .....	525
表 3-709. 函数 rcu_pllusb_config .....	526
表 3-710. 函数 rcu_adc_clock_config .....	527

表 3-711. 函数 rcu_usb_clock_config .....	528
表 3-712. 函数 rcu_rtc_clock_config .....	528
表 3-713. 函数 rcu_shrtimer_clock_config .....	529
表 3-714. 函数 rcu_usart5_clock_config .....	530
表 3-715. 函数 rcu_i2c2_clock_config .....	530
表 3-716. 函数 rcu_i2s1_clock_config .....	531
表 3-717. 函数 rcu_i2s2_clock_config .....	532
表 3-718. 函数 rcu_ck48m_clock_config .....	532
表 3-719. 函数 rcu_usbhssel_config .....	533
表 3-720. 函数 rcu_usbdrv_config .....	533
表 3-721. 函数 rcu_lxtal_drive_capability_config .....	534
表 3-722. 函数 rcu_osci_stab_wait .....	535
表 3-723. 函数 rcu_osci_on .....	535
表 3-724. 函数 rcu_osci_off .....	536
表 3-725. 函数 rcu_irc8m_adjust_value_set .....	536
表 3-726. 函数 rcu_osci_bypass_mode_enable .....	537
表 3-727. 函数 rcu_osci_bypass_mode_disable .....	537
表 3-728. 函数 rcu_hxtal_clock_monitor_enable .....	538
表 3-729. 函数 rcu_hxtal_clock_monitor_disable .....	538
表 3-730. 函数 rcu_deepsleep_voltage_set .....	539
表 3-731. 函数 rcu_clock_freq_get .....	539
表 3-732. 函数 rcu_flag_get .....	540
表 3-733. 函数 rcu_all_reset_flag_clear .....	540
表 3-734. 函数 rcu_interrupt_flag_get .....	541
表 3-735. 函数 rcu_interrupt_flag_clear .....	541
表 3-736. 函数 rcu_interrupt_enable .....	542
表 3-737. 函数 rcu_interrupt_disable .....	542
表 3-738. RTC 寄存器 .....	543
表 3-739. RTC 库函数 .....	543
表 3-740. 函数 rtc_configuration_mode_enter .....	544
表 3-741. 函数 rtc_configuration_mode_exit .....	544
表 3-742. 函数 rtc_lwoff_wait .....	545
表 3-743. 函数 rtc_register_sync_wait .....	545
表 3-744. 函数 rtc_counter_get .....	546
表 3-745. Function rtc_counter_set .....	546
表 3-746. 函数 rtc_prescaler_set .....	547
表 3-747. 函数 rtc_alarm_config .....	547
表 3-748. 函数 rtc_divider_get .....	548
表 3-749. 函数 rtc_interrupt_enable .....	549
表 3-750. 函数 rtc_interrupt_disable .....	549
表 3-751. 函数 rtc_flag_get .....	550
表 3-752. 函数 rtc_flag_clear .....	551
表 3-753. SDIO 寄存器 .....	551
表 3-754. SDIO 库函数 .....	552

表 3-755. 函数 sdio_deinit .....	553
表 3-756. 函数 sdio_clock_config .....	554
表 3-757. 函数 sdio_hardware_clock_enable .....	555
表 3-758. 函数 sdio_hardware_clock_disable .....	555
表 3-759. 函数 sdio_bus_mode_set .....	556
表 3-760. 函数 sdio_power_state_set .....	556
表 3-761. 函数 sdio_power_state_get .....	557
表 3-762. 函数 sdio_clock_enable .....	557
表 3-763. 函数 sdio_clock_disable .....	558
表 3-764. 函数 sdio_command_response_config .....	558
表 3-765. 函数 sdio_wait_type_set .....	559
表 3-766. 函数 sdio_csm_enable .....	560
表 3-767. 函数 sdio_csm_disable .....	560
表 3-768. 函数 sdio_command_index_get .....	561
表 3-769. 函数 sdio_response_get .....	561
表 3-770. 函数 sdio_data_config .....	562
表 3-771. 函数 sdio_data_transfer_config .....	563
表 3-772. 函数 sdio_dsm_enable .....	564
表 3-773. 函数 sdio_dsm_disable .....	564
表 3-774. 函数 sdio_data_write .....	565
表 3-775. 函数 sdio_data_read .....	565
表 3-776. 函数 sdio_data_counter_get .....	566
表 3-777. 函数 sdio_data_counter_get .....	566
表 3-778. 函数 sdio_dma_enable .....	567
表 3-779. 函数 sdio_dma_disable .....	567
表 3-780. 函数 sdio_flag_get .....	568
表 3-781. 函数 sdio_flag_clear .....	569
表 3-782. 函数 sdio_interrupt_enable .....	570
表 3-783. 函数 sdio_interrupt_disable .....	572
表 3-784. 函数 sdio_interrupt_flag_get .....	573
表 3-785. 函数 sdio_interrupt_flag_clear .....	575
表 3-786. 函数 sdio_readwait_enable .....	576
表 3-787. 函数 sdio_readwait_disable .....	576
表 3-788. 函数 sdio_stop_readwait_enable .....	577
表 3-789. 函数 sdio_stop_readwait_disable .....	577
表 3-790. 函数 sdio_readwait_type_set .....	578
表 3-791. 函数 sdio_operation_enable .....	578
表 3-792. 函数 sdio_operation_disable .....	579
表 3-793. 函数 sdio_suspend_enable .....	579
表 3-794. 函数 sdio_suspend_disable .....	580
表 3-795. 函数 sdio_ceata_command_enable .....	580
表 3-796. 函数 sdio_ceata_command_disable .....	581
表 3-797. 函数 sdio_ceata_interrupt_enable .....	581
表 3-798. 函数 sdio_ceata_interrupt_disable .....	582

表 3-799. 函数 <code>sdio_ceata_command_completion_enable</code> .....	582
表 3-800. 函数 <code>sdio_ceata_command_completion_disable</code> .....	583
表 3-801. SPI/I2S 寄存器 .....	583
表 3-802. SPI/I2S 库函数 .....	584
表 3-803. 结构体 <code>spi_parameter_struct</code> .....	585
表 3-804. 函数 <code>spi_i2s_deinit</code> .....	585
表 3-805. 函数 <code>spi_struct_para_init</code> .....	586
表 3-806. 函数 <code>spi_init</code> .....	586
表 3-807. 函数 <code>spi_enable</code> .....	587
表 3-808. 函数 <code>spi_disable</code> .....	588
表 3-809. 函数 <code>i2s_init</code> .....	588
表 3-810. 函数 <code>i2s_psc_config</code> .....	589
表 3-811. 函数 <code>i2s_enable</code> .....	591
表 3-812. 函数 <code>i2s_disable</code> .....	591
表 3-813. 函数 <code>spi_nss_output_enable</code> .....	592
表 3-814. 函数 <code>spi_nss_output_disable</code> .....	592
表 3-815. 函数 <code>spi_nss_internal_high</code> .....	593
表 3-816. 函数 <code>spi_nss_internal_low</code> .....	593
表 3-817. 函数 <code>spi_dma_enable</code> .....	594
表 3-818. 函数 <code>spi_dma_disable</code> .....	594
表 3-819. 函数 <code>spi_i2s_data_frame_format_config</code> .....	595
表 3-820. 函数 <code>spi_i2s_data_transmit</code> .....	596
表 3-821. 函数 <code>spi_i2s_data_receive</code> .....	596
表 3-822. 函数 <code>spi_bidirectional_transfer_config</code> .....	597
表 3-823. 函数 <code>spi_i2s_format_error_clear</code> .....	597
表 3-824. 函数 <code>spi_crc_polynomial_set</code> .....	598
表 3-825. 函数 <code>spi_crc_polynomial_get</code> .....	599
表 3-826. 函数 <code>spi_crc_on</code> .....	599
表 3-827. 函数 <code>spi_crc_off</code> .....	600
表 3-828. 函数 <code>spi_crc_next</code> .....	600
表 3-829. 函数 <code>spi_crc_get</code> .....	601
表 3-830. 函数 <code>spi_crc_error_clear</code> .....	601
表 3-831. 函数 <code>spi_ti_mode_enable</code> .....	602
表 3-832. 函数 <code>spi_ti_mode_disable</code> .....	602
表 3-833. 函数 <code>spi_nssp_mode_enable</code> .....	603
表 3-834. 函数 <code>spi_nssp_mode_disable</code> .....	603
表 3-835. 函数 <code>i2s_full_duplex_mode_config</code> .....	604
表 3-836. 函数 <code>spi_quad_enable</code> .....	605
表 3-837. 函数 <code>spi_quad_disable</code> .....	606
表 3-838. 函数 <code>spi_quad_write_enable</code> .....	606
表 3-839. 函数 <code>spi_quad_read_enable</code> .....	607
表 3-840. 函数 <code>spi_quad_io23_output_enable</code> .....	607
表 3-841. 函数 <code>spi_quad_io23_output_disable</code> .....	608
表 3-842. 函数 <code>spi_i2s_interrupt_enable</code> .....	608

表 3-843. 函数 spi_i2s_interrupt_disable .....	609
表 3-844. 函数 spi_i2s_interrupt_flag_get.....	609
表 3-845. 函数 spi_i2s_flag_get .....	610
表 3-846. SQPI 寄存器.....	612
表 3-847. SQPI 库函数.....	612
表 3-848. 结构体 sqpi_parameter_struct .....	612
表 3-849. 函数 sqpi_deinit.....	612
表 3-850. 函数 sqpi_struct_para_init.....	613
表 3-851. 函数 sqpi_init .....	613
表 3-852. 函数 sqpi_read_id_command.....	614
表 3-853. 函数 sqpi_special_command .....	615
表 3-854. 函数 sqpi_read_command_config.....	615
表 3-855. 函数 sqpi_write_command_config.....	616
表 3-856. 函数 sqpi_low_id_receive .....	617
表 3-857. 函数 sqpi_high_id_receive .....	617
表 3-858. TIMER 寄存器 .....	618
表 3-859. TIMER 库函数 .....	619
表 3-860. 结构体 timer_parameter_struct.....	621
表 3-861. 结构体 timer_break_parameter_struct.....	621
表 3-862. 结构体 timer_oc_parameter_struct .....	622
表 3-863. 结构体 timer_ic_parameter_struct.....	622
表 3-864. 函数 timer_deinit .....	622
表 3-865. 函数 timer_struct_para_init .....	623
表 3-866. 函数 timer_init.....	623
表 3-867. 函数 timer_enable.....	624
表 3-868. 函数 timer_disable.....	625
表 3-869. 函数 timer_auto_reload_shadow_enable.....	625
表 3-870. 函数 timer_auto_reload_shadow_disable.....	626
表 3-871. 函数 timer_update_event_enable .....	626
表 3-872. 函数 timer_update_event_disable .....	627
表 3-873. 函数 timer_counter_alignment.....	627
表 3-874. 函数 timer_counter_up_direction .....	628
表 3-875. 函数 timer_counter_down_direction.....	628
表 3-876. 函数 timer_prescaler_config .....	629
表 3-877. 函数 timer_repetition_value_config .....	630
表 3-878. 函数 timer_autoreload_value_config .....	630
表 3-879. 函数 timer_counter_value_config .....	631
表 3-880. 函数 timer_counter_read.....	631
表 3-881. 函数 timer_prescaler_read.....	632
表 3-882. 函数 timer_single_pulse_mode_config .....	632
表 3-883. 函数 timer_update_source_config .....	633
表 3-884. 函数 timer_dma_enable.....	634
表 3-885. 函数 timer_dma_disable.....	635
表 3-886. 函数 timer_channel_dma_request_source_select .....	635

表 3-887. 函数 timer_dma_transfer_config .....	636
表 3-888. 函数 timer_event_software_generate .....	638
表 3-889. 函数 timer_break_struct_para_init .....	639
表 3-890. 函数 timer_break_config .....	639
表 3-891. 函数 timer_break_enable .....	640
表 3-892. 函数 timer_break_disable .....	641
表 3-893. 函数 timer_automatic_output_enable .....	641
表 3-894. 函数 timer_automatic_output_disable .....	642
表 3-895. 函数 timer_primary_output_config .....	642
表 3-896. 函数 timer_channel_control_shadow_config .....	643
表 3-897. 函数 timer_channel_control_shadow_update_config .....	643
表 3-898. 函数 timer_channel_output_struct_para_init .....	644
表 3-899. 函数 timer_channel_output_config .....	645
表 3-900. 函数 timer_channel_output_mode_config .....	646
表 3-901. 函数 timer_channel_output_pulse_value_config .....	647
表 3-902. 函数 timer_channel_output_shadow_config .....	647
表 3-903. 函数 timer_channel_output_fast_config .....	648
表 3-904. 函数 timer_channel_output_clear_config .....	649
表 3-905. 函数 timer_channel_output_polarity_config .....	650
表 3-906. 函数 timer_channel_complementary_output_polarity_config .....	651
表 3-907. 函数 timer_channel_output_state_config .....	652
表 3-908. 函数 timer_channel_complementary_output_state_config .....	652
表 3-909. 函数 timer_channel_input_struct_para_init .....	653
表 3-910. 函数 timer_input_capture_config .....	654
表 3-911. 函数 timer_channel_input_capture_prescaler_config .....	655
表 3-912. 函数 timer_channel_capture_value_register_read .....	656
表 3-913. 函数 timer_input_pwm_capture_config .....	656
表 3-914. 函数 timer_hall_mode_config .....	657
表 3-915. 函数 timer_input_trigger_source_select .....	658
表 3-916. 函数 timer_master_output_trigger_source_select .....	659
表 3-917. 函数 timer_slave_mode_select .....	660
表 3-918. 函数 timer_master_slave_mode_config .....	661
表 3-919. 函数 timer_external_trigger_config .....	661
表 3-920. 函数 timer_quadrature_decoder_mode_config .....	662
表 3-921. 函数 timer_internal_clock_config .....	663
表 3-922. 函数 timer_internal_trigger_as_external_clock_config .....	664
表 3-923. 函数 timer_external_trigger_as_external_clock_config .....	665
表 3-924. 函数 timer_external_clock_mode0_config .....	666
表 3-925. 函数 timer_external_clock_mode1_config .....	667
表 3-926. 函数 timer_external_clock_mode1_disable .....	668
表 3-927. 函数 timer_channel_remap_config .....	668
表 3-928. 函数 timer_write_chxval_register_config .....	669
表 3-929. 函数 timer_output_value_selection_config .....	670
表 3-930. 函数 timer_flag_get .....	670

表 3-931. 函数 timer_flag_clear .....	671
表 3-932. 函数 timer_interrupt_enable .....	672
表 3-933. 函数 timer_interrupt_disable .....	673
表 3-934. 函数 timer_interrupt_flag_get .....	674
表 3-935. 函数 timer_interrupt_flag_clear .....	675
表 3-936. TMU 寄存器 .....	676
表 3-937. TMU 库函数 .....	676
表 3-938. 函数 tmu_deinit .....	677
表 3-939. 函数 tmu_enable .....	677
表 3-940. 函数 tmu_mode_set .....	677
表 3-941. 函数 tmu_idata0_write .....	678
表 3-942. 函数 tmu_idata1_write .....	679
表 3-943. 函数 tmu_data0_read .....	679
表 3-944. 函数 tmu_data1_read .....	680
表 3-945. 函数 tmu_interrupt_enable .....	680
表 3-946. 函数 tmu_interrupt_disable .....	681
表 3-947. 函数 tmu_flag_get .....	681
表 3-948. 函数 tmu_interrupt_flag_get .....	682
表 3-949. USART 寄存器 .....	682
表 3-950. USART 库函数 .....	683
表 3-951. 枚举类型 usart_flag_enum .....	685
表 3-952. 枚举类型 usart5_flag_enum .....	686
表 3-953. 枚举类型 usart_interrupt_flag_enum .....	687
表 3-954. 枚举类型 usart5_interrupt_flag_enum .....	687
表 3-955. 枚举类型 usart_interrupt_enum .....	688
表 3-956. 枚举类型 usart5_interrupt_enum .....	688
表 3-957. 枚举类型 usart_invert_enum .....	688
表 3-958. 枚举类型 usart5_invert_enum .....	689
表 3-959. 函数 usart_deinit .....	689
表 3-960. 函数 usart_baudrate_set .....	689
表 3-961. 函数 usart_parity_config .....	690
表 3-962. 函数 usart_word_length_set .....	691
表 3-963. 函数 usart_stop_bit_set .....	691
表 3-964. 函数 usart_enable .....	692
表 3-965. 函数 usart_disable .....	693
表 3-966. 函数 usart_transmit_config .....	693
表 3-967. 函数 usart_receive_config .....	694
表 3-968. 函数 usart_oversample_config .....	694
表 3-969. 函数 usart_sample_bit_config .....	695
表 3-970. 函数 usart_receiver_timeout_enable .....	696
表 3-971. 函数 usart_receiver_timeout_disable .....	696
表 3-972. 函数 usart_receiver_timeout_threshold_config .....	697
表 3-973. 函数 usart_data_transmit .....	697
表 3-974. 函数 usart_data_receive .....	698

表 3-975. 函数 usart_mute_mode_enable .....	698
表 3-976. 函数 usart_mute_mode_disable .....	699
表 3-977. 函数 usart_mute_mode_wakeup_config .....	699
表 3-978. 函数 usart_lin_mode_enable .....	700
表 3-979. 函数 usart_lin_mode_disable .....	701
表 3-980. 函数 usart_lin_break_dection_length_config .....	701
表 3-981. 函数 usart_halfduplex_enable .....	702
表 3-982. 函数 usart_halfduplex_disable .....	702
表 3-983. 函数 usart_synchronous_clock_enable .....	703
表 3-984. 函数 usart_synchronous_clock_disable .....	703
表 3-985. 函数 usart_synchronous_clock_config .....	704
表 3-986. 函数 usart_guard_time_config .....	705
表 3-987. 函数 usart_smartcard_mode_enable .....	705
表 3-988. 函数 usart_smartcard_mode_disable .....	706
表 3-989. 函数 usart_smartcard_mode_nack_enable .....	706
表 3-990. 函数 usart_smartcard_mode_nack_disable .....	707
表 3-991. 函数 usart_smartcard_autoretry_config .....	707
表 3-992. 函数 usart_block_length_config .....	708
表 3-993. 函数 usart_irda_mode_enable .....	708
表 3-994. 函数 usart_irda_mode_disable .....	709
表 3-995. 函数 usart_prescaler_config .....	709
表 3-996. 函数 usart_irda_lowpower_config .....	710
表 3-997. 函数 usart_dma_receive_config .....	711
表 3-998. 函数 usart_dma_transmit_config .....	711
表 3-999. 函数 usart_hardware_flow_rts_config .....	712
表 3-1000. 函数 usart_hardware_flow_cts_config .....	713
表 3-1001. 函数 usart_data_first_config .....	713
表 3-1002. 函数 usart_invert_config .....	714
表 3-1003. 函数 usart_address_config .....	715
表 3-1004. 函数 usart_send_break .....	715
表 3-1005. 函数 usart_flag_get .....	716
表 3-1006. 函数 usart_flag_clear .....	717
表 3-1007. 函数 usart_interrupt_enable .....	718
表 3-1008. 函数 usart_interrupt_disable .....	719
表 3-1009. 函数 usart_interrupt_flag_get .....	719
表 3-1010. 函数 usart_interrupt_flag_clear .....	721
表 3-1011. 函数 usart5_data_first_config .....	722
表 3-1012. 函数 usart5_invert_config .....	722
表 3-1013. 函数 usart5_overrun_enable .....	723
表 3-1014. 函数 usart5_overrun_disable .....	724
表 3-1015. 函数 usart5_address_config .....	724
表 3-1016. 函数 usart5_address_detection_mode_config .....	725
表 3-1017. 函数 usart5_smartcard_mode_early_nack_enable .....	725
表 3-1018. 函数 usart5_smartcard_mode_early_nack_disable .....	726

表 3-1019. 函数 usart5_reception_error_dma_disable .....	726
表 3-1020. 函数 usart5_reception_error_dma_enable .....	727
表 3-1021. 函数 usart5_wakeup_enable .....	727
表 3-1022. 函数 usart5_wakeup_disable .....	728
表 3-1023. 函数 usart5_wakeup_mode_config.....	728
表 3-1024. 函数 usart5_receive_fifo_enable .....	729
表 3-1025. 函数 usart5_receive_fifo_disable .....	730
表 3-1026. 函数 usart5_receive_fifo_counter_number .....	730
表 3-1027. 函数 usart5_flag_get.....	731
表 3-1028. 函数 usart5_flag_clear.....	732
表 3-1029. 函数 usart5_interrupt_enable.....	733
表 3-1030. 函数 usart5_interrupt_disable.....	734
表 3-1031. 函数 usart5_command_enable .....	735
表 3-1032. 函数 usart5_interrupt_flag_get .....	736
表 3-1033. 函数 usart5_interrupt_flag_clear .....	737
表 3-1034. WWDGT 寄存器.....	739
表 3-1035. WWDGT 库函数.....	739
表 3-1036. 函数 wwdgt_deinit.....	739
表 3-1037. 函数 wwdgt_enable .....	740
表 3-1038. 函数 wwdgt_counter_update.....	740
表 3-1039. 函数 wwdgt_config.....	740
表 3-1040. 函数 wwdgt_interrupt_enable .....	741
表 3-1041. 函数 wwdgt_flag_get .....	742
表 3-1042. 函数 wwdgt_flag_clear .....	743
表 4-1. 版本历史.....	744

## 1. 介绍

本手册介绍了32位基于ARM微控制器GD32E51x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GD32E51x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	备份寄存器
CAN	局域网控制器模块
CRC	循环冗余校验计算单元
CTC	时钟校准控制器
CMP	比较器
DAC	数模转换器
DBG	调试模块
DMA	直接存储器访问控制器
ENET	以太网

外设缩写	说明
EXMC	外部存储器控制器
EXTI	外部中断事件控制器
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
SHRTIMER	高精度定时器
I2C	内部集成电路总线接口
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
SDIO	SDIO接口
SPI/I2S	串行外设接口/片上音频接口
SQPI	SQPI接口
TIMER	定时器
TMU	三角数学单元
USART	通用同步异步收发器
WWDGT	窗口看门狗
USB_D	USB_D
USBHS	USBHS

### 1.1.2. 命名规则

固件库遵从以下命名规则：

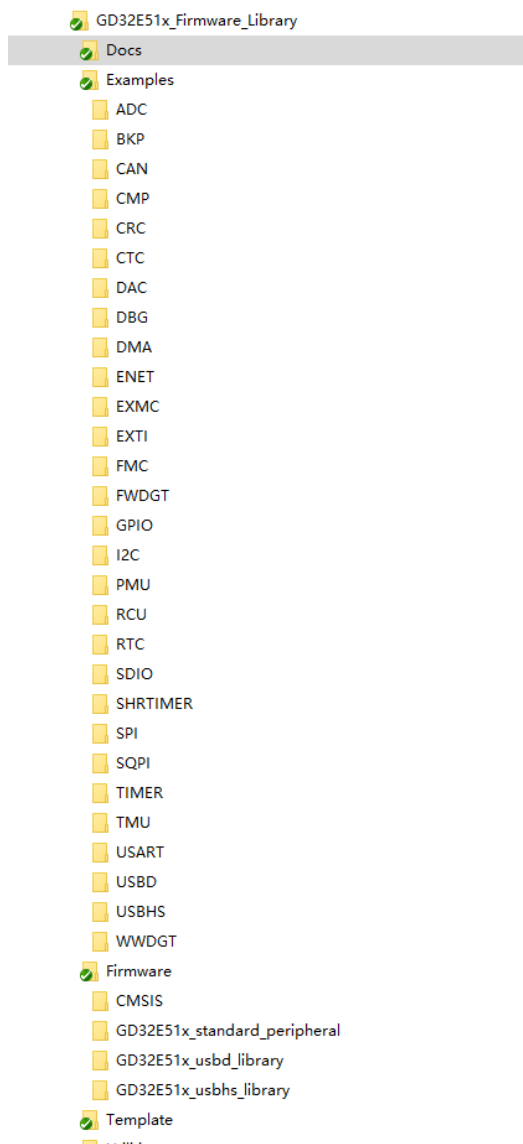
- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；
- 源文件和头文件命名都以“GD32E51x\_”作为开头，例如：GD32E51x\_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

## 2. 固件库概述

### 2.1. 文件组织结构

GD32E51x\_Firmware\_Library，文件组织结构见下图：

图 2-1. GD32E51x 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；
- **GD32E51x\_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）；

- **GD32E51x\_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）；
- **GD32E51x\_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GD32E51x的全局头文件和系统配置文件；
- **GD32E51x\_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；
- **GD32E51x\_usbd\_library**子文件夹包含了关于USB D外设的所有文件：
  - **Include**子文件夹包含了USB D外设所需的头文件，用户无需修改该文件夹；
  - **Source**子文件夹包含了USB D外设所需的源文件，用户无需修改该文件夹；
- **GD32E51x\_usbhs\_library**子文件夹包含了关于USB HS外设的所有文件：
  - **Include**子文件夹包含了USB HS外设所需的头文件，用户无需修改该文件夹；
  - **Source**子文件夹包含了USB HS外设所需的源文件，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。

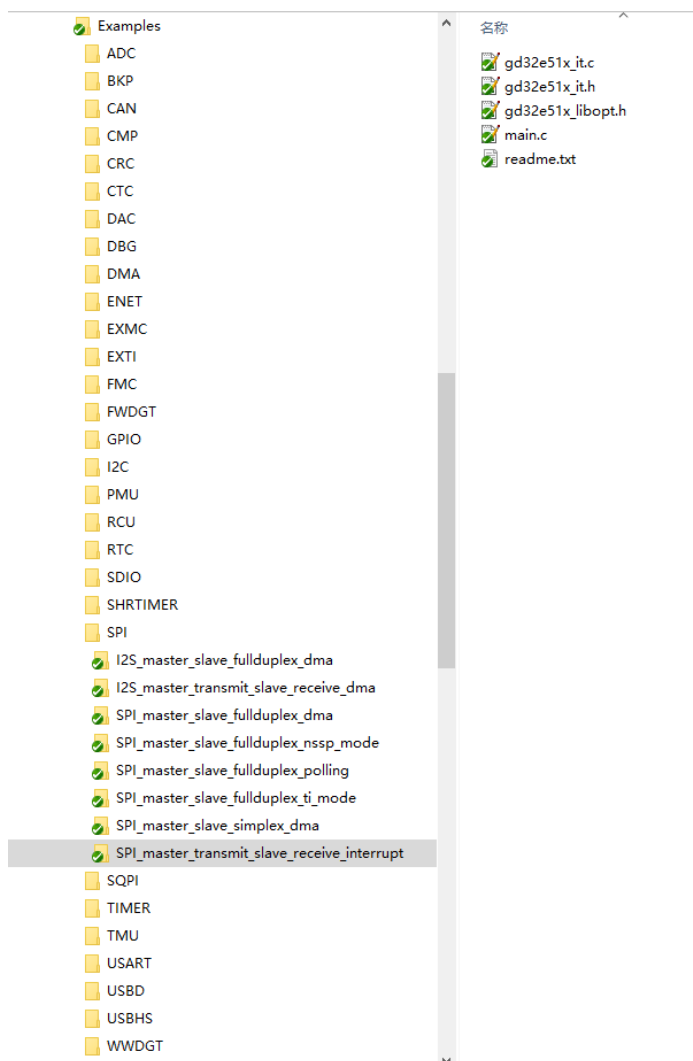
### 2.1.3. Template 文件夹

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SPI，打开“SPI”文件夹，选择SPI的一个例程，如“SPI\_master\_transmit\_slave\_receive\_interrupt”，如下图所示：

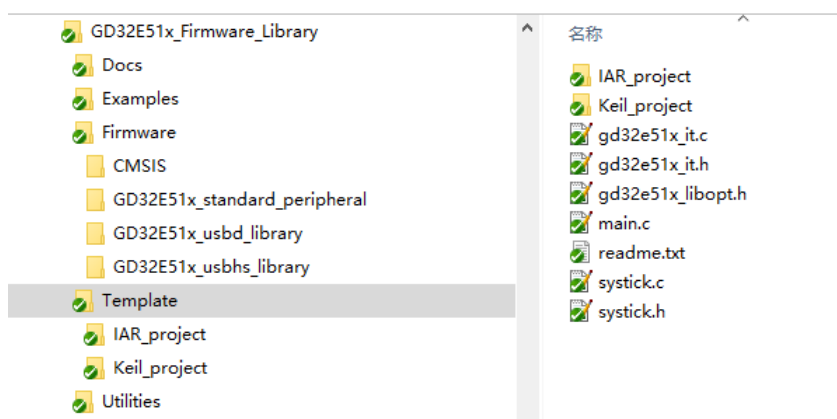
图 2-2. 选择外设例程文件



### 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“SPI\_master\_transmit\_slave\_receive\_interrupt”文件夹中的所有文件拷到“Template”文件夹子目录下，如下图所示：

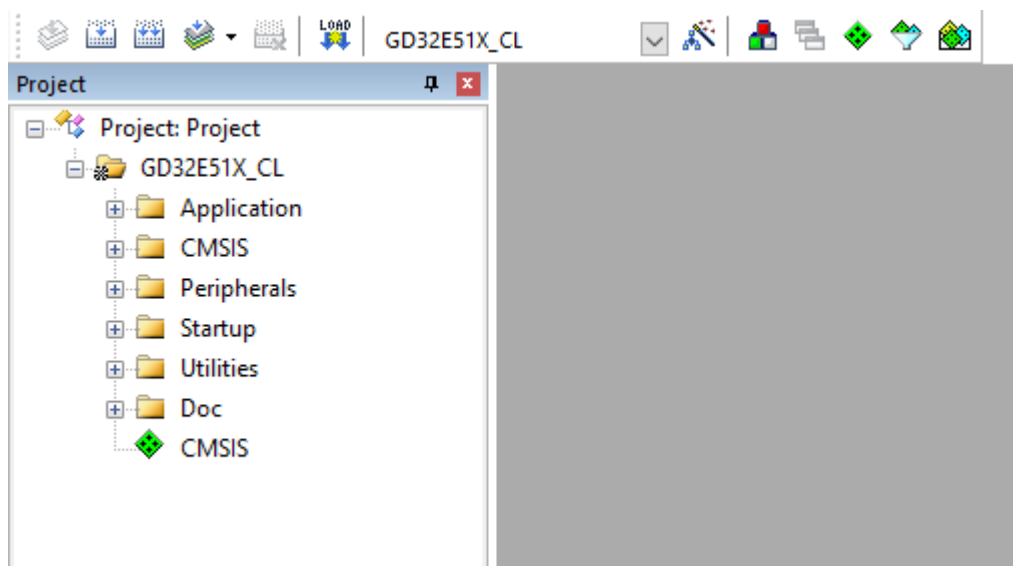
图 2-3. 拷贝外设例程文件



### 打开工程

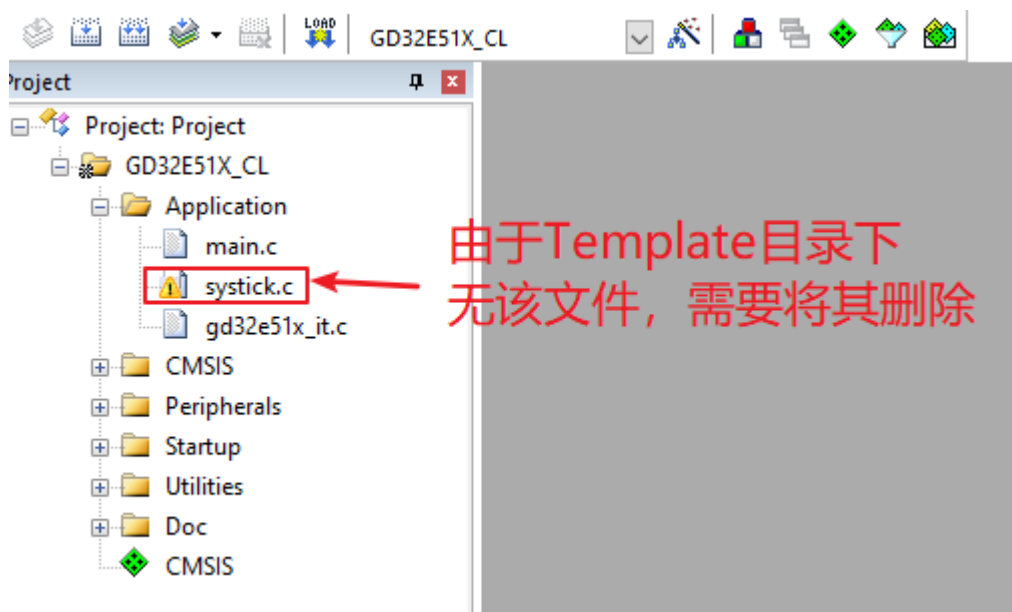
GD 提供 Keil 和 IAR 两种版本的工程，根据客户所安装的软件，打开不同的 project，如“Keil\_project”，打开\Template\Keil\_project\Project.uvprojx，如下图所示：

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：

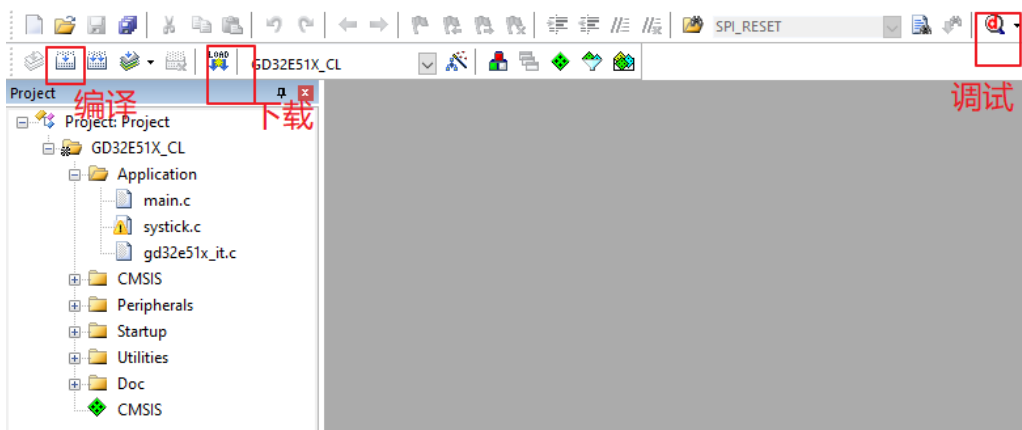
图 2-5. 配置工程文件



### 编译调试下载

首先编译整个工程，若无错误，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：

- LCD\_Commom子文件夹包含有USB测试所需文件；
- GD32E51x\_eval.h及GD32E51x\_lcd\_eval.h文件是运行固件库例程所需关于评估板的头文件；
- GD32E51x\_eval.c及GD32E51x\_lcd\_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
GD32E51x_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。
GD32E51x_it.h	头文件，包含所有中断处理函数原形。
GD32E51x_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。
GD32E51x_xxx.h	外设PPP的头文件。包含外设PPP函数的定义，以及这些函数使用的变量。
GD32E51x_xxx.c	由C语言编写的外设PPP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 (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 (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_WD2SR	看门狗2通道选择寄存器
ADC_WDT1	看门狗1阈值寄存器
ADC_WDT2	看门狗2阈值寄存器
ADC_DIFCTL	差分模式控制寄存器

### 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_calibration_number	配置ADC校准次数
adc_dma_mode_enable	ADC DMA请求使能
adc_dma_mode_disable	ADC DMA请求禁能
adc_tempsensor_vrefint_enable	温度传感器和vrefint通道使能
adc_tempsensor_vrefint_disable	温度传感器和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_channel_differential_mode_config	配置ADC通道差分模式
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

库函数名称	库函数描述
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
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_watchdog2_channel_config	配置ADC模拟看门狗2通道
adc_watchdog1_disable	ADC模拟看门狗1禁能
adc_watchdog2_disable	ADC模拟看门狗2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_watchdog2_threshold_config	配置ADC模拟看门狗2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..2)	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..2)	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..2)	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	<code>adc_calibration_enable</code>
函数原形	<code>void adc_calibration_enable(uint32_t adc_periph);</code>
功能描述	ADC校准复位
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..2)</code>	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_calibration_number`

函数`adc_calibration_number`描述见下表：

表 3-8. 函数 `adc_calibration_number`

函数名称	<code>adc_calibration_number</code>
函数原形	<code>void adc_calibration_number(uint32_t adc_periph, uint32_t clb_num);</code>
功能描述	配置ADC校准次数
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..2)</code>	ADC外设选择
输入参数{in}	
<code>clb_num</code>	校准次数
<code>ADC_CALIBRATION_NUM1</code>	校准1次

ADC_CALIBRATION_NUM2	校准2次
ADC_CALIBRATION_NUM4	校准4次
ADC_CALIBRATION_NUM8	校准8次
ADC_CALIBRATION_NUM16	校准16次
ADC_CALIBRATION_NUM32	校准32次
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
adc_calibration_number(ADC0, ADC_CALIBRATION_NUM16);
```

### 函数 adc\_dma\_mode\_enable

函数adc\_dma\_mode\_enable描述见下表：

表 3-9. 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..2)	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 3-10. 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..2)	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\_vrefint\_enable

函数adc\_tempsensor\_vrefint\_enable描述见下表：

**表 3-11. 函数 adc\_tempsensor\_vrefint\_enable**

函数名称	adc_tempsensor_vrefint_enable
函数原形	void adc_tempsensor_vrefint_enable(void);
功能描述	温度传感器和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 temperature sensor and vrefint channel */
adc_tempsensor_vrefint_enable();
```

函数 `adc_tempsensor_vrefint_disable`

函数`adc_tempsensor_vrefint_disable`描述见下表:

表 3-12. 函数 `adc_tempsensor_vrefint_disable`

函数名称	<code>adc_tempsensor_vrefint_disable</code>
函数原形	<code>void adc_tempsensor_vrefint_disable(void);</code>
功能描述	温度传感器和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 temperature sensor and vrefint channel */
```

```
adc_tempsensor_vrefint_disable();
```

函数 `adc_discontinuous_mode_config`

函数`adc_discontinuous_mode_config`描述见下表:

表 3-13. 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}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0..2)</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
<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 3-14. 函数 adc\_mode\_config**

函数名称	adc_mode_config
函数原形	void adc_mode_config(uint32_t mode);
功能描述	配置ADC同步模式
先决条件	-
被调用函数	-
输入参数{in}	
<b>mode</b>	ADC模式
ADC_MODE_FREE	所有ADC运行于独立模式
ADC_DUAL_REGULAR_PARALLEL_INSERTED_PARALLEL	ADC0和ADC1运行在规则并行+注入并行组合模式
ADC_DUAL_REGULAR_PARALLEL_INSERTED_ROTATION	ADC0和ADC1运行在规则并行+交替触发组合模式
ADC_DUAL_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_DUAL_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_DUAL_INSERTED_PARALLEL	ADC0和ADC1运行在注入并行模式
ADC_DUAL_REGULAR	ADC0和ADC1运行在规则并行模式

LAL_PARALLEL	
ADC_DUAL_REGU LAL_FOLLOWUP_F AST	ADC0和ADC1运行在快速交叉模式
ADC_DUAL_REGU LAL_FOLLOWUP_S LOW	ADC0和ADC1运行在慢速交叉模式
ADC_DUAL_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 3-15. 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..2)	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 3-16. 函数 `adc_data_alignment_config`**

函数名称	<code>adc_data_alignment_config</code>
函数原形	<code>void adc_data_alignment_config(uint32_t adc_periph, uint32_t data_alignment);</code>
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}	
<b>adc_periph</b>	ADC外设
<i>ADCx(x=0..2)</i>	ADC外设选择
输入参数{in}	
<b>data_alignment</b>	数据对齐方式选择
<i>ADC_DATAALIGN_RIGHT</i>	右对齐
<i>ADC_DATAALIGN_LEFT</i>	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 3-17. 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..2)</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 3-18. 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..2)</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
<i>ADC_CHANNEL_x</i> (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
<i>ADC_SAMPLETIME_1POINT5</i>	1.5周期
<i>ADC_SAMPLETIME_7POINT5</i>	7.5周期
<i>ADC_SAMPLETIME_13POINT5</i>	13.5周期
<i>ADC_SAMPLETIME_28POINT5</i>	28.5周期
<i>ADC_SAMPLETIME_41POINT5</i>	41.5周期
<i>ADC_SAMPLETIME_55POINT5</i>	55.5周期
<i>ADC_SAMPLETIME_71POINT5</i>	71.5周期
<i>ADC_SAMPLETIME_239POINT5</i>	239.5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_7POINT5);
```

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

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

**表 3-19. 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
$ADCx(x=0..2)$	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
$ADC\_CHANNEL\_x$ ( $x=0..17$ )	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
$ADC\_SAMPLETIME\_1POINT5$	1.5周期
$ADC\_SAMPLETIME\_7POINT5$	7.5周期
$ADC\_SAMPLETIME\_13POINT5$	13.5周期
$ADC\_SAMPLETIME\_28POINT5$	28.5周期
$ADC\_SAMPLETIME\_41POINT5$	41.5周期
$ADC\_SAMPLETIME\_55POINT5$	55.5周期
$ADC\_SAMPLETIME\_71POINT5$	71.5周期
$ADC\_SAMPLETIME\_239POINT5$	239.5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_7POINT5);
```

### 函数 `adc_inserted_channel_offset_config`

函数`adc_inserted_channel_offset_config`描述见下表：

**表 3-20. 函数 `adc_inserted_channel_offset_config`**

函数名称	<code>adc_inserted_channel_offset_config</code>
函数原形	<code>void adc_inserted_channel_offset_config(uint32_t adc_periph, uint8_t</code>

	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..2)	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\_channel\_differential\_mode\_config

函数adc\_channel\_differential\_mode\_config描述见下表:

表 3-21. 函数 adc\_channel\_differential\_mode\_config

函数名称	adc_channel_differential_mode_config
函数原形	void adc_channel_differential_mode_config(uint32_t adc_periph, uint32_t adc_channel, ControlStatus newvalue);
功能描述	配置ADC通道差分模式
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
inserted_channel	通道使用差分模式
ADC_DIFFERENTIAL_MODE_CHANNEL_x (x=0..14), ADC_DIFFERENTIAL_MODE_CHANNEL_x	ADC通道差分模式 (仅适用于通道0~通道14)

<i>EL_ALL</i>	
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:

```
/* configure differential mode for ADC channel */
```

```
adc_channel_differential_mode_config(ADC0,  
ADC_DIFFERENTIAL_MODE_CHANNEL_ALL, ENABLE);
```

### 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}	
<b>adc_periph</b>	ADC外设
<i>ADCx(x=0..2)</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>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}	
adc_periph	ADC外设
ADCx(x=0..2)	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}	
external_trigger_source	规则通道组或注入通道组触发源
ADC0_1_EXTTRIG_REGULAR_T0_CH0	TIMER0 CH0事件（规则组）
ADC0_1_EXTTRIG_REGULAR_T0_CH1	TIMER0 CH1事件（规则组）
ADC0_1_EXTTRIG_REGULAR_T0_CH2	TIMER0 CH2事件（规则组）
ADC0_1_EXTTRIG_REGULAR_T1_CH1	TIMER1 CH1事件（规则组）
ADC0_1_EXTTRIG	TIMER2 TRGO事件（规则组）

_REGULAR_T2 _TRGO	
ADC0_1_EXTTRIG _REGULAR_T3 _CH3	TIMER3 CH3事件（规则组）
ADC0_1_EXTTRIG _REGULAR_T7 _TRGO	TIMER7 TRGO事件（规则组）
ADC0_1_EXTTRIG _REGULAR_ EXTI_11	外部中断线11（规则组）
ADC0_1_EXTTRIG _REGULAR_SHRTI MER_ADCTRG0	SHRTIMER_ADCTRG0输出选择（规则组）（仅用于GD32E51X_HD和GD32E51X_CL系列芯片）
ADC0_1_EXTTRIG _REGULAR_SHRTI MER_ADCTRG2	SHRTIMER_ADCTRG2输出选择（规则组）（仅用于GD32E51X_HD和GD32E51X_CL系列芯片）
ADC2_EXTTRIG_ REGULAR_T2_CH0	TIMER2 CH0事件（规则组）
ADC2_EXTTRIG_ REGULAR_T1_CH2	TIMER1 CH2事件（规则组）
ADC2_EXTTRIG_ REGULAR_T0_CH2	TIMER0 CH2事件（规则组）
ADC2_EXTTRIG_ REGULAR_T7_CH0	TIMER7 CH0事件（规则组）
ADC2_EXTTRIG_ REGULAR_T7 _TRGO	TIMER7 TRGO事件（规则组）
ADC2_EXTTRIG_ REGULAR_T4_CH0	TIMER4 CH0事件（规则组）
ADC2_EXTTRIG_ REGULAR_T4_CH2	TIMER4 CH2事件（规则组）
ADC0_1_2_EXTTRI G_REGULAR _NONE	软件触发（规则组）
ADC0_1_EXTTRIG _INSERTED_ T0_TRGO	TIMER0 TRGO事件（注入组）
ADC0_1_EXTTRIG _INSERTED_T0 _CH3	TIMER0 CH3事件（注入组）
ADC0_1_EXTTRIG _INSERTED_T1	TIMER1 TRGO事件（注入组）

_TRGO	
ADC0_1_EXTTRIG_INSERTED_T1_CH0	TIMER1 CH0事件（注入组）
ADC0_1_EXTTRIG_INSERTED_T2_CH3	TIMER2 CH3事件（注入组）
ADC0_1_EXTTRIG_INSERTED_T3_TRGO	TIMER3 TRGO事件（注入组）
ADC0_1_EXTTRIG_INSERTED_T7_CH3	TIMER7 CH3事件（注入组）
ADC0_1_EXTTRIG_INSERTED_EXTI_15	外部中断线15（注入组）
ADC0_1_EXTTRIG_INSERTED_SHRTIMER_ADCTRG1	SHRTIMER_ADCTRG1输出选择（注入组）（仅用于GD32E51X_HD和GD32E51X_CL系列芯片）
ADC0_1_EXTTRIG_INSERTED_SHRTIMER_ADCTRG3	SHRTIMER_ADCTRG3输出选择（注入组）（仅用于GD32E51X_HD和GD32E51X_CL系列芯片）
ADC2_EXTTRIG_INSERTED_T0_TRGO	TIMER0 TRGO事件（注入组）
ADC2_EXTTRIG_INSERTED_T0_CH3	TIMER0 CH3事件（注入组）
ADC2_EXTTRIG_INSERTED_T3_CH2	TIMER3 CH2事件（注入组）
ADC2_EXTTRIG_INSERTED_T7_CH1	TIMER7 CH1事件（注入组）
ADC2_EXTTRIG_INSERTED_T7_CH3	TIMER7 CH3事件（注入组）
ADC2_EXTTRIG_INSERTED_T4_TRGO	TIMER4 TRGO事件（注入组）
ADC2_EXTTRIG_INSERTED_T4_CH3	TIMER4 CH3事件（注入组）
ADC0_1_2_EXTTRIG_INSERTED_NONE	软件触发（注入组）
输出参数{out}	
-	-

返回值	
-	-

例如：

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

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

### 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	adc_software_trigger_enable
函数原形	void adc_software_trigger_enable(uint32_t adc_periph, uint8_t adc_channel_group);
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}	
adc_periph	ADC外设
ADCx(x=0..2)	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	adc_regular_data_read
------	-----------------------

函数原形	uint16_t adc_regular_data_read(uint32_t adc_periph);
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}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输出参数{out}	
-	-
返回值	
uint16_t	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	adc_inserted_data_read
函数原形	uint16_t adc_inserted_data_read(uint32_t adc_periph, uint8_t inserted_channel);
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}	
adc_periph	ADC外设
ADCx(x=0..2)	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, 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	<code>adc_sync_mode_convert_value_read</code>
函数原形	<code>uint32_t adc_sync_mode_convert_value_read(void);</code>
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	
<code>uint32_t</code>	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	<code>adc_watchdog0_single_channel_enable</code>
函数原形	<code>void adc_watchdog0_single_channel_enable(uint32_t adc_periph, uint8_t adc_channel);</code>
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}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0..2)</code>	ADC外设选择
输入参数{in}	
<code>adc_channel</code>	选择ADC通道
<code>ADC_CHANNEL_x</code> ( <code>x=0..17</code> )	ADC通道x( <code>x=0..17</code> ) (只有ADC0, 可取值x=16和17)

输出参数{out}	
-	-
返回值	
-	-

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`**

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

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..2)	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..2)	ADC外设选择
输入参数{in}	
adc_channel	选择ADC通道
ADC_AWD1_2_SELECTION_CHANNEL_x (x=0..17), ADC_AWD1_2_SELECTION_CHANNEL_ALL	ADC通道模拟看门狗1/2选择 (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_2_SELECTION_CHANNEL_1,
ENABLE);
```

### 函数 `adc_watchdog2_channel_config`

函数`adc_watchdog2_channel_config`描述见下表：

**表 3-32. 函数 `adc_watchdog2_channel_config`**

函数名称	<code>adc_watchdog2_channel_config</code>
函数原形	<code>void adc_watchdog2_channel_config(uint32_t adc_periph, uint32_t adc_channel, ControlStatus newvalue);</code>
功能描述	配置ADC模拟看门狗2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..2)</i>	ADC外设选择
输入参数{in}	
<b>adc_channel</b>	选择ADC通道
<i>ADC_AWD1_2_SELECTION_CHANNEL_x (x=0..17),</i> <i>ADC_AWD1_2_SELECTION_CHANNEL_ALL</i>	ADC通道模拟看门狗1/2选择 (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 2 channel */
```

```
adc_watchdog2_channel_config(ADC0, ADC_AWD1_2_SELECTION_CHANNEL_1,
ENABLE);
```

### 函数 `adc_watchdog1_disable`

函数`adc_watchdog1_disable`描述见下表：

**表 3-33. 函数 `adc_watchdog1_disable`**

函数名称	<code>adc_watchdog1_disable</code>
函数原形	<code>void adc_watchdog1_disable(uint32_t adc_periph);</code>
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}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0..2)</code>	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_watchdog2_disable`

函数`adc_watchdog2_disable`描述见下表：

**表 3-34. 函数 `adc_watchdog2_disable`**

函数名称	<code>adc_watchdog2_disable</code>
函数原形	<code>void adc_watchdog2_disable(uint32_t adc_periph);</code>
功能描述	ADC模拟看门狗2禁能
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..2)</code>	ADC外设选择
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable ADC0 analog watchdog 2 */
```

```
adc_watchdog2_disable(ADC0);
```

### 函数 `adc_watchdog0_threshold_config`

函数 `adc_watchdog0_threshold_config` 描述见下表：

**表 3-35. 函数 `adc_watchdog0_threshold_config`**

函数名称	<code>adc_watchdog0_threshold_config</code>
函数原形	<code>void adc_watchdog0_threshold_config(uint32_t adc_periph, uint16_t low_threshold, uint16_t high_threshold);</code>
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}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0..2)</code>	ADC外设选择
输入参数{in}	
<code>low_threshold</code>	模拟看门狗0低阈值，0..4095
输入参数{in}	
<code>high_threshold</code>	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 3-36. 函数 `adc_watchdog1_threshold_config`**

函数名称	<code>adc_watchdog1_threshold_config</code>
函数原形	<code>void adc_watchdog1_threshold_config(uint32_t adc_periph, uint8_t low_threshold, uint8_t high_threshold);</code>
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..2)	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}	
high_threshold	模拟看门狗1高阈值, 0..255
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

### 函数 adc\_watchdog2\_threshold\_config

函数adc\_watchdog2\_threshold\_config描述见下表:

表 3-37. 函数 adc\_watchdog2\_threshold\_config

函数名称	adc_watchdog2_threshold_config
函数原形	void adc_watchdog2_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2阈值
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
low_threshold	模拟看门狗2低阈值, 0..255
输入参数{in}	
high_threshold	模拟看门狗2高阈值, 0..255
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC0 analog watchdog 2 threshold */
```

```
adc_watchdog2_threshold_config(ADC0, 0x40, 0xA0);
```

## 函数 `adc_resolution_config`

函数`adc_resolution_config`描述见下表：

**表 3-38. 函数 `adc_resolution_config`**

函数名称	<code>adc_resolution_config</code>
函数原形	<code>void adc_resolution_config(uint32_t adc_periph, uint32_t resolution);</code>
功能描述	配置ADC分辨率
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..2)</code>	ADC外设选择
输入参数{in}	
<code>resolution</code>	ADC分辨率
<code>ADC_RESOLUTION_12B</code>	12位分辨率
<code>ADC_RESOLUTION_10B</code>	10位分辨率
<code>ADC_RESOLUTION_8B</code>	8位分辨率
<code>ADC_RESOLUTION_6B</code>	6位分辨率
输出参数{out}	
-	-
返回值	
-	-

Example:

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

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

## 函数 `adc_oversample_mode_config`

函数`adc_oversample_mode_config`描述见下表：

**表 3-39. 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}	
<b>adc_periph</b>	ADC外设
<i>ADCx(x=0..2)</i>	ADC外设选择
输入参数{in}	
<b>mode</b>	ADC过采样触发模式
<i>ADC_OVERSAMPLING_ALL_CONVERT</i>	在一个触发之后，对一个通道连续进行过采样转换
<i>ADC_OVERSAMPLING_ONE_CONVERT</i>	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}	
<b>shift</b>	ADC过滤采样移位
<i>ADC_OVERSAMPLING_SHIFT_NONE</i>	不移位
<i>ADC_OVERSAMPLING_SHIFT_1B</i>	移1位
<i>ADC_OVERSAMPLING_SHIFT_2B</i>	移2位
<i>ADC_OVERSAMPLING_SHIFT_3B</i>	移3位
<i>ADC_OVERSAMPLING_SHIFT_4B</i>	移4位
<i>ADC_OVERSAMPLING_SHIFT_5B</i>	移5位
<i>ADC_OVERSAMPLING_SHIFT_6B</i>	移6位
<i>ADC_OVERSAMPLING_SHIFT_7B</i>	移7位
<i>ADC_OVERSAMPLING_SHIFT_8B</i>	移8位
输入参数{in}	
<b>ratio</b>	ADC过采样率
<i>ADC_OVERSAMPLING_RATIO_2X</i>	2x

MPLING_RATIO _MUL2	
ADC_OVERSA MPLING_RATIO _MUL4	4x
ADC_OVERSA MPLING_RATIO _MUL8	8x
ADC_OVERSA MPLING_RATIO _MUL16	16x
ADC_OVERSA MPLING_RATIO _MUL32	32x
ADC_OVERSA MPLING_RATIO _MUL64	64x
ADC_OVERSA MPLING_RATIO _MUL128	128x
ADC_OVERSA MPLING_RATIO _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 3-40. 函数 adc\_oversample\_mode\_enable

函数名称	adc_oversample_mode_enable
函数原形	void adc_oversample_mode_enable(uint32_t adc_periph);
功能描述	使能ADC过采样
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..2)</i>	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 3-41. 函数 **adc\_oversample\_mode\_disable****

<b>函数名称</b>	adc_oversample_mode_disable
<b>函数原形</b>	void adc_oversample_mode_disable(uint32_t adc_periph);
<b>功能描述</b>	禁能ADC过采样
<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..2)</i>	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 3-42. 函数 **adc\_flag\_get****

<b>函数名称</b>	adc_flag_get
<b>函数原形</b>	FlagStatus adc_flag_get(uint32_t adc_periph , uint32_t flag);
<b>功能描述</b>	获取ADC标志位
<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..2)</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
<i>ADC_FLAG_WDE2</i>	模拟看门狗2事件标志位
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	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 3-43. 函数 **adc\_flag\_clear****

<b>函数名称</b>	adc_flag_clear
<b>函数原形</b>	void adc_flag_clear(uint32_t adc_periph, uint32_t flag);
<b>功能描述</b>	清除ADC标志位
<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..2)</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

ADC_FLAG_WDE1	模拟看门狗1事件标志位
ADC_FLAG_WDE2	模拟看门狗2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 3-44. 函数 adc\_interrupt\_enable

函数名称	adc_interrupt_enable
函数原形	void adc_interrupt_enable(uint32_t adc_periph, uint32_t interrupt);
功能描述	ADC中断使能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	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
ADC_INT_WDE2	模拟看门狗2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 3-45. 函数 `adc_interrupt_disable`

函数名称	<code>adc_interrupt_disable</code>
函数原形	<code>void adc_interrupt_enable(uint32_t adc_periph, uint32_t interrupt);</code>
功能描述	ADC中断禁能
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..2)</code>	ADC外设选择
输入参数{in}	
<code>interrupt</code>	ADC中断标志位
<code>ADC_INT_WDE0</code>	模拟看门狗0中断标志位
<code>ADC_INT_EOC</code>	组转换结束中断标志位
<code>ADC_INT_EOIC</code>	注入通道组转换结束中断标志位
<code>ADC_INT_WDE1</code>	模拟看门狗1中断标志位
<code>ADC_INT_WDE2</code>	模拟看门狗2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 3-46. 函数 `adc_interrupt_flag_get`

函数名称	<code>adc_interrupt_flag_get</code>
函数原形	<code>FlagStatus adc_interrupt_flag_get(uint32_t adc_periph, uint32_t int_flag);</code>
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}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0..2)</code>	ADC外设选择
输入参数{in}	
<code>int_flag</code>	ADC中断标志位
<code>ADC_INT_FLAG_WDE0</code>	模拟看门狗0中断标志位
<code>ADC_INT_FLAG_</code>	组转换结束中断标志位

EOC	
ADC_INT_FLAG_EOIC	注入通道组转换结束中断标志位
ADC_INT_FLAG_WDE1	模拟看门狗1中断标志位
ADC_INT_FLAG_WDE2	模拟看门狗2中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	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 3-47. 函数 adc\_interrupt\_flag\_clear

函数名称	adc_interrupt_flag_clear
函数原形	void adc_interrupt_flag_clear(uint32_t adc_periph, uint32_t int_flag);
功能描述	清除ADC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
int_flag	ADC中断标志位
ADC_INT_FLAG_WDE0	模拟看门狗0中断标志位
ADC_INT_FLAG_EOC	组转换结束中断标志位
ADC_INT_FLAG_EOIC	注入通道组转换结束中断标志位
ADC_INT_FLAG_WDE1	模拟看门狗1中断标志位
ADC_INT_FLAG_WDE2	模拟看门狗2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可在 $V_{DD}$ 电源关闭时由 $V_{BAT}$ 供电，备份寄存器有42个16位（84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 3-48. BKP 寄存器

寄存器名称	寄存器描述
BKP_DATAx (x=0..41)	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 3-49. BKP 库函数

库函数名称	库函数描述
bkp_deinit	复位备份数据寄存器
bkp_write_data	写备份数据寄存器
bkp_read_data	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
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	RTC时钟校准值

库函数名称	库函数描述
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_tamper_interrupt_enable	TAMPER中断使能
bkp_tamper_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 3-50. 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
BKP_DATA_10	BKP数据寄存器10
BKP_DATA_11	BKP数据寄存器11
BKP_DATA_12	BKP数据寄存器12
BKP_DATA_13	BKP数据寄存器13
BKP_DATA_14	BKP数据寄存器14
BKP_DATA_15	BKP数据寄存器15
BKP_DATA_16	BKP数据寄存器16
BKP_DATA_17	BKP数据寄存器17
BKP_DATA_18	BKP数据寄存器18
BKP_DATA_19	BKP数据寄存器19
BKP_DATA_20	BKP数据寄存器20
BKP_DATA_21	BKP数据寄存器21
BKP_DATA_22	BKP数据寄存器22
BKP_DATA_23	BKP数据寄存器23
BKP_DATA_24	BKP数据寄存器24
BKP_DATA_25	BKP数据寄存器25
BKP_DATA_26	BKP数据寄存器26
BKP_DATA_27	BKP数据寄存器27

成员名称	功能描述
BKP_DATA_28	BKP数据寄存器28
BKP_DATA_29	BKP数据寄存器29
BKP_DATA_30	BKP数据寄存器30
BKP_DATA_31	BKP数据寄存器31
BKP_DATA_32	BKP数据寄存器32
BKP_DATA_33	BKP数据寄存器33
BKP_DATA_34	BKP数据寄存器34
BKP_DATA_35	BKP数据寄存器35
BKP_DATA_36	BKP数据寄存器36
BKP_DATA_37	BKP数据寄存器37
BKP_DATA_38	BKP数据寄存器38
BKP_DATA_39	BKP数据寄存器39
BKP_DATA_40	BKP数据寄存器40
BKP_DATA_41	BKP数据寄存器41

### 函数 bkp\_deinit

函数bkp\_deinit描述见下表：

表 3-51. 函数 bkp\_deinit

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

例如：

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

```
bkp_deinit ();
```

### 函数 bkp\_write\_data

函数bkp\_write\_data描述见下表：

表 3-52. 函数 bkp\_write\_data

函数名称	bkp_write_data
函数原型	void bkp_write_data(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3-50. 枚举类型bkp_data_register_enum</a>
BKP_DATA_x(x = 0..41)	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_write_data (BKP_DATA_0, 0x1226);
```

### 函数 bkp\_read\_data

函数bkp\_read\_data描述见下表:

表 3-53. 函数 bkp\_read\_data

函数名称	bkp_read_data
函数原型	uint16_t bkp_read_data(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3-50. 枚举类型bkp_data_register_enum</a>
BKP_DATA_x(x = 0..41)	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_read_data (BKP_DATA_0);
```

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

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

**表 3-54. 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 3-55. 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 3-56. 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 3-57. 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 3-58. 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_AL ARM_PULSE	RTC闹钟脉冲被选择为RTC输出
RTC_OUTPUT_SE COND_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 3-59. 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 3-60. 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_SLO WED_DOWN	RTC时钟变慢
RTC_CLOCK_SPE ED_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 3-61. 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\_tamper\_detection\_enable

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

表 3-62. 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 3-63. 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 3-64. 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 3-65. 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 3-66. 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 3-67. 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}	
flag	要获取的标志
BKP_FLAG_TAMP	侵入事件标志

<i>ER</i>	
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 3-68. 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}	
flag	要清除的标志
BKP_FLAG_TAMP ER	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 3-69. 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 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}	
int_flag	要获取的中断标志
BKP_INT_FLAG_T AMPER	侵入事件中标志
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 3-70. 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 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}	
int_flag	要清除的中断标志
BKP_INT_FLAG_T AMPER	侵入事件中标志
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或者设备之间相互通信的总线标准。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 3-71. CAN 寄存器

寄存器名称	寄存器描述
CAN_CTL	控制寄存器
CAN_STAT	状态寄存器
CAN_TSTAT	发送状态寄存器
CAN_RFIFO0	接收 FIFO0 寄存器
CAN_RFIFO1	接收 FIFO1 寄存器
CAN_INTEN	中断使能寄存器
CAN_ERR	错误寄存器
CAN_BT	位时序寄存器
CAN_TMIx	发送邮箱标识符寄存器
CAN_TMPx	发送邮箱属性寄存器
CAN_TMDATA0x	发送邮箱 data0 寄存器
CAN_TMDATA1x	发送邮箱 data1 寄存器
CAN_RFIFOMIx	接收 FIFO 邮箱标识符寄存器
CAN_RFIFOMPx	接收 FIFO 邮箱属性寄存器
CAN_RFIFOMDATA0x	接收 FIFO 邮箱 data0 寄存器
CAN_RFIFOMDATA1x	接收 FIFO 邮箱 data1 寄存器
CAN_FCTL	过滤器控制寄存器
CAN_FMCFG	过滤器模式配置寄存器
CAN_FSCFG	过滤器位宽配置寄存器
CAN_FAFIFO	过滤器关联 FIFO 寄存器
CAN_FW	过滤器激活寄存器
CAN_FxDATAy	过滤器(x)数据(y)寄存器

表 3-72. CAN-FD 寄存器

寄存器名称	寄存器描述
CAN_CTL	控制寄存器
CAN_STAT	状态寄存器
CAN_TSTAT	发送状态寄存器
CAN_RFIFO0	接收 FIFO0 寄存器
CAN_RFIFO1	接收 FIFO1 寄存器
CAN_INTEN	中断使能寄存器
CAN_ERR	错误寄存器

寄存器名称	寄存器描述
CAN_BT	位时序寄存器
CAN_FDCTL	FD 控制寄存器
CAN_FDSTAT	FD 状态寄存器
CAN_FDTDC	FD 传输延迟补偿寄存器
CAN_DBT	数据位时序寄存器
CAN_TMIx	发送邮箱标识符寄存器
CAN_TMPx	发送邮箱属性寄存器
CAN_TMDATA0x	发送邮箱 data0 寄存器
CAN_TMDATA1x	发送邮箱 data1 寄存器
CAN_RFIFOMIx	接收 FIFO 邮箱标识符寄存器
CAN_RFIFOMPx	接收 FIFO 邮箱属性寄存器
CAN_RFIFOMDATA0x	接收 FIFO 邮箱 data0 寄存器
CAN_RFIFOMDATA1x	接收 FIFO 邮箱 data1 寄存器
CAN_FCTL	过滤器控制寄存器
CAN_FMCFG	过滤器模式配置寄存器
CAN_FSCFG	过滤器位宽配置寄存器
CAN_FAFIFO	过滤器关联 FIFO 寄存器
CAN_FW	过滤器激活寄存器
CAN_FxDATAy	过滤器(x)数据(y)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 3-73. CAN 库函数**

库函数名称	库函数描述
can_deinit	复位外设 CAN
can_init	初始化外设 CAN
can_filter_init	CAN 过滤器初始化
can_filter_mask_mode_init	CAN 过滤器掩码模式初始化
can_struct_para_init	CAN 外设库使用到的各类结构体初始化
can_monitor_mode_set	CAN 总线监听模式配置
can_fd_init	CAN FD 功能初始化
can_fd_function_enable	FD 功能使能
can_fd_function_disable	FD 功能关闭
can1_filter_start_bank	CAN1 过滤器序起始编号设置
can_debug_freeze_enable	CAN 调试冻结使能
can_debug_freeze_disable	CAN 调试冻结关闭
can_time_trigger_mode_enable	CAN 时间触发模式使能
can_time_trigger_mode_disable	CAN 时间触发模式关闭
can_message_transmit	CAN 传输报文
can_transmit_states	获取 CAN 传输状态

库函数名称	库函数描述
can_transmission_stop	CAN 邮箱停止发送
can_message_receive	CAN 接收报文
can_fifo_release	CAN 释放 FIFO
can_receive_message_length_get	获取 CAN 接收帧的数量
can_working_mode_set	CAN 工作模式设置
can_wakeup	从睡眠模式中唤醒 CAN
can_error_get	获取 CAN 总线错误
can_receive_error_number_get	获取 CAN 接收错误
can_transmit_error_number_get	获取 CAN 发送错误
can_interrupt_enable	CAN 中断使能
can_interrupt_disable	CAN 中断关闭
can_flag_get	获取 CAN 标志位状态
can_flag_clear	清除 CAN 标志位状态
can_interrupt_flag_get	获取 CAN 中断标志位状态
can_interrupt_flag_clear	清除 CAN 中断标志位状态

### 结构体 can\_parameter\_struct

表 3-74. 结构体 can\_parameter\_struct

成员名称	功能描述
working_mode	工作模式
resync_jump_width	再同步补偿宽度
time_segment_1	位段 1
time_segment_2	位段 2
time_triggered	时间触发通信模式
auto_bus_off_recovery	自动离线恢复
auto_wake_up	自动唤醒
auto_retrans	自动重传
rec_fifo_overwrite	接收 FIFO 满时覆盖
trans_fifo_order	发送 FIFO 顺序
prescaler	波特率分频系数

### 结构体 can\_transmit\_message\_struct

表 3-75. 结构体 can\_transmit\_message\_struct

成员名称	功能描述
tx_sfid	标准格式帧标识符
tx_efid	扩展格式帧标识符
tx_ff	帧格式：标准格式/扩展格式
tx_ft	帧类型：数据帧/远程帧
tx_dlen	数据长度

tx_data[8]	数据值
------------	-----

**表 3-76. 结构体 can\_transmit\_message\_struct（用于 CAN-FD）**

成员名称	功能描述
tx_sfid	标准格式帧标识符
tx_efid	扩展格式帧标识符
tx_ff	帧格式：标准格式/扩展格式
tx_ft	帧类型：数据帧/远程帧
tx_dlen	数据长度
tx_data[64]	数据值
fd_flag	FD帧标志位
fd_brs	位速率转换开关
fd_esi	错误状态指示

### 结构体 can\_receive\_message\_struct

**表 3-77. 结构体 can\_receive\_message\_struct**

成员名称	功能描述
rx_sfid	标准格式帧标识符
rx_efid	扩展格式帧标识符
rx_ff	帧格式：标准格式/扩展格式
rx_ft	帧类型：数据帧/远程帧
rx_dlen	数据长度
rx_data[64]	数据值
rx_fi	过滤器索引

**表 3-78. 结构体 can\_receive\_message\_struct（用于 CAN-FD）**

成员名称	功能描述
rx_sfid	标准格式帧标识符
rx_efid	扩展格式帧标识符
rx_ff	帧格式：标准格式/扩展格式
rx_ft	帧类型：数据帧/远程帧
rx_dlen	数据长度
rx_data[64]	数据值
rx_fi	过滤器索引
fd_flag	FD帧标志位
fd_brs	位速率转换开关
fd_esi	错误状态指示

### 结构体 can\_filter\_parameter\_struct

**表 3-79. 结构体 can\_filter\_parameter\_struct**

成员名称	功能描述
filter_list_high	过滤器列表数高位

filter_list_low	过滤器列表数低位
filter_mask_high	过滤器掩码数高位
filter_mask_low	过滤器掩码数低位
filter_fifo_number	接收FIFO编号
filter_number	过滤器索引号
filter_mode	过滤模式：列表模式/掩码模式
filter_bits	过滤器位宽
filter_enable	过滤器是否工作

### 结构体 can\_fd\_tdc\_struct

表 3-80. 结构体 can\_fd\_tdc\_struct（用于 CAN-FD）

成员名称	功能描述
tdc_mode	传输延迟补偿工作模式
tdc_filter	传输延迟补偿过滤器
tdc_offset	传输延迟补偿偏移

### 结构体 can\_fdframe\_struct

表 3-81. 结构体 can\_fdframe\_struct（用于 CAN-FD）

成员名称	功能描述
fd_frame	FD功能开关
excp_event_detect	协议异常事件检测功能
delay_compensation	传输延迟补偿
p_delay_compensation	传输延迟补偿配置结构体指针，详见 <a href="#">结构体can_fd_tdc_struct</a>
iso_bosch	ISO/Bosch模式选择
esi_mode	错误状态指示模式
data_resync_jump_width	数据域再同步补偿宽度
data_time_segment_1	数据域位段1
data_time_segment_2	数据域位段2
data_prescaler	数据域波特率预分频器

### 枚举类型 can\_interrupt\_flag\_enum

表 3-82. 枚举 can\_interrupt\_flag\_enum

枚举名称	枚举描述
CAN_INT_FLAG_SLPIF	进入睡眠工作模式的状态改变中断标志
CAN_INT_FLAG_WUIF	从睡眠工作模式唤醒的状态改变中断标志
CAN_INT_FLAG_ERRIF	错误中断标志
CAN_INT_FLAG_MTF2	邮箱 2 发送完成中断标志

枚举名称	枚举描述
CAN_INT_FLAG_MTF1	邮箱 1 发送完成中断标志
CAN_INT_FLAG_MTF0	邮箱 0 发送完成中断标志
CAN_INT_FLAG_RFO0	接收 FIFO0 溢出中断标志
CAN_INT_FLAG_RFF0	接收 FIFO0 满中断标志
CAN_INT_FLAG_RFL0	接收 FIFO0 非空中断标志
CAN_INT_FLAG_RFO1	接收 FIFO1 溢出中断标志
CAN_INT_FLAG_RFF1	接收 FIFO1 满中断标志
CAN_INT_FLAG_RFL1	接收 FIFO1 非空中断标志
CAN_INT_FLAG_ERRN	错误种类
CAN_INT_FLAG_BOERR	离线错误
CAN_INT_FLAG_PERR	被动错误
CAN_INT_FLAG_WERR	警告错误

### 枚举类型 `can_flag_enum`

表 3-83. 枚举 Enum `can_flag_enum`

枚举名称	枚举描述
CAN_FLAG_RXL	RX 引脚电平
CAN_FLAG_LASTRX	RX 引脚最近一次的采样值
CAN_FLAG_RS	接收状态
CAN_FLAG_TS	发送状态
CAN_FLAG_SLPIF	进入睡眠工作模式的状态改变中断标志
CAN_FLAG_WUIF	从睡眠工作模式唤醒的状态改变中断标志
CAN_FLAG_ERRIF	错误中断标志
CAN_FLAG_SLPWS	睡眠工作状态
CAN_FLAG_IWS	初始化工作状态
CAN_FLAG_TMLS2	在发送 FIFO 中邮箱 2 最后发送
CAN_FLAG_TMLS1	在发送 FIFO 中邮箱 1 最后发送
CAN_FLAG_TMLS0	在发送 FIFO 中邮箱 0 最后发送
CAN_FLAG_TME2	发送邮箱 2 空
CAN_FLAG_TME1	发送邮箱 1 空
CAN_FLAG_TME0	发送邮箱 0 空
CAN_FLAG_MTE2	邮箱 2 发送错误
CAN_FLAG_MTE1	邮箱 1 发送错误
CAN_FLAG_MTE0	邮箱 0 发送错误
CAN_FLAG_MTF2	邮箱 2 发送完成
CAN_FLAG_MTF1	邮箱 1 发送完成
CAN_FLAG_MTF0	邮箱 0 发送完成
CAN_FLAG_RFO0	接收 FIFO0 溢出
CAN_FLAG_RFF0	接收 FIFO0 满
CAN_INT_FLAG_RFL0	接收 FIFO0 非空中断标志
CAN_FLAG_RFO1	接收 FIFO1 溢出

枚举名称	枚举描述
CAN_FLAG_RFF1	接收 FIFO1 满
CAN_INT_FLAG_RFL1	接收 FIFO1 非空中断标志
CAN_FLAG_BOERR	离线错误
CAN_FLAG_PERR	被动错误
CAN_FLAG_WERR	警告错误

#### 枚举类型 `can_error_enum`

表 3-84. 枚举 `can_error_enum`

枚举名称	枚举描述
CAN_ERROR_NONE	无错误
CAN_ERROR_FILL	填充错误
CAN_ERROR_FORMATE	格式错误
CAN_ERROR_ACK	ACK 错误
CAN_ERROR_BITRECES SIVE	位隐性错误
CAN_ERROR_BITDOMIN ANTER	位显性错误
CAN_ERROR_CRC	CRC 错误
CAN_ERROR_SOFTWARE ECFG	软件设置错误

#### 枚举类型 `can_transmit_state_enum`

表 3-85. 枚举 `can_transmit_state_enum`

枚举名称	枚举描述
CAN_TRANSMIT_FAILED	CAN 发送错误
CAN_TRANSMIT_OK	CAN 发送成功
CAN_TRANSMIT_PENDING	CAN 发送挂起
CAN_TRANSMIT_NOMAILBO X	CAN 无可用的空邮箱

#### 枚举类型 `can_format_fifo_enum`

表 3-86. 枚举 `can_format_fifo_enum`

枚举名称	枚举描述
CAN_STANDARD_FIFO0	使用标准帧格式，FIFO0 存储
CAN_STANDARD_FIFO1	使用标准帧格式，FIFO1 存储
CAN_EXTENDED_FIFO0	使用扩展帧格式，FIFO0 存储
CAN_EXTENDED_FIFO1	使用扩展帧格式，FIFO1 存储

## 枚举类型 `can_struct_type_enum`

表 3-87. 枚举 `can_struct_type_enum`

枚举名称	枚举描述
<code>CAN_INIT_STRUCT</code>	CAN 参数初始化结构体
<code>CAN_FILTER_STRUCT</code>	CAN 过滤器结构体
<code>CAN_TX_MESSAGE_STRUCT</code>	CAN 发送消息结构体
<code>CAN_RX_MESSAGE_STRUCT</code>	CAN 接收消息结构体

## 函数 `can_deinit`

函数`can_deinit`描述见下表：

表 3-88. 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	复位外设CAN
先决条件	-
被调用函数	<code>rcu_periph_reset_enable/ rcu_periph_reset_disable</code>
输入参数{in}	
<code>can_periph</code>	CAN 外设
<code>CANx(x=0,1,2)</code>	CAN外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
can_deinit (CAN0);
```

## 函数 `can_struct_para_init`

函数`can_struct_para_init`描述见下表：

表 3-89. 函数 `can_struct_para_init`

函数名称	<code>can_struct_para_init</code>
函数原型	<code>void can_struct_para_init(can_struct_type_enum type, void* p_struct)</code>
功能描述	CAN 外设库使用到的各类结构体初始化
先决条件	-
被调用函数	-
输入参数{in}	

type	需要初始化的结构体类型参考 <a href="#">表 3-87. 枚举 can_struct_type_enum</a> ，仅可选择 唯一参数
输出参数{out}	
p_struct	对应的需要初始化的结构体指针
返回值	
-	-

例如：

```
can_parameter_struct can_init;
```

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

### 函数 can\_init

函数can\_init描述见下表：

表 3-90. 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	初始化外设CAN
先决条件	can_struct_para_init()
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN外设选择
输入参数{in}	
can_parameter_init	初始化结构体，结构体成员参考 <a href="#">表3-74. 结构体can_parameter_struct</a>
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS / ERROR

例如：

```
/* CAN0 initialize*/
```

```
can_init (CAN0);
```

### 函数 can\_filter\_init

函数can\_filter\_init描述见下表：

表 3-91. 函数 can\_filter\_init

函数名称	can_filter_init
函数原型	void can_filter_init(uint32_t can_periph,can_filter_parameter_struct*

	can_filter_parameter_init;
功能描述	CAN 过滤器初始化
先决条件	can_struct_para_init()
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,2)	CAN 外设选择
输入参数{in}	
can_filter_paramet er_init	过滤器初始化结构体，结构体成员参考 <a href="#">表 3-79. 结构体 can_filter_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
can_filter_init(CAN2, &can_filter);
```

### 函数 can\_filter\_mask\_mode\_init

函数can\_filter\_mask\_mode\_init描述见下表：

**表 3-92. 函数 can\_filter\_mask\_mode\_init**

函数名称	can_filter_mask_mode_init
函数原型	void can_filter_mask_mode_init(uint32_t id, uint32_t mask, can_format_fifo_enum format_fifo, uint16_t filter_number)
功能描述	CAN 过滤器掩码模式初始化
先决条件	-
被调用函数	can_filter_init()
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,2)	CAN 外设选择
输入参数{in}	
id	取值范围（0x00000000 - 0xFFFFFFFF）
输入参数{in}	
mask	取值范围（0x00000000 - 0xFFFFFFFF）
输入参数{in}	
format_fifo	帧格式及 FIFO 选择参考 <a href="#">表 3-86. 枚举 can_format_fifo_enum</a> ，仅可选择唯一参数
输入参数{in}	
filter_number	使用过滤器序号，取值范围（0x00 - 0x1C）

输出参数{out}	
-	-
返回值	
-	-

例如：

```
can_filter_mask_mode_init(CAN0,0x11, 0x11, CAN_STANDARD_FIFO0, 0);
```

### 函数 can\_monitor\_mode\_set

函数can\_monitor\_mode\_set描述见下表：

表 3-93. 函数 can\_monitor\_mode\_set

函数名称	can_monitor_mode_set
函数原型	ErrStatus can_monitor_mode_set(uint32_t can_periph, uint8_t mode)
功能描述	CAN总线监听模式配置
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1,2)	CAN外设选择
输入参数{in}	
mode	监听模式，仅可选择唯一参数
CAN_NORMAL_MODE	正常模式
CAN_LOOPBACK_MODE	回环通讯模式
CAN_SILENT_MODE	静默通讯模式
CAN_SILENT_LOOPBACK_MODE	静默回环通讯模式
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS / ERROR

例如：

```
can_monitor_mode_set(CAN0, CAN_NORMAL_MODE);
```

### 函数 can\_fd\_init（用于 CAN-FD）

函数can\_fd\_init描述见下表：

表 3-94. 函数 can\_fd\_init

函数名称	can_fd_init
------	-------------

函数原型	ErrStatus can_fd_init(uint32_t can_periph, can_fdframe_struct* can_fdframe_init);
功能描述	初始化外设 CAN FD 功能
先决条件	can_struct_para_init()
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
can_fdframe_init	初始化 FD 功能结构体，结构体成员参考 <a href="#">表 3-81. 结构体 can_fdframe_struct (用于 CAN-FD)</a>
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS / ERROR

例如：

```
/* CAN0 FD initialize*/
```

```
can_fdframe_struct fd_init_para;
```

```
can_fd_init(CAN0, &fd_init_para);
```

### 函数 can\_fd\_function\_enable（用于 CAN-FD）

函数can\_fd\_function\_enable描述见下表：

表 3-95. 函数 can\_fd\_function\_enable

函数名称	can_fd_function_enable
函数原型	void can_fd_function_enable(uint32_t can_periph)
功能描述	CAN FD功能使能
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1,2)	CAN外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
can_fd_function_enable(CAN0);
```

## 函数 can\_fd\_function\_disable（用于 CAN-FD）

函数can\_fd\_function\_disable描述见下表：

**表 3-96. 函数 can\_fd\_function\_disable**

函数名称	can_fd_function_disable
函数原型	void can_fd_function_disable(uint32_t can_periph)
功能描述	CAN FD功能关闭
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1,2)	CAN外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
can_fd_function_disable(CAN0);
```

## 函数 can1\_filter\_start\_bank

函数can1\_filter\_start\_bank描述见下表：

**表 3-97. 函数 can1\_filter\_start\_bank**

函数名称	can1_filter_start_bank
函数原型	void can1_filter_start_bank(uint8_t start_bank);
功能描述	CAN1过滤器序起始编号设置
先决条件	-
被调用函数	-
输入参数{in}	
start_bank	CAN1过滤器序起始编号
1..27	可选的编号
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
can1_filter_start_bank (15);
```

## 函数 can\_debug\_freeze\_enable

函数can\_debug\_freeze\_enable描述见下表：

**表 3-98. 函数 can\_debug\_freeze\_enable**

函数名称	can_debug_freeze_enable
函数原型	void can_debug_freeze_enable(uint32_t can_periph);
功能描述	CAN调试冻结使能
先决条件	-
被调用函数	dbg_periph_enable
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

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

## 函数 can\_debug\_freeze\_disable

函数can\_debug\_freeze\_disable描述见下表：

**表 3-99. 函数 can\_debug\_freeze\_disable**

函数名称	can_debug_freeze_disable
函数原型	void can_debug_freeze_disable(uint32_t can_periph);
功能描述	CAN调试冻结关闭
先决条件	-
被调用函数	dbg_periph_disable
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CAN0 debug freeze */
can_debug_freeze_disable (CAN0);
```

## 函数 can\_time\_trigger\_mode\_enable

函数can\_time\_trigger\_mode\_enable描述见下表：

**表 3-100. 函数 can\_time\_trigger\_mode\_enable**

函数名称	can_time_trigger_mode_enable
函数原型	void can_time_trigger_mode_enable(uint32_t can_periph);
功能描述	CAN时间触发模式使能
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

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

## 函数 can\_time\_trigger\_mode\_disable

函数can\_time\_trigger\_mode\_disable描述见下表：

**表 3-101. 函数 can\_time\_trigger\_mode\_disable**

函数名称	can_time_trigger_mode_disable
函数原型	void can_time_trigger_mode_disable(uint32_t can_periph);
功能描述	CAN时间触发模式关闭
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CAN0 time trigger mode */
can_time_trigger_mode_disable (CAN0);
```

函数 `can_message_transmit`

函数 `can_message_transmit` 描述见下表：

表 3-102. 函数 `can_message_transmit`

函数名称	<code>can_message_transmit</code>
函数原型	<code>uint8_t can_message_transmit(uint32_t can_periph, can_transmit_message_struct* transmit_message);</code>
功能描述	CAN 传输报文
先决条件	<code>can_struct_para_init()</code>
被调用函数	-
输入参数{in}	
<code>can_periph</code>	CAN 外设
<code>CANx(x=0,1,2)</code>	CAN 外设选择
输入参数{in}	
<code>transmit_message</code>	CAN 的报文发送结构体，结构体成员参考 <a href="#">表 3-75. 结构体 <code>can_transmit_message_struct</code></a> CAN-FD 的报文发送结构体，结构体成员参考 <a href="#">表 3-76. 结构体 <code>can_transmit_message_struct (用于 CAN-FD)</code></a>
输出参数{out}	
-	-
返回值	
<code>uint8_t</code>	<code>0x00-0x03</code>

例如：

```
/* CAN0 transmit message and return the mailbox number */
```

```
uint8_t transmit_mailbox = 0;
```

```
transmit_mailbox = can_message_transmit(CAN0, &transmit_message);
```

函数 `can_transmit_states`

函数 `can_transmit_states` 描述见下表：

表 3-103. 函数 `can_transmit_states`

函数名称	<code>can_transmit_states</code>
函数原型	<code>can_transmit_state_enum can_transmit_states(uint32_t can_periph, uint8_t mailbox_number);</code>
功能描述	获取 CAN 传输状态
先决条件	-
被调用函数	-
输入参数{in}	
<code>can_periph</code>	CAN 外设
<code>CANx(x=0,1,2)</code>	CAN 外设选择

输入参数{in}	
<b>mailbox_number</b>	邮箱标号
<i>CAN_MAILBOXx</i>	CAN_MAILBOXx(x=0,1,2)
输出参数{out}	
-	-
返回值	
<b>can_transmit_state_enum</b>	返回值参考 <a href="#">表 3-85. 枚举 can_transmit_state_enum</a>

例如：

```
/* CAN0 mailbox0 transmit state */
uint8_t transmit_state = 0;

transmit_state = can_transmit_states (CAN0, CAN_MAILBOX0);
```

### 函数 can\_transmission\_stop

函数can\_transmission\_stop描述见下表：

**表 3-104. 函数 can\_transmission\_stop**

<b>函数名称</b>	can_transmission_stop
<b>函数原型</b>	void can_transmission_stop(uint32_t can_periph, uint8_t mailbox_number);
<b>功能描述</b>	CAN邮箱停止发送
<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(x=0,1,2)</i>	CAN外设选择
输入参数{in}	
<b>mailbox_number</b>	邮箱标号
<i>CAN_MAILBOXx</i>	CAN_MAILBOXx(x=0,1,2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* stop CAN0 mailbox0 transmission */

can_transmission_stop (CAN0, CAN_MAILBOX0);
```

### 函数 can\_message\_receive

函数can\_message\_receive描述见下表：

表 3-105. 函数 can\_message\_receive

函数名称	can_message_receive
函数原型	void can_message_receive(uint32_t can_periph, uint8_t fifo_number, can_receive_message_struct* receive_message);
功能描述	CAN 接收报文
先决条件	can_struct_para_init()
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
fifo_number	FIFO 编号
CAN_FIFOx	CAN_FIFOx(x=0,1)
输入参数{in}	
receive_message	CAN 接收报文结构体，结构体成员参考 <a href="#">表 3-77. 结构体 can_receive_message_struct</a> CAN-FD 接收报文结构体，结构体成员参考 <a href="#">表 3-78. 结构体 can_receive_message_struct (用于 CAN-FD)</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CAN0 FIFO0 receive message */
```

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

### 函数 can\_fifo\_release

函数can\_fifo\_release描述见下表：

表 3-106. 函数 can\_fifo\_release

函数名称	can_fifo_release
函数原型	void can_fifo_release(uint32_t can_periph, uint8_t fifo_number);
功能描述	CAN释放FIFO
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN外设选择
输入参数{in}	
fifo_number	FIFO编号
CAN_FIFOx	CAN_FIFOx(x=0,1)

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CAN0 release FIFO0 */
```

```
can_fifo_release (CAN0, CAN_FIFO0);
```

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

函数can\_receive\_message\_length\_get描述见下表：

表 3-107. 函数 can\_receive\_message\_length\_get

函数名称	can_receive_message_length_get
函数原型	uint8_t can_receive_message_length_get(uint32_t can_periph, uint8_t fifo_number);
功能描述	获取CAN接收帧的数量
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN外设选择
输入参数{in}	
fifo_number	FIFO编号
CAN_FIFOx	CAN_FIFOx(x=0,1)
输出参数{out}	
-	-
返回值	
uint8_t	0..3

例如：

```
/* CAN0 FIFO0 receive message length */
```

```
uint8_t frame_number = 0;
```

```
frame_number = can_receive_message_length_get (CAN0, CAN_FIFO0);
```

### 函数 can\_working\_mode\_set

函数can\_working\_mode\_set描述见下表：

表 3-108. 函数 can\_working\_mode\_set

函数名称	can_working_mode_set
函数原型	ErrStatus can_working_mode_set(uint32_t can_periph, uint8_t working_mode);

功能描述	CAN工作模式设置
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN外设选择
输入参数{in}	
can_working_mode	模式选择
CAN_MODE_INITIALIZE	初始化模式
CAN_MODE_NORMAL	正常模式
CAN_MODE_SLEEP	睡眠模式
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS / ERROR

例如：

```
/* set CAN0 working at initialize mode */
```

```
can_working_mode_set (CAN0, CAN_MODE_INITIALIZE);
```

## 函数 can\_wakeup

函数can\_wakeup描述见下表：

表 3-109. 函数 can\_wakeup

函数名称	can_wakeup
函数原型	ErrStatus can_wakeup(uint32_t can_periph);
功能描述	从睡眠模式中唤醒CAN
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN外设选择
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS / ERROR

例如：

```
/* wake up CAN0 */
```

```
can_wakeup (CAN0);
```

### 函数 can\_error\_get

函数can\_error\_get描述见下表:

表 3-110. 函数 can\_error\_get

函数名称	can_error_get
函数原型	can_error_enum can_error_get(uint32_t can_periph);
功能描述	获取 CAN 总线错误
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
can_error_enum	返回值参考 <a href="#">表 3-84. 枚举 can_error_enum</a>

例如:

```
/* get CAN0 error type */
```

```
can_error_enum err_type;
```

```
err_type = can_error_get (CAN0);
```

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

函数can\_receive\_error\_number\_get描述见下表:

表 3-111. 函数 can\_receive\_error\_number\_get

函数名称	can_receive_error_number_get
函数原型	uint8_t can_receive_error_number_get(uint32_t can_periph);
功能描述	获取CAN接收错误
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN外设选择
输出参数{out}	
-	-
返回值	
uint8_t	0..255

例如:

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

```
uint8_t error_num;
```

```
error_num = can_receive_error_number_get (CAN0);
```

### 函数 can\_transmit\_error\_number\_get

函数can\_transmit\_error\_number\_get描述见下表:

表 3-112. 函数 can\_transmit\_error\_number\_get

函数名称	can_transmit_error_number_get
函数原型	uint8_t can_transmit_error_number_get(uint32_t can_periph);
功能描述	获取CAN发送错误
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN外设选择
输出参数{out}	
-	-
返回值	
uint8_t	0..255

例如:

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

```
uint8_t error_num;
```

```
error_num = can_transmit_error_number_get (CAN0);
```

### 函数 can\_interrupt\_enable

函数can\_interrupt\_enable描述见下表:

表 3-113. 函数 can\_interrupt\_enable

函数名称	can_interrupt_enable
函数原型	void can_interrupt_enable(uint32_t can_periph, uint32_t interrupt);
功能描述	CAN中断使能
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN外设选择
输入参数{in}	

interrupt	中断类型
CAN_INT_TME	发送邮箱空中断使能
CAN_INT_RFNE0	接收FIFO0非空中断使能
CAN_INT_RFF0	接收FIFO0满中断使能
CAN_INT_RFO0	接收FIFO0溢出中断使能
CAN_INT_RFNE1	接收FIFO1非空中断使能
CAN_INT_RFF1	接收FIFO1满中断使能
CAN_INT_RFO1	接收FIFO1溢出中断使能
CAN_INT_WERR	警告错误中断使能
CAN_INT_PERR	被动错误中断使能
CAN_INT_BO	离线中断使能
CAN_INT_ERRN	错误种类中断使能
CAN_INT_ERR	错误中断使能
CAN_INT_WU	唤醒中断使能
CAN_INT_SLPW	睡眠中断使能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* CAN0 transmit mailbox empty interrupt enable */
```

```
can_interrupt_enable (CAN0, CAN_INT_TME);
```

### 函数 can\_interrupt\_disable

函数can\_interrupt\_disable描述见下表:

表 3-114. 函数 can\_interrupt\_disable

函数名称	can_interrupt_disable
函数原型	void can_interrupt_disable(uint32_t can_periph, uint32_t interrupt);
功能描述	CAN中断关闭
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN外设选择
输入参数{in}	
interrupt	中断类型
CAN_INT_TME	发送邮箱空中断使能
CAN_INT_RFNE0	接收FIFO0非空中断使能
CAN_INT_RFF0	接收FIFO0满中断使能
CAN_INT_RFO0	接收FIFO0溢出中断使能

CAN_INT_RFNE1	接收FIFO1非空中断使能
CAN_INT_RFF1	接收FIFO1满中断使能
CAN_INT_RFO1	接收FIFO1溢出中断使能
CAN_INT_WERR	警告错误中断使能
CAN_INT_PERR	被动错误中断使能
CAN_INT_BO	离线中断使能
CAN_INT_ERRN	错误种类中断使能
CAN_INT_ERR	错误中断使能
CAN_INT_WU	唤醒中断使能
CAN_INT_SLPW	睡眠中断使能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* CAN0 transmit mailbox empty interrupt disable */
```

```
can_interrupt_disable (CAN0, CAN_INT_TME);
```

### 函数 can\_flag\_get

函数can\_flag\_get描述见下表:

表 3-115. 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 标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
flag	CAN 标志位参考 <a href="#">表 3-83. 枚举 Enum can_flag_enum</a>
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如:

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

```
can_flag_get (CAN0, CAN_FLAG_MTF0);
```

函数 `can_flag_clear`

函数 `can_flag_clear` 描述见下表：

表 3-116. 函数 `can_flag_clear`

函数名称	<code>can_flag_clear</code>
函数原型	<code>void can_flag_clear(uint32_t can_periph, can_flag_enum flag);</code>
功能描述	清除 CAN 标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
<code>can_periph</code>	CAN 外设
<code>CANx(x=0,1,2)</code>	CAN 外设选择, CAN1 仅适用于 GD32F30X_CL
输入参数{in}	
<code>flag</code>	CAN 标志位参考 <a href="#">表 3-83. 枚举 Enum can_flag_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 `can_interrupt_flag_get`

函数 `can_interrupt_flag_get` 描述见下表：

表 3-117. 函数 `can_interrupt_flag_get`

函数名称	<code>can_interrupt_flag_get</code>
函数原型	<code>FlagStatus can_interrupt_flag_get(uint32_t can_periph, can_interrupt_flag_enum flag);</code>
功能描述	获取 CAN 中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
<code>can_periph</code>	CAN 外设
<code>CANx(x=0,1,2)</code>	CAN 外设选择
输入参数{in}	
<code>flag</code>	CAN 中断标志位参考 <a href="#">表 3-82. 枚举 can_interrupt_flag_enum</a>
输出参数{out}	
-	-
返回值	
<code>FlagStatus</code>	SET / RESET

例如：

```
/* get CAN0 mailbox 0 transmit finished interrupt flag */
can_interrupt_flag_get (CAN0, CAN_INT_FLAG_MTF0);
```

### 函数 can\_interrupt\_flag\_clear

函数can\_interrupt\_flag\_clear描述见下表：

表 3-118. 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 flag);
功能描述	清除 CAN 中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
flag	CAN 中断标志位参考 <a href="#">表 3-82. 枚举 can_interrupt_flag_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

## 3.5. 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[3.5.1](#)描述了CRC的寄存器列表，章节[3.5.2](#)对CRC库函数进行说明。

### 3.5.1. 外设寄存器说明

CRC寄存器列表如下表所示：

表 3-119. 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

### 3.5.2. 外设库函数说明

CRC库函数列表如下表所示：

表 3-120. CRC 库函数

库函数名称	库函数描述
crc_deinit	复位CRC计算单元
crc_data_register_reset	根据数据寄存器的复位值（0xFFFFFFFF）复位数据寄存器
crc_reverse_output_data_enable	使能输出数据翻转功能
crc_reverse_output_data_disable	禁能输出数据翻转功能
crc_input_data_reverse_config	配置输入数据翻转功能
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_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计算一个32位数据
crc_block_data_calculate	CRC计算一个32位数组

#### 函数 crc\_deinit

函数crc\_deinit描述见下表：

表 3-121. 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\_data\_register\_reset

函数crc\_data\_register\_reset描述见下表：

表 3-122. 函数 crc\_data\_register\_reset

函数名称	crc_data_register_reset
函数原形	void crc_data_register_reset(void);
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\_reverse\_output\_data\_enable

函数crc\_reverse\_output\_data\_enable描述见下表：

表 3-123. 函数 crc\_reverse\_output\_data\_enable

函数名称	crc_reverse_output_data_enable
函数原形	void crc_reverse_output_data_enable (void);

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 3-124. 函数 crc\_reverse\_output\_data\_disable

函数名称	crc_reverse_output_data_disable
函数原形	void crc_reverse_output_data_disable (void);
功能描述	禁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\_input\_data\_reverse\_config

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

表 3-125. 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
<b>CRC_INPUT_DATA_NOT</b>	输入数据不翻转
<b>CRC_INPUT_DATA_BYTE</b>	输入数据按字节翻转
<b>CRC_INPUT_DATA_HALFWORD</b>	输入数据按半字翻转
<b>CRC_INPUT_DATA_WORD</b>	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\_data\_register\_read

函数crc\_data\_register\_read描述见下表：

表 3-126. 函数 crc\_data\_register\_read

函数名称	crc_data_register_read
函数原形	uint32_t crc_data_register_read(void);
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	
<b>uint32_t</b>	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 3-127. 函数 `crc_free_data_register_read`

函数名称	<code>crc_free_data_register_read</code>
函数原形	<code>uint8_t crc_free_data_register_read(void);</code>
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	
<code>uint8_t</code>	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 3-128. 函数 `crc_free_data_register_write`

函数名称	<code>crc_free_data_register_write</code>
函数原形	<code>void crc_free_data_register_write(uint8_t free_data);</code>
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}	
<code>free_data</code>	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 3-129. 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\_polynomial\_size\_set

函数crc\_polynomial\_size\_set描述见下表：

表 3-130. 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}	
poly_size	多项式的长度
CRC_CTL_PS_32	32位多项式值用于CRC计算
CRC_CTL_PS_16	16位多项式值用于CRC计算
CRC_CTL_PS_8	8位多项式值用于CRC计算
CRC_CTL_PS_7	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 3-131. 函数 crc\_polynomial\_set

函数名称	crc_polynomial_set
函数原形	void crc_polynomial_set(uint32_t poly)
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}	
poly	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 3-132. 函数 crc\_single\_data\_calculate

函数名称	crc_single_data_calculate
函数原形	uint32_t crc_single_data_calculate(uint32_t sdata);
功能描述	CRC计算一个32位数据
先决条件	-
被调用函数	-
输入参数{in}	
sdata	设定的32位数据
输出参数{out}	
-	-

返回值	
uint32_t	32位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);
```

### 函数 crc\_block\_data\_calculate

函数crc\_block\_data\_calculate描述见下表:

表 3-133. 函数 crc\_block\_data\_calculate

函数名称	crc_block_data_calculate
函数原形	uint32_t crc_block_data_calculate(uint32_t array[], uint32_t size);
功能描述	CRC计算一个32位数组
先决条件	-
被调用函数	-
输入参数{in}	
array	32位数据数组的指针
输入参数{in}	
size	数据长度
输出参数{out}	
-	-
返回值	
uint32_t	32位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((uint32_t *) data_buffer, BUFFER_SIZE);
```

## 3.6. CTC

CTC模块基于外部高精度的参考信号源来校准IRC48M的时钟频率，通过自动的或手动的调整

校准值,以得到一个精准的IRC48M时钟。章节[3.6.1](#)描述了CTC的寄存器列表,章节[3.6.2](#)对CTC库函数进行说明。

### 3.6.1. 外设寄存器说明

CTC寄存器列表如下表所示:

**表 3-134. CTC 寄存器**

寄存器名称	寄存器描述
CTC_CTL0	CTC控制寄存器0
CTC_CTL1	CTC控制寄存器1
CTC_STAT	CTC状态寄存器
CTC_INTC	CTC中断清除寄存器

### 3.6.2. 外设库函数说明

CTC库函数列表如下表所示:

**表 3-135. CTC 库函数**

库函数名称	库函数描述
ctc_deinit	复位CTC单元
ctc_counter_enable	使能CTC校准
ctc_counter_disable	禁能CTC校准
ctc_irc48m_trim_value_config	配置IRC48M时钟校准值
ctc_software_refsource_pulse_generate	产生CTC参考时钟源同步脉冲
ctc_hardware_trim_mode_config	CTC硬件自动校准模式配置
ctc_refsource_polarity_config	CTC参考信号源时钟极性配置
ctc_refsource_signal_select	CTC参考信号源选择
ctc_refsource_prescaler_config	CTC参考信号源分频配置
ctc_clock_limit_value_config	CTC时钟校准时基限值设置
ctc_counter_reload_value_config	CTC计数器重载值配置
ctc_counter_capture_value_read	读取CTC计数器捕获值
ctc_counter_direction_read	读取CTC校准时钟计数方向
ctc_counter_reload_value_read	读取CTC计数器重载值
ctc_irc48m_trim_value_read	读取IRC48M校准值
ctc_flag_get	CTC状态标志获取
ctc_flag_clear	CTC状态标志清除
ctc_interrupt_enable	CTC中断使能
ctc_interrupt_disable	CTC中断禁能
ctc_interrupt_flag_get	CTC中断标志获取
ctc_interrupt_flag_clear	CTC中断标志清除

## 函数 ctc\_deinit

函数ctc\_deinit描述见下表：

**表 3-136. 函数 ctc\_deinit**

函数名称	ctc_deinit
函数原形	void ctc_deinit (void);
功能描述	复位CTC单元
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 CTC */
```

```
ctc_deinit();
```

## 函数 ctc\_counter\_enable

函数ctc\_counter\_enable描述见下表：

**表 3-137. 函数 ctc\_counter\_enable**

函数名称	ctc_counter_enable
函数原形	void ctc_counter_enable (void);
功能描述	使能CTC校准计数器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
ctc_counter_enable ();
```

## 函数 ctc\_counter\_disable

函数ctc\_counter\_disable描述见下表：

**表 3-138. 函数 ctc\_counter\_disable**

函数名称	ctc_counter_disable
函数原形	void ctc_counter_disable (void);
功能描述	禁能CTC计数器校准
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 CTC trim counter */
```

```
ctc_counter_disable ();
```

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

函数ctc\_irc48m\_trim\_value\_config描述见下表：

**表 3-139. 函数 ctc\_irc48m\_trim\_value\_config**

函数名称	ctc_irc48m_trim_value_config
函数原形	void ctc_irc48m_trim_value_config(uint8_t trim_value);
功能描述	配置IRC48M时钟校准值
先决条件	-
被调用函数	-
输入参数{in}	
trim_value	0~63
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

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

函数ctc\_software\_refsource\_pulse\_generate描述见下表：

表 3-140. 函数 ctc\_software\_refsource\_pulse\_generate

函数名称	ctc_software_refsource_pulse_generate
函数原形	void ctc_software_refsource_pulse_generate(void);
功能描述	产生CTC参考时钟源同步脉冲
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
ctc_software_refsource_pulse_generate ();
```

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

函数ctc\_hardware\_trim\_mode\_config描述见下表：

表 3-141. 函数 ctc\_hardware\_trim\_mode\_config

函数名称	ctc_hardware_trim_mode_config
函数原形	void ctc_hardware_trim_mode_config(uint32_t hardmode);
功能描述	配置硬件自动校准
先决条件	-
被调用函数	-
输入参数{in}	
<b>hardmode</b>	硬件校准开启还是关闭
CTC_HARDWARE_TRIM_MODE_ENABLE	硬件校准开启
CTC_HARDWARE_TRIM_MODE_DISABLE	硬件校准关闭
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

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

```
ctc_hardware_trim_mode_config (CTC_HARDWARE_TRIM_MODE_ENABLE);
```

### 函数 ctc\_refsource\_polarity\_config

函数ctc\_refsource\_polarity\_config描述见下表:

表 3-142. 函数 ctc\_refsource\_polarity\_config

函数名称	ctc_refsource_polarity_config
函数原形	void ctc_refsource_polarity_config(uint32_t polarity);
功能描述	CTC参考时钟极性配置
先决条件	-
被调用函数	-
输入参数{in}	
polarity	时钟极性
CTC_REFSOURCE_POLARITY_FALLING	参考信号源的同步极性为下降沿
CTC_REFSOURCE_POLARITY_RISING	参考信号源的同步极性为上升沿
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
ctc_refsource_polarity_config (CTC_REFSOURCE_POLARITY_RISING);
```

### 函数 ctc\_refsource\_signal\_select

函数ctc\_refsource\_signal\_select描述见下表:

表 3-143. 函数 ctc\_refsource\_signal\_select

函数名称	ctc_refsource_signal_select
函数原形	void ctc_refsource_signal_select(uint32_t refs);
功能描述	CTC参考信号源选择
先决条件	-
被调用函数	-
输入参数{in}	

refs	参考信号源
CTC_REFSOURCE_GPIO	选择GPIO输入信号
CTC_REFSOURCE_LXTAL	选择LXTAL时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
ctc_refsource_signal_select (CTC_REFSOURCE_LXTAL);
```

### 函数 ctc\_refsource\_prescaler\_config

函数ctc\_refsource\_prescaler\_config描述见下表：

表 3-144. 函数 ctc\_refsource\_prescaler\_config

函数名称	ctc_refsource_prescaler_config
函数原形	void ctc_refsource_prescaler_config(uint32_t prescaler);
功能描述	参考信号源的分频设置
先决条件	-
被调用函数	-
输入参数{in}	
prescaler	分频系数
CTC_REFSOURCE_PSC_OFF	参考信号不分频
CTC_REFSOURCE_PSC_DIV2	参考信号2分频
CTC_REFSOURCE_PSC_DIV4	参考信号4分频
CTC_REFSOURCE_PSC_DIV8	参考信号8分频
CTC_REFSOURCE_PSC_DIV16	参考信号16分频
CTC_REFSOURCE_PSC_DIV32	参考信号32分频
CTC_REFSOURCE_PSC_DIV64	参考信号64分频
CTC_REFSOURCE_PSC_DIV128	参考信号128分频
输出参数{out}	

-	-
返回值	
-	-

例如：

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

```
ctc_refsource_prescaler_config(CTC_REFSOURCE_PSC_DIV2);
```

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

函数ctc\_clock\_limit\_value\_config描述见下表：

表 3-145. 函数 ctc\_clock\_limit\_value\_config

函数名称	ctc_clock_limit_value_config
函数原形	void ctc_clock_limit_value_config(uint8_t limit_value);
功能描述	CTC时钟校准时基限值设置
先决条件	-
被调用函数	-
输入参数{in}	
limit_value	0x00 - 0xFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure clock trim base limit value */
```

```
ctc_clock_limit_value_config (0x1F);
```

### 函数 ctc\_counter\_reload\_value\_config

函数ctc\_counter\_reload\_value\_config描述见下表：

表 3-146. 函数 ctc\_counter\_reload\_value\_config

函数名称	ctc_counter_reload_value_config
函数原形	void ctc_counter_reload_value_config(uint16_t reload_value);
功能描述	CTC计数器重载值设置
先决条件	-
被调用函数	-
输入参数{in}	
reload_value	0x0000 - 0xFFFF
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

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

### 函数 `ctc_counter_capture_value_read`

函数`ctc_counter_capture_value_read`描述见下表：

表 3-147. 函数 `ctc_counter_capture_value_read`

函数名称	<code>ctc_counter_capture_value_read</code>
函数原形	<code>uint16_t ctc_counter_capture_value_read(void);</code>
功能描述	读取计数器捕获值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<code>uint16_t</code>	读取计数器捕获值(0x0000 - 0xFFFF)

例如：

```
/* read CTC counter capture value */  
  
uint16_t ctc_value = 0;  
  
ctc_value = ctc_counter_capture_value_read ();
```

### 函数 `ctc_counter_direction_read`

函数`ctc_counter_direction_read`描述见下表：

表 3-148. 函数 `ctc_counter_direction_read`

函数名称	<code>ctc_counter_direction_read</code>
函数原形	<code>FlagStatus ctc_counter_direction_read(void);</code>
功能描述	读取CTC校准时钟计数方向
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

FlagStatus	SET(向下计数) / RESET(向上计数)
------------	-------------------------

例如:

```
/* read ctc counter direction */
```

```
FlagStatus ctc_direction = SET;
```

```
ctc_direction = ctc_counter_direction_read ();
```

### 函数 ctc\_counter\_reload\_value\_read

函数ctc\_counter\_reload\_value\_read描述见下表:

表 3-149. 函数 ctc\_counter\_reload\_value\_read

函数名称	ctc_counter_reload_value_read
函数原形	uint16_t ctc_counter_reload_value_read(void);
功能描述	读取CTC计数器重载值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	读取计数器重载值的16位数据 (0x0000 - 0xFFFF)

例如:

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

```
uint16_t ctc_reload_value = 0;
```

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

### 函数 ctc\_irc48m\_trim\_value\_read

函数ctc\_irc48m\_trim\_value\_read描述见下表:

表 3-150. 函数 ctc\_irc48m\_trim\_value\_read

函数名称	ctc_irc48m_trim_value_read
函数原形	uint8_t ctc_irc48m_trim_value_read(void);
功能描述	读IRC48M校准值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
uint8_t	6位IRC48M校准值 (0-63)

例如:

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

uint8_t ctc_trim_value = 0;

ctc_trim_value = ctc_irc48m_trim_value_read ();
```

### 函数 ctc\_flag\_get

函数ctc\_flag\_get描述见下表:

表 3-151. 函数 ctc\_flag\_get

函数名称	ctc_flag_get
函数原形	FlagStatus ctc_flag_get (uint32_t flag);
功能描述	获取CTC状态标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	CTC状态标志
CTC_FLAG_CKOK	时钟校准完成标志位
CTC_FLAG_CKWARN	时钟校准警告中断标志位
CTC_FLAG_ERR	错误中断标志位
CTC_FLAG_EREFS	期望参考信号中断标志位
CTC_FLAG_CKERR	时钟校准错误位
CTC_FLAG_REFMISS	参考同步脉冲信号丢失
CTC_FLAG_TRIMERR	校准值错误位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get CTC flag status */

FlagStatus state = ctc_flag_get (CTC_FLAG_CKOK);
```

### 函数 ctc\_flag\_clear

函数ctc\_flag\_clear描述见下表:

表 3-152. 函数 `ctc_flag_clear`

函数名称	<code>ctc_flag_clear</code>
函数原形	<code>void ctc_flag_clear (uint32_t flag);</code>
功能描述	清除CTC状态标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	CTC状态标志
<code>CTC_FLAG_CKOK</code>	时钟校准完成标志位
<code>CTC_FLAG_CKWARN</code>	时钟校准警告中断标志位
<code>CTC_FLAG_ERR</code>	错误中断标志位
<code>CTC_FLAG_EREFS</code>	期望参考信号中断标志位
<code>CTC_FLAG_CKERR</code>	时钟校准错误位
<code>CTC_FLAG_REFMISS</code>	参考同步脉冲信号丢失
<code>CTC_FLAG_TRIMERR</code>	校准值错误位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
ctc_flag_clear (CTC_FLAG_CKOK);
```

### 函数 `ctc_interrupt_enable`

函数`ctc_interrupt_enable`描述见下表：

表 3-153. 函数 `ctc_interrupt_enable`

函数名称	<code>ctc_interrupt_enable</code>
函数原形	<code>void ctc_interrupt_enable(uint32_t interrupt);</code>
功能描述	使能外设CTC中断
先决条件	-
被调用函数	-
输入参数{in}	
<b>interrupt</b>	CTC中断
<code>CTC_INT_CKOK</code>	时钟校准完成中断
<code>CTC_INT_CKWARN</code>	时钟校准警告中断

<i>CTC_INT_ERR</i>	错误中断
<i>CTC_INT_EREFP</i>	期望参考信号中断
输出参数{out}	
-	-
返回值	
-	-

例如:

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

### 函数 **ctc\_interrupt\_disable**

函数ctc\_interrupt\_disable描述见下表:

**表 3-154. 函数 ctc\_interrupt\_disable**

函数名称	ctc_interrupt_disable
函数原形	void ctc_interrupt_disable(uint32_t interrupt);
功能描述	禁能外设CTC中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	CTC中断
<i>CTC_INT_CKOK</i>	时钟校准完成中断
<i>CTC_INT_CKWARN</i>	时钟校准警告中断
<i>CTC_INT_ERR</i>	错误中断
<i>CTC_INT_EREFP</i>	期望参考信号中断
输出参数{out}	
-	-
返回值	
-	-

例如:

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

### 函数 **ctc\_interrupt\_flag\_get**

函数ctc\_interrupt\_flag\_get描述见下表:

**表 3-155. 函数 ctc\_interrupt\_flag\_get**

函数名称	ctc_interrupt_flag_get
函数原形	FlagStatus ctc_interrupt_flag_get(uint32_t int_flag);

功能描述	获取CTC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	CTC中断标志
CTC_INT_FLAG_C KOK	时钟校准完成中断标志位
CTC_INT_FLAG_C KWARN	时钟校准警告中断标志位
CTC_INT_FLAG_E RR	错误中断标志位
CTC_INT_FLAG_E REF	期望参考信号中断标志位
CTC_INT_FLAG_C KERR	时钟校准错误位
CTC_INT_FLAG_R EFMISS	参考同步脉冲信号丢失
CTC_INT_FLAG_T RIMERR	校准值错误位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

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

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

### 函数 ctc\_interrupt\_flag\_clear

函数ctc\_interrupt\_flag\_clear描述见下表：

表 3-156. 函数 ctc\_interrupt\_flag\_clear

函数名称	ctc_interrupt_flag_clear
函数原形	void ctc_interrupt_flag_clear(uint32_t int_flag);
功能描述	清除CTC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	CTC中断标志
CTC_INT_FLAG_C KOK	时钟校准完成中断标志位
CTC_INT_FLAG_C	时钟校准警告中断标志位

<i>KWARN</i>	
<i>CTC_INT_FLAG_ERR</i>	错误中断标志位
<i>CTC_INT_FLAG_REF</i>	期望参考信号中断标志位
<i>CTC_INT_FLAG_CKERR</i>	时钟校准错误位
<i>CTC_INT_FLAG_REFMIS</i>	参考同步脉冲信号丢失
<i>CTC_INT_FLAG_TRIMERR</i>	校准值错误位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
ctc_interrupt_flag_clear (CTC_INT_FLAG_CKOK);
```

## 3.7. CMP

CMP 通用比较器可独立工作，其输出端口可用于 I/O 口，也可和定时器结合使用。比较器在一定的条件下，可将模拟信号作为 TIMER 的触发源。章节 [3.7.1](#) 描述了 CMP 的寄存器列表，章节 [3.7.2](#) 对 CMP 库函数进行说明。

### 3.7.1. 外设寄存器说明

CMP寄存器列表如下表所示：

表 3-157. CMP 寄存器

寄存器名称	寄存器描述
CMP1_CS	CMP1控制状态寄存器
CMP3_CS	CMP3控制状态寄存器
CMP5_CS	CMP5控制状态寄存器

### 3.7.2. 外设库函数说明

CMP库函数列表如下表所示：

表 3-158. CMP 库函数

库函数名称	库函数描述
cmp_deinit	复位CMP

库函数名称	库函数描述
cmp_mode_init	CMP工作模式初始化
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_output_level_get	获取CMP输出状态

### 枚举类型 cmp\_enum

表 3-159. 枚举类型 cmp\_enum

成员名称	功能描述
CMP1	比较器1
CMP3	比较器3
CMP5	比较器5

### 函数 cmp\_deinit

函数cmp\_deinit描述见下表：

表 3-160. 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	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表3-159. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize CMP1 */
```

```
cmp_deinit(CMP1);
```

### 函数 cmp\_mode\_init

函数cmp\_mode\_init描述见下表：

表 3-161. 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 inverting_input);
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3-159. 枚举类型cmp_enum</a>
输入参数{in}	
inverting_input	反向输入源选择
CMP_INVERTING_IN PUT_1_4VREFINT	VREFINT *1/4作为输入源
CMP_INVERTING_IN PUT_1_2VREFINT	VREFINT *1/2作为输入源
CMP_INVERTING_IN PUT_3_4VREFINT	VREFINT *3/4作为输入源
CMP_INVERTING_IN PUT_VREFINT	VREFINT作为输入源
CMP_INVERTING_IN PUT_PA4	PA4作为输入源
CMP_INVERTING_IN PUT_PA5	PA5作为输入源
CMP_INVERTING_IN PUT_PA2_DAC0_OUT0	PA2作为CMP1输入源，DAC0_OUT0作为CMP3和CMP5输入源
CMP_INVERTING_IN PUT_DAC0_OUT0_PB2_PB15	DAC0_OUT0作为CMP1输入源，PB2作为CMP3输入源，PB15作为CMP5输入源
CMP_INVERTING_IN PUT_DAC1_OUT0	DAC1_OUT0作为输入源
CMP_INVERTING_IN PUT_DAC0_OUT1	DAC0_OUT1作为输入源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize CMP1 mode */
```

```
cmp_mode_init(CMP1, CMP_INVERTING_INPUT_1_4VREFINT);
```

### 函数 cmp\_output\_init

函数cmp\_output\_init描述见下表：

表 3-162. 函数 cmp\_output\_init

函数名称	cmp_output_init
函数原型	void cmp_output_init(cmp_enum cmp_periph, uint32_t output_selection, uint32_t output_polarity)
功能描述	CMP输出初始化
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表3-159. 枚举类型cmp_enum</a>
输入参数{in}	
output_selection	CMP输出选择
CMP_OUTPUT_NONE	无选择
CMP_OUTPUT_TIMER0_BKIN	CMP输出到TIMER0中止输入
CMP_OUTPUT_TIMER2_IC2	CMP输出到TIMER2_CH2输入捕获（仅对于CMP3）
CMP_OUTPUT_TIMER1_IC1	CMP输出到TIMER1_CH1输入捕获（仅对于CMP5）
CMP_OUTPUT_TIMER0_IC0	CMP输出到TIMER0_CH0输入捕获（仅对于CMP1）
CMP_OUTPUT_TIMER1_IC3	CMP输出到TIMER1_CH3输入捕获（仅对于CMP1）
CMP_OUTPUT_TIMER14_IC1	CMP输出到TIMER14_CH1输入捕获（仅对于CMP3）
CMP_OUTPUT_TIMER2_IC0	CMP输出到TIMER2_CH0输入捕获（仅对于CMP1）
CMP_OUTPUT_TIMER15_IC0	CMP输出到TIMER15_CH0输入捕获（仅对于CMP5）
输入参数{in}	
output_polarity	CMP输出极性
CMP_OUTPUT_POLARITY_INVERTED	输出反相
CMP_OUTPUT_POLARITY_NOINVERTED	输出正相
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize CMP1 output */
```

```
cmp_output_init(CMP1,CMP_OUTPUT_NONE,
```

CMP\_OUTPUT\_POLARITY\_NOINVERTED);

### 函数 cmp\_blanking\_init

函数cmp\_blanking\_init描述见下表:

表 3-163. 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3-159. 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_TIMER2_OC3	TIMER2_CH3输出比较信号作为消隐源（仅对于CMP3）
CMP_BLANKING_TIMER1_OC2	TIMER1_CH2输出比较信号作为消隐源（仅对于CMP1）
CMP_BLANKING_TIMER2_OC2	TIMER2_CH2输出比较信号作为消隐源（仅对于CMP1）
CMP_BLANKING_TIMER1_OC3	TIMER1_CH3输出比较信号作为消隐源（仅对于CMP5）
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize CMP1 blanking function */
```

```
cmp_blanking_init(CMP1, CMP_BLANKING_TIMER1_OC2);
```

### 函数 cmp\_enable

函数cmp\_enable描述见下表:

表 3-164. 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	CMP使能
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表3-159. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CMP1 */
cmp_enable(CMP1);
```

### 函数 cmp\_disable

函数cmp\_disable描述见下表：

**表 3-165. 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3-159. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CMP1 */
cmp_disable(CMP1);
```

### 函数 cmp\_lock\_enable

函数cmp\_lock\_enable描述见下表：

**表 3-166. 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3-159. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* lock CMP1 register */
cmp_lock_enable(CMP1);
```

### 函数 cmp\_output\_level\_get

函数cmp\_output\_level\_get描述见下表:

表 3-167. 函数 cmp\_output\_level\_get

函数名称	cmp_output_level_get
函数原型	uint32_t cmp_output_level_get(cmp_enum 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3-159. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
uint32_t	输出电平
CMP_OUTPUTLEV_EL_HIGH	比较器输出高电平
CMP_OUTPUTLEV_EL_LOW	比较器输出低电平

例如:

```
uint32_t level;
/* get CMP1 output level */
level = cmp_output_level_get(CMP1);
```

## 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 3-168. 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_OUT1_R12DH	DACx_OUT1 12位右对齐数据保持寄存器
DAC_OUT1_L12DH	DACx_OUT1 12位左对齐数据保持寄存器
DAC_OUT1_R8DH	DACx_OUT1 8位右对齐数据保持寄存器
DACC_R12DH	DACx并发模式12位右对齐数据保持寄存器
DACC_L12DH	DACx并发模式12位左对齐数据保持寄存器
DACC_R8DH	DACx并发模式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_OUT1_DO	DACx_OUT1数据输出寄存器
DAC_STAT0	DACx状态寄存器0
DAC_CTL1	DACx控制寄存器1
DAC_STAT1	DACx状态寄存器1

### 3.8.2. 外设库函数说明

DAC库函数列表如下表所示：

**表 3-169. 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_enable	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

库函数名称	库函数描述
<code>dac_wave_mode_config</code>	DAC噪声波模式配置
<code>dac_lfsr_noise_config</code>	DAC LFSR模式配置
<code>dac_triangle_noise_config</code>	DAC三角波模式配置
<code>dac_concurrent_enable</code>	并发DAC模式使能
<code>dac_concurrent_disable</code>	并发DAC模式禁能
<code>dac_concurrent_software_trigger_enable</code>	并发DAC模式软件触发使能
<code>dac_concurrent_output_buffer_enable</code>	并发DAC模式输出缓冲区使能
<code>dac_concurrent_output_buffer_disable</code>	并发DAC模式输出缓冲区禁能
<code>dac_concurrent_data_set</code>	并发DAC模式输出数据设置
<code>dac_output_fifo_enable</code>	DAC输出FIFO使能
<code>dac_output_fifo_disable</code>	DAC输出FIFO禁能
<code>dac_output_fifo_number_get</code>	DAC输出FIFO大小获取
<code>dac_flag_get</code>	DAC标志位获取
<code>dac_flag_clear</code>	DAC标志位清除
<code>dac_interrupt_enable</code>	DAC中断使能
<code>dac_interrupt_disable</code>	DAC中断禁能
<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
<code>dac_deinit</code>	DAC外设复位
<code>dac_enable</code>	DAC使能
<code>dac_disable</code>	DAC禁能

### 函数 `dac_deinit`

函数`dac_deinit`描述见下表：

**表 3-170. 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（ <code>x = 0,1</code> ）
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 3-171. 函数 **dac\_enable**

函数名称	dac_enable
函数原型	void dac_enable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC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 = 0,1）
输入参数{in}	
<b>dac_out</b>	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择（x = 0,1），DAC_OUT1仅可用于DAC0
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 3-172. 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 = 0,1）
输入参数{in}	
<b>dac_out</b>	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择（x = 0,1），DAC_OUT1仅可用于DAC0
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 3-173. 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 = 0,1)
输入参数{in}	
<b>dac_out</b>	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1), DAC_OUT1仅可用于DAC0
输出参数{out}	
-	-
返回值	
-	-

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 3-174. 函数 **dac\_dma\_disable**

函数名称	dac_dma_disable
函数原型	void dac_dma_disable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC的DMA功能禁能
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 = 0,1)
输入参数{in}	
<b>dac_out</b>	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1), DAC_OUT1仅可用于DAC0
输出参数{out}	
-	-
返回值	
-	-

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 3-175. 函数 dac\_gpio\_connect\_config

函数名称	dac_gpio_connect_config
函数原型	void dac_gpio_connect_config(uint32_t dac_periph, uint8_t dac_out, uint32_t gpio_connect);
功能描述	DAC的GPIO连接模式配置
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 = 0,1)
输入参数{in}	
<b>dac_out</b>	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1), DAC_OUT1仅可用于DAC0
输入参数{in}	
<b>gpio_connect</b>	DAC_OUTx连接GPIO模式
<i>PIN_PERIPHERAL</i>	DAC_OUTx输出连接外部管脚以及片上CMP
<i>PIN_PERIPHERAL_BUFFER</i>	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 3-176. 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 = 0,1)
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择 (x = 0,1), DAC_OUT1仅可用于DAC0
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 3-177. 函数 `dac_output_buffer_disable`

函数名称	<code>dac_output_buffer_disable</code>
函数原型	<code>void dac_output_buffer_disable(uint32_t dac_periph, uint8_t dac_out);</code>
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}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 (x = 0,1)
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择 (x = 0,1), DAC_OUT1仅可用于DAC0
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 3-178. 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}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0,1)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0,1), DAC_OUT1仅可用于DAC0
输出参数{out}	
-	-
返回值	
uint16_t	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 3-179. 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 = 0,1)
输入参数{in}	
<b>dac_out</b>	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1)，DAC_OUT1仅可用于DAC0
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 3-180. 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}	
<b>dac_periph</b>	DAC外设
<i>DACx</i>	DAC外设选择 (x = 0,1)
输入参数{in}	
<b>dac_out</b>	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1)，DAC_OUT1仅可用于DAC0
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 3-181. 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}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0,1)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0,1), DAC_OUT1仅可用于DAC0
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 3-182. 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}	
dac_periph	DAC外设

<i>DACx</i>	DAC外设选择 (x = 0,1)
<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 = 0,1), DAC_OUT1仅可用于DAC0
<b>输入参数{in}</b>	
<b>triggersource</b>	DAC触发源
<i>DAC_TRIGGER_T5_TRGO</i>	TIMER5 TRGO
<i>DAC_TRIGGER_T7_TRGO</i>	TIMER7 TRGO (仅用于GD32E51X_HD系列芯片)
<i>DAC_TRIGGER_T2_TRGO</i>	TIMER2 TRGO(仅用于GD32E51X_CL系列芯片)
<i>DAC_TRIGGER_T6_TRGO</i>	TIMER6 TRGO
<i>DAC_TRIGGER_T4_TRGO</i>	TIMER4 TRGO
<i>DAC_TRIGGER_T1_TRGO</i>	TIMER1 TRGO
<i>DAC_TRIGGER_T3_TRGO</i>	TIMER3 TRGO
<i>DAC_TRIGGER_EXTI_9</i>	EXTI线9中断
<i>DAC_TRIGGER_SOFTWARE</i>	软件触发
<i>DAC_TRIGGER_SHRTIMER_DACTRIG0</i>	SHRTIMER_DACTRIG0触发
<i>DAC_TRIGGER_SHRTIMER_DACTRIG1</i>	SHRTIMER_DACTRIG1触发
<i>DAC_TRIGGER_SHRTIMER_DACTRIG2</i>	SHRTIMER_DACTRIG2触发
<i>DAC_TRIGGER_T14_TRGO</i>	TIMER14 TRGO
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

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

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

函数 `dac_software_trigger_enable`

函数 `dac_software_trigger_enable` 描述见下表：

表 3-183. 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（ $x = 0,1$ ）
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择（ $x = 0,1$ ）， <code>DAC_OUT1</code> 仅可用于DAC0
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 3-184. 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（ $x = 0,1$ ）
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择（ $x = 0,1$ ）， <code>DAC_OUT1</code> 仅可用于DAC0
输入参数{in}	
<code>wave_mode</code>	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 3-185. 函数 **dac\_lfsr\_noise\_config**

函数名称	dac_lfsr_noise_config
函数原型	void dac_lfsr_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t unmask_bits);
功能描述	DAC LFSR模式配置
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 = 0,1)
输入参数{in}	
<b>dac_out</b>	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1), DAC_OUT1仅可用于DAC0
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 3-186. 函数 `dac_triangle_noise_config`

函数名称	<code>dac_triangle_noise_config</code>
函数原型	<code>void dac_triangle_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t amplitude);</code>
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}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择（ $x = 0, 1$ ）
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择（ $x = 0, 1$ ）， <code>DAC_OUT1</code> 仅可用于DAC0
输入参数{in}	
<code>amplitude</code>	三角波幅值
<code>DAC_TRIANGLE_AMPLITUDE_x</code>	$x = 2^n - 1$ （ $n = 1..12$ ）
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_concurrent_enable`

函数 `dac_concurrent_enable` 描述见下表：

表 3-187. 函数 `dac_concurrent_enable`

函数名称	<code>dac_concurrent_enable</code>
函数原型	<code>void dac_concurrent_enable(uint32_t dac_periph);</code>
功能描述	并发DAC模式使能
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（ $x = 0$ ）
输出参数{out}	

-	-
返回值	
-	-

例如:

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

```
dac_concurrent_enable(DAC0);
```

### 函数 **dac\_concurrent\_disable**

函数 **dac\_concurrent\_disable** 描述见下表:

表 3-188. 函数 **dac\_concurrent\_disable**

函数名称	dac_concurrent_disable
函数原型	void dac_concurrent_disable(uint32_t dac_periph);
功能描述	并发DAC模式禁能
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)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
dac_concurrent_disable(DAC0);
```

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

函数 **dac\_concurrent\_software\_trigger\_enable** 描述见下表:

表 3-189. 函数 **dac\_concurrent\_software\_trigger\_enable**

函数名称	dac_concurrent_software_trigger_enable
函数原型	void dac_concurrent_software_trigger_enable(uint32_t dac_periph);
功能描述	并发DAC模式软件触发使能
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)
输出参数{out}	

-	-
返回值	
-	-

例如:

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

```
dac_concurrent_software_trigger_enable(DAC0);
```

### 函数 `dac_concurrent_output_buffer_enable`

函数 `dac_concurrent_output_buffer_enable` 描述见下表:

表 3-190. 函数 `dac_concurrent_output_buffer_enable`

函数名称	<code>dac_concurrent_output_buffer_enable</code>
函数原型	<code>void dac_concurrent_output_buffer_enable(uint32_t dac_periph);</code>
功能描述	并发DAC模式输出缓冲区使能
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 ( $x = 0$ )
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC0 concurrent buffer function */
```

```
dac_concurrent_output_buffer_enable(DAC0);
```

### 函数 `dac_concurrent_output_buffer_disable`

函数 `dac_concurrent_output_buffer_disable` 描述见下表:

表 3-191. 函数 `dac_concurrent_output_buffer_disable`

函数名称	<code>dac_concurrent_output_buffer_disable</code>
函数原型	<code>void dac_concurrent_output_buffer_disable(uint32_t dac_periph);</code>
功能描述	并发DAC模式输出缓冲区禁能
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 ( $x = 0$ )
输出参数{out}	

-	-
返回值	
-	-

例如:

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

```
dac_concurrent_output_buffer_disable(DAC0);
```

### 函数 **dac\_concurrent\_data\_set**

函数 **dac\_concurrent\_data\_set** 描述见下表:

表 3-192. 函数 **dac\_concurrent\_data\_set**

函数名称	dac_concurrent_data_set
函数原型	void dac_concurrent_data_set(uint32_t dac_periph, uint32_t dac_align, uint16_t data0, uint16_t data1);
功能描述	并发DAC模式输出数据设置
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_align</b>	DAC对齐模式
<i>DAC_ALIGN_8B_R</i>	8位数据右对齐
<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
输入参数{in}	
<b>data0</b>	写入DACx_OUT0的数据 (0~4095)
输入参数{in}	
<b>data1</b>	写入DACx_OUT1的数据 (0~4095)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

### 函数 **dac\_output\_fifo\_enable**

函数 **dac\_output\_fifo\_enable** 描述见下表:

表 3-193. 函数 `dac_output_fifo_enable`

函数名称	<code>dac_output_fifo_enable</code>
函数原型	<code>void dac_output_fifo_enable(uint32_t dac_periph, uint8_t dac_out);</code>
功能描述	DAC输出FIFO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 = 0,1)
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择 (x = 0,1)，DAC_OUT1仅可用于DAC0
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC0_OUT0 output FIFO */
dac_output_fifo_enable(DAC0, DAC_OUT0);
```

### 函数 `dac_output_fifo_disable`

函数`dac_output_fifo_disable`描述见下表:

表 3-194. 函数 `dac_output_fifo_disable`

函数名称	<code>dac_output_fifo_disable</code>
函数原型	<code>void dac_output_fifo_disable(uint32_t dac_periph, uint8_t dac_out);</code>
功能描述	DAC输出FIFO禁能
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 = 0,1)
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择 (x = 0,1)，DAC_OUT1仅可用于DAC0
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DAC0_OUT0 output FIFO */
```

```
dac_output_fifo_disable(DAC0, DAC_OUT0);
```

### 函数 dac\_output\_fifo\_number\_get

函数dac\_output\_fifo\_number\_get描述见下表：

表 3-195. 函数 dac\_output\_fifo\_number\_get

函数名称	dac_output_fifo_number_get
函数原型	uint16_t dac_output_fifo_number_get(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC输出FIFO大小获取
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 = 0,1）
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择（x = 0,1），DAC_OUT1仅可用于DAC0
输出参数{out}	
-	-
返回值	
uint16_t	DAC输出FIFO大小（0~4）

例如：

```
/* get DAC0_OUT0 output FIFO number */
```

```
uint16_t number =0;
```

```
number = dac_output_fifo_number_get (DAC0, DAC_OUT0);
```

### 函数 dac\_flag\_get

函数dac\_flag\_get描述见下表：

表 3-196. 函数 dac\_flag\_get

函数名称	dac_flag_get
函数原型	FlagStatus dac_flag_get(uint32_t dac_periph, uint32_t flag);
功能描述	DAC标志位获取
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 = 0,1）
输入参数{in}	

flag	DAC状态标志位
<i>DAC_FLAG_DDUDR0</i>	DACx_OUT0 DMA欠载标志位
<i>DAC_FLAG_FIFOF0</i>	DACx_OUT0 FIFO满载标志位
<i>DAC_FLAG_FIFOE0</i>	DACx_OUT0 FIFO空载标志位
<i>DAC_FLAG_FIFO0VR0</i>	DACx_OUT0 FIFO过载标志位
<i>DAC_FLAG_FIFOU0DR0</i>	DACx_OUT0 FIFO欠载标志位
<i>DAC_FLAG_DDUDR1</i>	DACx_OUT1 DMA欠载标志位
<i>DAC_FLAG_FIFOF1</i>	DACx_OUT1 FIFO满载标志位
<i>DAC_FLAG_FIFOE1</i>	DACx_OUT1 FIFO空载标志位
<i>DAC_FLAG_FIFO1VR1</i>	DACx_OUT1 FIFO过载标志位
<i>DAC_FLAG_FIFOU1DR1</i>	DACx_OUT1 FIFO欠载标志位
输出参数{out}	
-	-
返回值	
FlagStatus	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 3-197. 函数 **dac\_flag\_clear**

函数名称	dac_flag_clear
函数原型	void dac_flag_clear(uint32_t dac_periph, uint32_t flag);
功能描述	DAC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 = 0,1)
输入参数{in}	
flag	DAC状态标志位
<i>DAC_FLAG_DDUDR0</i>	DACx_OUT0 DMA欠载标志位
<i>DAC_FLAG_FIFO0VR0</i>	DACx_OUT0 FIFO过载标志位
<i>DAC_FLAG_FIFOU0DR0</i>	DACx_OUT0 FIFO欠载标志位
<i>DAC_FLAG_DDUDR1</i>	DACx_OUT1 DMA欠载标志位
<i>DAC_FLAG_FIFO1VR1</i>	DACx_OUT1 FIFO过载标志位
<i>DAC_FLAG_FIFOU1DR1</i>	DACx_OUT1 FIFO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 3-198. 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}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0,1)
输入参数{in}	
interrupt	DAC中断
DAC_INT_DDUDR0	DACx_OUT0 DMA欠载中断
DAC_INT_FIFOOVR0	DACx_OUT0 FIFO过载中断
DAC_INT_FIFODR0	DACx_OUT0 FIFO欠载中断
DAC_INT_DDUDR1	DACx_OUT1 DMA欠载中断
DAC_INT_FIFOOVR1	DACx_OUT1 FIFO过载中断
DAC_INT_FIFODR1	DACx_OUT1 FIFO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 3-199. 函数 `dac_interrupt_disable`

函数名称	<code>dac_interrupt_disable</code>
函数原型	<code>void dac_interrupt_disable(uint32_t dac_periph, uint32_t interrupt);</code>
功能描述	DAC中断禁能
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 = 0,1)
输入参数{in}	
<code>interrupt</code>	DAC中断
<code>DAC_INT_DDUDR0</code>	DACx_OUT0 DMA欠载中断
<code>DAC_INT_FIFOOVR0</code>	DACx_OUT0 FIFO过载中断
<code>DAC_INT_FIFOU DR0</code>	DACx_OUT0 FIFO欠载中断
<code>DAC_INT_DDUDR1</code>	DACx_OUT1 DMA欠载中断
<code>DAC_INT_FIFOOVR1</code>	DACx_OUT1 FIFO过载中断
<code>DAC_INT_FIFOU DR1</code>	DACx_OUT1 FIFO欠载中断
输出参数{out}	
-	-
返回值	
-	-

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 3-200. 函数 `dac_interrupt_flag_get`

函数名称	<code>dac_interrupt_flag_get</code>
函数原型	<code>FlagStatus dac_interrupt_flag_get(uint32_t dac_periph, uint32_t int_flag);</code>
功能描述	DAC中断标志位获取
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 = 0,1)
输入参数{in}	
<code>int_flag</code>	DAC中断标志位
<code>DAC_INT_FLAG_DDU DR0</code>	DACx_OUT0 DMA欠载中断标志位

<i>DAC_INT_FLAG_FIFO</i> <i>OVR0</i>	DACx_OUT0 FIFO过载中断标志位
<i>DAC_INT_FLAG_FIFO</i> <i>UDR0</i>	DACx_OUT0 FIFO欠载中断标志位
<i>DAC_INT_FLAG_DDU</i> <i>DR1</i>	DACx_OUT1 DMA欠载中断标志位
<i>DAC_INT_FLAG_FIFO</i> <i>OVR1</i>	DACx_OUT1 FIFO过载中断标志位
<i>DAC_INT_FLAG_FIFO</i> <i>UDR1</i>	DACx_OUT1 FIFO欠载中断标志位
输出参数{out}	
-	-
返回值	
<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 3-201. 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}	
<b>dac_periph</b>	DAC外设
<i>DACx</i>	DAC外设选择（x = 0,1）
输入参数{in}	
<b>int_flag</b>	DAC中断标志位
<i>DAC_INT_FLAG_DDU</i> <i>DR0</i>	DACx_OUT0 DMA欠载中断标志位
<i>DAC_INT_FLAG_FIFO</i> <i>OVR0</i>	DACx_OUT0 FIFO过载中断标志位
<i>DAC_INT_FLAG_FIFO</i> <i>UDR0</i>	DACx_OUT0 FIFO欠载中断标志位
<i>DAC_INT_FLAG_DDU</i> <i>DR1</i>	DACx_OUT1 DMA欠载中断标志位

DAC_INT_FLAG_FIFO OVR1	DACx_OUT1 FIFO过载中断标志位
DAC_INT_FLAG_FIFO UDR1	DACx_OUT1 FIFO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. 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[3.9.1](#)描述了DBG的寄存器列表，章节[3.9.2](#)对DBG库函数进行说明。

### 3.9.1. 外设寄存器说明

DBG寄存器列表如下表所示:

表 3-202. 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

### 3.9.2. 外设库函数说明

DBG库函数列表如下表所示:

表 3-203. DBG 库函数

库函数名称	库函数描述
dbg_deinit	复位DEBUG模块
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
dbg_trace_pin_enable	使能跟踪引脚分配
dbg_trace_pin_disable	禁能跟踪引脚分配

## 枚举类型 dbg\_periph\_enum

表 3-204. 枚举类型 dbg\_periph\_enum

成员名称	功能描述
DBG_CAN2_HOLD	当内核停止时，CAN2接收寄存器停止接收数据
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
DBG_TIMER2_HOLD	当内核停止时，保持TIMER2计数器计数值不变
DBG_TIMER3_HOLD	当内核停止时，保持TIMER3计数器计数值不变
DBG_CAN0_HOLD	当内核停止时，CAN0接收寄存器停止接收数据
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_TIMER4_HOLD	当内核停止时，保持TIMER4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_CAN1_HOLD	当内核停止时，CAN1接收寄存器停止接收数据
DBG_I2C2_HOLD	当内核停止时，保持I2C2的SMBUS状态不变，用于调试
DBG_TIMER11_HOLD	当内核停止时，保持TIMER11计数器计数值不变
DBG_TIMER12_HOLD	当内核停止时，保持TIMER12计数器计数值不变
DBG_TIMER13_HOLD	当内核停止时，保持TIMER13计数器计数值不变
DBG_TIMER8_HOLD	当内核停止时，保持TIMER8计数器计数值不变
DBG_TIMER9_HOLD	当内核停止时，保持TIMER9计数器计数值不变
DBG_TIMER10_HOLD	当内核停止时，保持TIMER10计数器计数值不变
DBG_SHRTIMER_HOLD	当内核停止时，保持SHRTIMER计数器计数值不变，除GD32EPRT系列外

## 函数 dbg\_deinit

函数dbg\_deinit描述见下表：

表 3-205. 函数 dbg\_deinit

函数名称	dbg_deinit
函数原形	void dbg_deinit(void);
功能描述	复位DEBUG模块
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 DBG */
```

```
dbg_deinit();
```

### 函数 **dbg\_id\_get**

函数dbg\_id\_get描述见下表:

**表 3-206. 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 3-207. 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	在深度睡眠模式下, 保持调试器连接, 可进行调试

<i>R_DEEPSLEEP</i>	
<i>DBG_LOW_POWER</i> <i>R_STANDBY</i>	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：

```
/* keep debugger connection during sleep 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 3-208. 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	-
输入参数{in}	
<b>dbg_low_power</b>	低功耗模式调试保持
<i>DBG_LOW_POWER</i> <i>R_SLEEP</i>	在睡眠模式下，保持调试器连接，可进行调试
<i>DBG_LOW_POWER</i> <i>R_DEEPSLEEP</i>	在深度睡眠模式下，保持调试器连接，可进行调试
<i>DBG_LOW_POWER</i> <i>R_STANDBY</i>	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：

```
/* do not keep debugger connection during sleep 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 3-209. 函数 `dbg_periph_enable`

函数名称	<code>dbg_periph_enable</code>
函数原形	<code>void dbg_periph_enable(dbg_periph_enum dbg_periph);</code>
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	-
输入参数{in}	
<code>dbg_periph</code>	参考 <a href="#">表3-204. 枚举类型dbg_periph_enum</a>
<code>DBG_FWDGT_HOLD</code>	当内核停止时，保持FWDGT计数器时钟
<code>DBG_WWDGT_HOLD</code>	当内核停止时，保持WWDGT计数器时钟
<code>DBG_CANx_HOLD</code>	当内核停止时，CANx (x=0,1,2) 接收寄存器停止接收数据
<code>DBG_I2Cx_HOLD</code>	当内核停止时，保持I2Cx (x=0,1,2) 的SMBUS状态不变，用于调试
<code>DBG_TIMERx_HOLD</code>	当内核停止时，保持TIMERx计数器计数值不变 (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)，TIMER8..13 GD32EPRT系列不支持。
<code>DBG_SHRTIMER_HOLD</code>	当内核停止时，保持SHRTIMER计数器计数值不变，除GD32EPRT系列外
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* hold TIMER0 counter when core is halted */
```

```
dbg_periph_enable(DBG_TIMER0_HOLD);
```

## 函数 `dbg_periph_disable`

函数`dbg_periph_disable`描述见下表：

表 3-210. 函数 `dbg_periph_disable`

函数名称	<code>dbg_periph_disable</code>
函数原形	<code>void dbg_periph_disable(dbg_periph_enum dbg_periph);</code>
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}	
<code>dbg_periph</code>	参考 <a href="#">表3-204. 枚举类型dbg_periph_enum</a>
<code>DBG_FWDGT_HOLD</code>	当内核停止时，保持FWDGT计数器时钟
<code>DBG_WWDGT_HOLD</code>	当内核停止时，保持WWDGT计数器时钟

<b>DBG_CANx_HOLD</b>	当内核停止时，CANx (x=0,1,2) 接收寄存器停止接收数据
<b>DBG_I2Cx_HOLD</b>	当内核停止时，保持I2Cx (x=0,1,2) 的SMBUS状态不变，用于调试
<b>DBG_TIMERx_HOLD</b>	当内核停止时，保持TIMERx计数器计数值不变 (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)，TIMER8..13 GD32EPRT系列不支持。
<b>DBG_SHRTIMER_HOLD</b>	当内核停止时，保持SHRTIMER计数器计数值不变，除GD32EPRT系列外
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* do not hold TIMER0 counter when core is halted */
```

```
dbg_periph_disable(DBG_TIMER0_HOLD);
```

### 函数 dbg\_trace\_pin\_enable

函数dbg\_trace\_pin\_enable描述见下表：

表 3-211. 函数 dbg\_trace\_pin\_enable

<b>函数名称</b>	dbg_trace_pin_enable
<b>函数原形</b>	void dbg_trace_pin_enable(void);
<b>功能描述</b>	使能跟踪引脚分配
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable trace pin assignment */
```

```
dbg_trace_pin_enable();
```

### 函数 dbg\_trace\_pin\_disable

函数dbg\_trace\_pin\_disable描述见下表：

表 3-212. 函数 dbg\_trace\_pin\_disable

<b>函数名称</b>	dbg_trace_pin_disable
<b>函数原形</b>	void dbg_trace_pin_disable(void);
<b>功能描述</b>	禁能跟踪引脚分配

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 trace pin assignment */
```

```
dbg_trace_pin_disable();
```

## 3.10. DMA

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[3.10.1](#)描述了DMA的寄存器列表，章节[3.10.2](#)对DMA库函数进行说明。

### 3.10.1. 外设寄存器说明

DMA寄存器列表如下表所示：

表 3-213. 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	中断标志位清除器
DMA_CHxCTL (x=0..6)	通道 x 控制寄存器
DMA_CHxCNT (x=0..6)	通道 x 计数寄存器
DMA_CHxPADDR (x=0..6)	通道 x 外设基地址寄存器
DMA_CHxMADDR (x=0..6)	通道 x 存储器基地址寄存器

### 3.10.2. 外设库函数说明

DMA库函数列表如下表所示：

表 3-214. DMA 库函数

库函数名称	库函数描述
dma_deinit	复位外设 DMAx 的通道 y 的所有寄存器

库函数名称	库函数描述
<code>dma_struct_para_init</code>	将 DMA 结构体中所有参数初始化为默认值
<code>dma_init</code>	初始化外设 DMAx 的通道 y
<code>dma_circulation_enable</code>	DMA 循环模式使能
<code>dma_circulation_disable</code>	DMA 循环模式禁能
<code>dma_memory_to_memory_enable</code>	存储器到存储器 DMA 传输使能
<code>dma_memory_to_memory_disable</code>	存储器到存储器 DMA 传输禁能
<code>dma_channel_enable</code>	外设 DMAx 的通道 y 传输使能
<code>dma_channel_disable</code>	外设 DMAx 的通道 y 传输禁能
<code>dma_periph_address_config</code>	DMAx 通道 y 传输的外设基地址配置
<code>dma_memory_address_config</code>	DMAx 通道 y 传输的存储器基地址配置
<code>dma_transfer_number_config</code>	配置 DMAx 通道 y 还有多少数据要传输
<code>dma_transfer_number_get</code>	获取 DMAx 通道 y 还有多少数据要传输
<code>dma_priority_config</code>	DMAx 通道 y 的传输软件优先级配置
<code>dma_memory_width_config</code>	DMAx 通道 y 传输的存储器数据宽度配置
<code>dma_periph_width_config</code>	DMAx 通道 y 传输的外设数据宽度配置
<code>dma_memory_increase_enable</code>	DMAx 通道 y 传输的存储器地址生成算法增量模式使能
<code>dma_memory_increase_disable</code>	DMAx 通道 y 传输的存储器地址生成算法增量模式禁能
<code>dma_periph_increase_enable</code>	DMAx 通道 y 传输的外设地址生成算法增量模式使能
<code>dma_periph_increase_disable</code>	DMAx 通道 y 传输的外设地址生成算法增量模式禁能
<code>dma_transfer_direction_config</code>	DMAx 通道 y 的传输方向配置
<code>dma_flag_get</code>	获取 DMAx 通道 y 标志位状态
<code>dma_flag_clear</code>	清除 DMAx 通道 y 标志位状态
<code>dma_interrupt_enable</code>	DMAx 通道 y 中断使能
<code>dma_interrupt_disable</code>	DMAx 通道 y 中断禁能
<code>dma_interrupt_flag_get</code>	获取 DMAx 通道 y 中断标志位状态
<code>dma_interrupt_flag_clear</code>	清除 DMAx 通道 y 中断标志位状态

## 枚举类型 `dma_channel_enum`

表 3-215. 枚举类型 `dma_channel_enum`

成员名称	功能描述
<code>DMA_CH0</code>	DMA通道0
<code>DMA_CH1</code>	DMA通道1
<code>DMA_CH2</code>	DMA通道2
<code>DMA_CH3</code>	DMA通道3
<code>DMA_CH4</code>	DMA通道4
<code>DMA_CH5</code>	DMA通道5
<code>DMA_CH6</code>	DMA通道6

## 结构体 dma\_parameter\_struct

表 3-216. 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

## 函数 dma\_deinit

函数dma\_deinit描述见下表:

表 3-217. 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	复位外设 DMAx 的通道 y 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
DMAx(x=0,1)	DMA 外设选择
输入参数{in}	
<b>channelx</b>	DMA 通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA 通道选择, 参考 <a href="#">表 3-215. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

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

## 函数 dma\_struct\_para\_init

函数dma\_struct\_para\_init描述见下表:

表 3-218. 函数 dma\_struct\_para\_init

函数名称	dma_struct_para_init
函数原型	void dma_struct_para_init(dma_parameter_struct* init_struct);
功能描述	将 DMA 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}	
*init_struct	一个已经定义的 dma_parameter_struct 结构体变量地址，参考 <a href="#">表 3-216. 结构体 dma_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

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 3-219. 函数 dma\_init

函数名称	dma_init
函数原型	void dma_init(uint32_t dma_periph, dma_channel_enum channelx, dma_parameter_struct* init_struct);
功能描述	初始化外设 DMAx 的通道 y
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(DMA0:x=0..6; DMA1:x=0..4)	DMA 通道选择，参考 <a href="#">表 3-215. 枚举类型 dma_channel_enum</a>
输入参数{in}	
init_struct	初始化结构体，结构体成员参考 <a href="#">表 3-216. 结构体 dma_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* DMA0 channel0 initialize */
dma_parameter_struct dma_init_struct;
dma_deinit(DMA0, DMA_CH0);
dma_struct_para_init(&dma_init_struct);
dma_init_struct.direction = DMA_PERIPHERAL_TO_MEMORY;
dma_init_struct.memory_addr = (uint32_t)g_destbuf;
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_struct.memory_width = DMA_MEMORY_WIDTH_32BIT;
dma_init_struct.number = TRANSFER_NUM/4;
dma_init_struct.periph_addr = (uint32_t) FLASH_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_32BIT;
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 3-220. 函数 dma\_circulation\_enable

函数名称	dma_circulation_enable
函数原型	void dma_circulation_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMA 循环模式使能
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(DMA0:x=0..6; DMA1:x=0..4)	DMA 通道选择，参考 <a href="#">表 3-215. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

## 函数 dma\_circulation\_disable

函数dma\_circulation\_disable描述见下表：

表 3-221. 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	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(DMA0:x=0..6; DMA1:x=0..4)	DMA 通道选择，参考 <a href="#">表 3-215. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA0 channel0 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 3-222. 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, dma_channel_enum channelx);
功能描述	存储器到存储器 DMA 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

<i>DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)</i>	DMA 通道选择, 参考 <a href="#">表 3-215. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DMA0 channel0 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 3-223. 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, dma_channel_enum channelx);
功能描述	存储器到存储器 DMA 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(DMA0:x =0..6; DMA1: x=0..4)</i>	DMA 通道选择, 参考 <a href="#">表 3-215. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DMA0 channel0 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 3-224. 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	外设 DMAx 的通道 y 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(DMA0:x=0..6; DMA1:x=0..4)	DMA 通道选择, 参考 <a href="#">表 3-215. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

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

### 函数 dma\_channel\_disable

函数dma\_channel\_disable描述见下表:

表 3-225. 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	外设 DMAx 的通道 y 传输禁能
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(DMA0:x=0..6; DMA1:x=0..4)	DMA 通道选择, 参考 <a href="#">表 3-215. 枚举类型 dma_channel_enum</a>
输出参数{out}	

-	-
返回值	
-	-

例如：

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

### 函数 dma\_periph\_address\_config

函数dma\_periph\_address\_config描述见下表：

表 3-226. 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	DMAx 通道 y 传输的外设基地址配置
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(DMA0:x=0..6; DMA1:x=0..4)	DMA 通道选择，参考 <a href="#">表 3-215. 枚举类型 dma_channel_enum</a>
输入参数{in}	
address	外设基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

### 函数 dma\_memory\_address\_config

函数dma\_memory\_address\_config描述见下表：

表 3-227. 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	DMAx 通道 y 传输的存储器基地址配置
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(DMA0:x=0..6; DMA1:x=0..4)	DMA 通道选择, 参考 <a href="#">表 3-215. 枚举类型 dma_channel_enum</a>
输入参数{in}	
address	存储器基地址
输出参数{out}	
-	-
返回值	
-	-

例如:

```
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 3-228. 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 DMAx 通道 y 还有多少数据要传输
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(DMA0:x=0..6; DMA1:x=0..4)	DMA 通道选择, 参考 <a href="#">表 3-215. 枚举类型 dma_channel_enum</a>

输入参数{in}	
number	数据传输数量（0x00000000 – 0x0000FFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
#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 3-229. 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 DMAx 通道 y 还有多少数据要传输
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(DMA0:x=0..6; DMA1:x=0..4)	DMA 通道选择，参考 <a href="#">表 3-215. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
uint32_t	DMA 数据传输剩余数量（0x00000000 – 0x0000FFFF）

例如：

```
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 3-230. 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	DMAx 通道 y 的传输软件优先级配置
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(DMA0:x=0..6; DMA1:x=0..4)	DMA 通道选择, 参考 <a href="#">表 3-215. 枚举类型 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:

```
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 3-231. 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	DMAx 通道 y 传输的存储器数据宽度配置
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(DMA0:x=0..6; DMA1:x=0..4)</i>	DMA 通道选择, 参考 <a href="#">表 3-215. 枚举类型 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:

```
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 3-232. 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	DMAx 通道 y 传输的外设数据宽度配置
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(DMA0:x=0..6; DMA1:x=0..4)</i>	DMA 通道选择, 参考 <a href="#">表 3-215. 枚举类型 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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
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 3-233. 函数 `dma_memory_increase_enable`**

<b>函数名称</b>	<code>dma_memory_increase_enable</code>
<b>函数原型</b>	<code>void dma_memory_increase_enable(uint32_t dma_periph, dma_channel_enum channelx);</code>
<b>功能描述</b>	DMAx 通道 y 传输的存储器地址生成算法增量模式使能
<b>先决条件</b>	无
<b>被调用函数</b>	无
<b>输入参数{in}</b>	
<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(DMA0:x=0..6; DMA1:x=0..4)</i>	DMA 通道选择，参考 <a href="#">表 3-215. 枚举类型 <code>dma_channel_enum</code></a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
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 3-234. 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	DMAx 通道 y 传输的存储器地址生成算法增量模式禁能
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(DMA0:x=0..6; DMA1:x=0..4)	DMA 通道选择，参考 <a href="#">表 3-215. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
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 3-235. 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	DMAx 通道 y 传输的外设地址生成算法增量模式使能
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(DMA0:x	DMA 通道选择，参考 <a href="#">表 3-215. 枚举类型 dma_channel_enum</a>

=0..6; DMA1: x=0..4)	
输出参数{out}	
-	-
返回值	
-	-

例如:

```
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 3-236. 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	DMAx 通道 y 传输的外设地址生成算法增量模式禁能
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(DMA0:x =0..6; DMA1: x=0..4)	DMA 通道选择, 参考 <a href="#">表 3-215. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
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 3-237. 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	DMAx 通道 y 的传输方向配置
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(DMA0:x=0..6; DMA1:x=0..4)</i>	DMA 通道选择, 参考 <a href="#">表 3-215. 枚举类型 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:

```
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 3-238. 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 DMAx 通道 y 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(DMA0:x=0..6; DMA1:x=0..4)</i>	DMA 通道选择, 参考 <a href="#">表 3-215. 枚举类型 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	
<b>FlagStatus</b>	SET 或 RESET

例如:

```
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 3-239. 函数 dma\_flag\_clear

<b>函数名称</b>	dma_flag_clear
<b>函数原型</b>	void dma_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
<b>功能描述</b>	清除 DMAx 通道 y 标志位状态
<b>先决条件</b>	无
<b>被调用函数</b>	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(DMA0:x=0..6; DMA1:x=0..4)</i>	DMA 通道选择, 参考 <a href="#">表 3-215. 枚举类型 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	

-	-
---	---

例如：

```
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 3-240. 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	DMAx 通道 y 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(DMA0:x=0..6; DMA1:x=0..4)	DMA 通道选择，参考 <a href="#">表 3-215. 枚举类型 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：

```
/* DMA0 channel0 interrupt configuration */
```

```
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 3-241. 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	DMAx 通道 y 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
DMAx(x=0,1)	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA 通道选择, 参考 <a href="#">表 3-215. 枚举类型 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:

```
/* DMA0 channel0 interrupt configuration */
```

```
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 3-242. 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 DMAx 通道 y 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

<i>DMA_CHx</i> (DMA0: <i>x</i> =0..6; DMA1: <i>x</i> =0..4)	DMA 通道选择, 参考 <a href="#">表 3-215. 枚举类型 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	
<b>FlagStatus</b>	SET 或 RESET

例如:

```
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 3-243. 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 DMAx 通道 y 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</i> ( <i>x</i> =0,1)	DMA 外设选择
输入参数{in}	
<b>channelx</b>	DMA 通道
<i>DMA_CHx</i> (DMA0: <i>x</i> =0..6; DMA1: <i>x</i> =0..4)	DMA 通道选择, 参考 <a href="#">表 3-215. 枚举类型 dma_channel_enum</a>
输入参数{in}	
<b>flag</b>	DMA 标志
<i>DMA_INT_FLAG_G</i>	DMA 通道全局中断标志
<i>DMA_INT_FLAG_F</i>	DMA 通道传输完成中断标志

<i>TF</i>	
<i>DMA_INT_FLAG_HTF</i>	DMA 通道半传输完成中断标志
<i>DMA_INT_FLAG_ERRR</i>	DMA 通道错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

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

## 3.11. ENET

以太网模块包含10/100Mbps以太网MAC（媒体访问控制器），采用DMA优化数据帧的发送与接收性能，支持MII（媒体独立接口）与RMII（简化的媒体独立接口）两种与物理层(PHY)通讯的标准接口，实现以太网数据帧的发送与接收。章节[3.11.1](#)描述了ENET的寄存器列表，章节[3.11.2](#)对ENET库函数进行说明。

### 3.11.1. 外设寄存器描述

ENET寄存器列表如下表所示：

表 3-244. ENET 寄存器

寄存器名称	寄存器描述
ENET_MAC_CFG	MAC配置寄存器
ENET_MAC_FRMF	MAC帧过滤器寄存器
ENET_MAC_HLH	MAC hash列表高寄存器
ENET_MAC_HLL	MAC hash列表低寄存器
ENET_MAC_PHY_CTL	MAC PHY控制寄存器
ENET_MAC_PHY_DATA	MAC PHY数据寄存器
ENET_MAC_FCTL	MAC流控寄存器
ENET_MAC_VLT	MAC VLAN标签寄存器
ENET_MAC_RWFF	MAC远程唤醒帧过滤器寄存器
ENET_MAC_WUM	MAC唤醒管理寄存器
ENET_MAC_DBG	MAC调试寄存器
ENET_MAC_INTF	MAC中断状态寄存器
ENET_MAC_INTMS	MAC中断屏蔽寄存器

寄存器名称	寄存器描述
K	
ENET_MAC_ADDR 0H	MAC地址0高寄存器
ENET_MAC_ADDR 0L	MAC地址0低寄存器
ENET_MAC_ADDR 1H	MAC地址1高寄存器
ENET_MAC_ADDR 1L	MAC地址1低寄存器
ENET_MAC_ADDT 2H	MAC地址2高寄存器
ENET_MAC_ADDR 2L	MAC地址2低寄存器
ENET_MAC_ADDR 3H	MAC地址3高寄存器
ENET_MAC_ADDR 3L	MAC地址3低寄存器
ENET_MAC_FCTH	MAC流控阈值寄存器
ENET_MSC_CTL	MSC控制寄存器
ENET_MSC_RINTF	MSC接收中断状态寄存器
ENET_MSC_TINTF	MSC发送中断状态寄存器
ENET_MSC_RINT MSK	MSC接收中断屏蔽寄存器
ENET_MSC_TINTM SK	MSC发送中断屏蔽寄存器
ENET_MSC_SCCN T	MSC 1次冲突后发送"好"帧的计数器寄存器
ENET_MSC_MSCC NT	MSC 1次以上冲突后发送"好"帧的计数器寄存器
ENET_MSC_TGFC NT	MSC发送"好"帧计数器寄存器
ENET_MSC_RFCE CNT	MSC CRC错误接收帧计数器寄存器
ENET_MSC_RFAE CNT	MSC对齐错误接收帧计数器寄存器
ENET_MSC_RGUF CNT	MSC"好"单播帧接收帧计数器寄存器
ENET_PTP_TSCTL	PTP时间戳控制寄存器
ENET_PTP_SSINC	PTP亚秒递增寄存器
ENET_PTP_TSH	PTP时间戳高寄存器
ENET_PTP_TSL	PTP时间戳低寄存器
ENET_PTP_TSUH	PTP时间戳高更新寄存器

寄存器名称	寄存器描述
ENET_PTP_TSUL	PTP时间戳低更新寄存器
ENET_PTP_TSADD END	PTP时间戳加数寄存器
ENET_PTP_ETH	PTP期望时间高寄存器
ENET_PTP_ETL	PTP期望时间低寄存器
ENET_PTP_TSF	PTP时间戳标志寄存器
ENET_PTP_PPSCTL	PTP PPS控制寄存器
ENET_DMA_BCTL	DMA总线控制寄存器
ENET_DMA_TPEN	DMA发送查询使能寄存器
ENET_DMA_RPEN	DMA接收查询使能寄存器
ENET_DMA_RDТА DDR	DMA接收描述符列表地址寄存器
ENET_DMA_TDТА DDR	DMA发送描述符列表地址寄存器
ENET_DMA_STAT	DMA状态寄存器
ENET_DMA_CTL	DMA控制寄存器
ENET_DMA_INTEN	DMA中断使能寄存器
ENET_DMA_MFBO CNT	DMA丢失帧和缓存溢出计数器寄存器
ENET_DMA_RSWD C	DMA接收接收状态看门狗计数器寄存器
ENET_DMA_CTDA DDR	DMA当前发送描述符地址寄存器
ENET_DMA_CRDA DDR	DMA当前接收描述符地址寄存器
ENET_DMA_CTBA DDR	DMA当前发送缓存地址寄存器
ENET_DMA_CRBA DDR	DMA当前接收缓存地址寄存器

### 3.11.2. 外设库函数说明

ENET库函数列表如下表所示：

**表 3-245. ENET 库函数**

库函数名称	库函数描述
常用函数	
enet_deinit	复位ENET模块及相关软件初始化所需结构体
enet_initpara_config	配置ENET模块的各类不常用功能。当enet_init()函数无法满足所需实现功能时调用，必须在enet_init()函数之前调用
enet_init	ENET模块初始化，配置用户最关心的功能

库函数名称	库函数描述
enet_software_reset	复位ENET寄存器，并检测CLK_TX/CLK_RX信号
enet_rxframe_size_get	检测接收帧是否有错误，正确时返回帧长度
enet_descriptors_chain_init	初始化DMA接收/发送描述符为链模式
enet_descriptors_ring_init	初始化DMA接收/发送描述符为环模式
enet_frame_receive	处理当前接收到的帧，并将当前描述符中存储的接收帧数据拷贝到指定区域
enet_frame_transmit	将指定区域内的数据拷贝到当前发送描述符中，并发送
enet_transmit_checksum_config	配置发送帧校验和模式
enet_enable	ENET Tx/Rx功能使能（包括ENET外设内的MAC和DMA模块）
enet_disable	ENET Tx/Rx功能禁能（包括ENET外设内的MAC和DMA模块）
enet_mac_address_set	配置MAC地址
enet_mac_address_get	获取MAC地址
enet_flag_get	获取ENET模块MAC/MSC/PTP/DMA状态标志位
enet_flag_clear	清除ENET状态标志位
enet_interrupt_enable	使能ENET模块MAC/MSC/DMA中断
enet_interrupt_disable	禁能ENET模块MAC/MSC/DMA中断
enet_interrupt_flag_get	获取ENET模块MAC/MSC/DMA中断标志位
enet_interrupt_flag_clear	禁能ENET模块DMA中断标志位
MAC功能函数	
enet_tx_enable	ENET发送功能使能（包括ENET外设内的MAC和DMA模块）
enet_tx_disable	ENET发送功能禁能（包括ENET外设内的MAC和DMA模块）
enet_rx_enable	ENET接收功能使能（包括ENET外设内的MAC和DMA模块）
enet_rx_disable	ENET发送功能禁能（包括ENET外设内的MAC和DMA模块）
enet_registers_get	获取指定范围ENET寄存器值
enet_debug_status_get	获取ENET调试状态信息
enet_address_filter_enable	MAC地址过滤器使能
enet_address_filter_disable	MAC地址过滤器禁能
enet_address_filter_config	配置MAC地址过滤器模式
enet_phy_config	PHY接口配置（配置SMI时钟并复位PHY芯片）
enet_phy_write_read	写/读PHY寄存器
enet_phyloopback_enable	使能PHY芯片回环模式
enet_phyloopback_disable	禁能PHY芯片回环模式
enet_forward_feature_enable	使能ENET帧通过相关功能
enet_forward_feature_disable	禁能ENET帧通过相关功能
enet_fliter_feature_enable	使能ENET帧过滤器相关功能
enet_fliter_feature_disable	禁能ENET帧过滤器相关功能
流控功能函数	
enet_pauseframe_generate	生成暂停帧，使能发送流控功能后ENET模块将发送暂停帧
enet_pauseframe_detect_config	配置暂停帧检测类型

库函数名称	库函数描述
enet_pauseframe_config	配置暂停帧参数
enet_flowcontrol_threshold_config	配置流控阈值
enet_flowcontrol_feature_enable	使能ENET流控相关功能
enet_flowcontrol_feature_disable	禁能ENET流控相关功能
DMA功能函数	
enet_dmaprocess_state_get	获取DMA发送/接收流程状态
enet_dmaprocess_resume	DMA发送/接收查询使能
enet_rxprocess_check_recovery	检测并恢复接收流程
enet_txfifo_flush	刷新ENET发送FIFO，并等待刷新操作完成
enet_current_desc_address_get	获取当前发送/接收描述符地址、当前缓冲区地址、描述符列表首地址
enet_desc_information_get	获取发送/接收描述符详细信息
enet_missed_frame_counter_get	获取接收丢弃帧数
描述符功能函数	
enet_desc_flag_get	获取ENET模块DMA描述符标志位
enet_desc_flag_set	设置ENET模块DMA描述符标志位
enet_desc_flag_clear	清除ENET模块DMA描述符标志位
enet_rx_desc_immediate_receive_complete_interrupt	接收完成后立即置位ENET_DMA_STAT寄存器的RS位
enet_rx_desc_delay_receive_complete_interrupt	接收完成后延迟指定时间再置位ENET_DMA_STAT寄存器的RS位
enet_rxframe_drop	丢弃当前接收到的帧
enet_dma_feature_enable	使能ENET模块DMA相关功能
enet_dma_feature_disable	禁能ENET模块DMA相关功能
增强型描述符功能函数	
enet_rx_desc_enhanced_status_get	获取接收描述符增强状态标志位信息
enet_desc_select_enhanced_mode	配置DMA描述符为增强型描述符
enet_ptp_enhanced_descriptors_chain_init	初始化具有PTP功能的增强型DMA接收/发送描述符为链模式
enet_ptp_enhanced_descriptors_ring_init	初始化具有PTP功能的增强型DMA接收/发送描述符为环模式
enet_ptpframe_receive_enhanced_mode	在PTP模式下处理当前接收到的帧，并将当前增强型描述符中存储的接收帧数据和时间戳拷贝到指定区域
enet_ptpframe_transmit_enhanced_mode	在PTP模式下将指定区域内的数据拷贝到当前增强型发送描述符中，并同时时间戳一起发送
常规型描述符功能函数	
enet_desc_select_normal_mode	配置DMA描述符为常规型描述符
enet_ptp_normal_descriptors_chain_init	初始化具有PTP功能的普通型DMA接收/发送描述符为链模式
enet_ptp_normal_descriptors_ring_init	初始化具有PTP功能的普通型DMA接收/发送描述符为环模式
enet_ptpframe_receive_normal_mode	在PTP模式下处理当前接收到的帧，并将当前普通型描述符中

库函数名称	库函数描述
	存储的接收帧数据和时间戳拷贝到指定区域
enet_ptpframe_transmit_normal_mode	在PTP模式下将指定区域内的数据拷贝到当前普通型发送描述符中，并同时时间戳一起发送
WUM功能函数	
enet_wum_filter_register_pointer_reset	远程唤醒帧过滤器寄存器指针复位
enet_wum_filter_config	配置远程唤醒帧寄存器
enet_wum_feature_enable	使能ENET模块唤醒管理相关功能
enet_wum_feature_disable	禁能ENET模块唤醒管理相关功能
MSC功能函数	
enet_msc_counters_reset	复位MAC统计计数器组
enet_msc_feature_enable	使能MAC统计计数器相关功能
enet_msc_feature_disable	禁能MAC统计计数器相关功能
enet_msc_counters_preset_config	配置MAC统计计数器的预设模式
enet_msc_counters_get	获取MAC相关统计计数器值
PTP功能函数	
enet_ptp_subsecond_2_nanosecond	将亚秒值变为纳秒值
enet_ptp_nanosecond_2_subsecond	将纳秒值变为亚秒值
enet_ptp_feature_enable	使能PTP相关功能
enet_ptp_feature_disable	禁能PTP相关功能
enet_ptp_timestamp_function_config	配置PTP时间戳相关功能
enet_ptp_subsecond_increment_config	配置PTP系统时间亚秒增加值
enet_ptp_timestamp_addend_config	精调模式下PTP时钟频率校准配置
enet_ptp_timestamp_update_config	初始化时用于替换系统时间，在更新时表示在系统时间上加上或减去的秒值
enet_ptp_expected_time_config	配置PTP期望时间
enet_ptp_system_time_get	获取PTP当前系统时间
enet_ptp_pps_output_frequency_config	配置PPS输出频率
enet_ptp_start	配置和启动PTP时间戳计数器
enet_ptp_finecorrection_adjfreq	通过PTP时间戳加数寄存器精调系统时间
enet_ptp_coarsecorrection_systime_update	粗调系统时间
enet_ptp_finecorrection_settime	精调系统时间
enet_ptp_flag_get	获取PTP标志位状态
内部函数	
enet_initpara_reset	复位 ENET initpara struct, 需在enet_initpara_config()函数前调用

## 结构体 `enet_initpara_struct`

表 3-246. 结构体 `enet_initpara_struct`

成员名称	功能描述
<code>option_enable</code>	选择配置功能
<code>forward_frame</code>	发送帧相关配置参数
<code>dmabus_mode</code>	DMA总线模式相关配置参数
<code>dma_maxburst</code>	DMA最大传输相关配置参数
<code>dma_arbitration</code>	DMA接收/发送仲裁相关配置参数
<code>store_forward_mode</code>	存储转发模式相关配置参数
<code>dma_function</code>	DMA控制相关配置参数
<code>vlan_config</code>	VLAN标签相关配置参数
<code>flow_control</code>	流控相关配置参数
<code>hashtable_high</code>	hash列表高32位相关配置参数
<code>hashtable_low</code>	hash列表低32位相关配置参数
<code>framesfilter_mode</code>	帧过滤器控制相关配置参数
<code>halfduplex_param</code>	半双工相关配置参数
<code>timer_config</code>	帧发送计数器相关配置参数
<code>interframegap</code>	帧间隔相关配置参数

## 结构体 `enet_descriptors_struct`

表 3-247. 结构体 `enet_descriptors_struct`

成员名称	功能描述
<code>status</code>	描述符状态位
<code>control_buffer_size</code>	描述符控制位及缓冲区1、2长度
<code>buffer1_addr</code>	缓冲区1地址指针/时间戳低
<code>buffer2_next_desc_addr</code>	缓冲区2或下一描述符地址指针/时间戳高
<code>extended_status</code>	增强型描述符状态
<code>reserved</code>	保留
<code>timestamp_low</code>	增强型描述符时间戳低位
<code>timestamp_high</code>	增强型描述符时间戳高位

## 结构体 `enet_ptp_systime_struct`

表 3-248. 结构体 `enet_ptp_systime_struct`

成员名称	功能描述
<code>second</code>	系统时间（单位秒）
<code>subsecond</code>	系统时间（单位纳秒）
<code>sign</code>	系统时间符号位

## 枚举类型 `enet_flag_enum`

表 3-249. 枚举类型 `enet_flag_enum`

成员名称	功能描述
ENET_MAC_FLAG_MPKR	接收到魔术帧标志位
ENET_MAC_FLAG_WUFR	接收到唤醒帧标志位
ENET_MAC_FLAG_FLOWCONTROL	流控状态标志位
ENET_MAC_FLAG_WUM	WUM状态标志位
ENET_MAC_FLAG_MSC	MSC状态标志位
ENET_MAC_FLAG_MSCR	MSC接收状态标志位
ENET_MAC_FLAG_MSCT	MSC发送状态标志位
ENET_MAC_FLAG_TMST	时间戳触发状态标志位
ENET_PTP_FLAG_TSSCO	时间戳秒计数溢出标志位
ENET_PTP_FLAG_TTM	目标时间匹配标志位
ENET_MSC_FLAG_RFCE	接收帧CRC错误标志位
ENET_MSC_FLAG_RFAE	接收帧对齐错误标志位
ENET_MSC_FLAG_RGUF	接收到“好”的单播帧标志位
ENET_MSC_FLAG_TGFSC	发送“好”的帧时仅遇到1个冲突标志位
ENET_MSC_FLAG_TGFMSC	发送“好”的帧时遇到1个以上冲突
ENET_MSC_FLAG_TGF	发送“好”的帧标志位
ENET_DMA_FLAG_TS	发送状态标志位
ENET_DMA_FLAG_TPS	发送流程停止状态标志位
ENET_DMA_FLAG_TBU	发送缓冲区不可用状态标志位
ENET_DMA_FLAG_	发送jabber超时状态标志位

TJT	
ENET_DMA_FLAG_ RO	接收溢出状态标志位
ENET_DMA_FLAG_ TU	发送下溢状态标志位
ENET_DMA_FLAG_ RS	接收状态标志位
ENET_DMA_FLAG_ RBU	接收缓冲区不可用状态标志位
ENET_DMA_FLAG_ RPS	接受流程停止状态标志位
ENET_DMA_FLAG_ RWT	接收看门狗超时状态标志位
ENET_DMA_FLAG_ ET	早发送状态标志位
ENET_DMA_FLAG_ FBE	致命总线错误状态标志位
ENET_DMA_FLAG_ ER	早接收状态标志位
ENET_DMA_FLAG_ AI	异常中断汇总标志位
ENET_DMA_FLAG_ NI	正常中断汇总标志位
ENET_DMA_FLAG_ EB_DMA_ERROR	DMA错误标志位
ENET_DMA_FLAG_ EB_TRANSFER_E RROR	发送错误标志位
ENET_DMA_FLAG_ EB_ACCESS_ERR OR	DMA访问错误标志位
ENET_DMA_FLAG_ MSC	MSC状态标志位
ENET_DMA_FLAG_ WUM	WUM状态标志位
ENET_DMA_FLAG_ TST	时间戳触发状态标志位

#### 枚举类型 `enet_flag_clear_enum`

表 3-250. 枚举类型 `enet_flag_clear_enum`

成员名称	功能描述
<code>ENET_DMA_FLAG_</code>	发送状态标志位清除

<i>TS_CLR</i>	
<i>ENET_DMA_FLAG_TPS_CLR</i>	发送流程停止状态标志位清除
<i>ENET_DMA_FLAG_TBU_CLR</i>	发送缓冲区不可用状态标志位清除
<i>ENET_DMA_FLAG_TJT_CLR</i>	发送jabber超时状态标志位清除
<i>ENET_DMA_FLAG_RO_CLR</i>	接收溢出状态标志位清除
<i>ENET_DMA_FLAG_TU_CLR</i>	发送下溢状态标志位清除
<i>ENET_DMA_FLAG_RS_CLR</i>	接收状态标志位清除
<i>ENET_DMA_FLAG_RBU_CLR</i>	接收缓冲区不可用状态标志位清除
<i>ENET_DMA_FLAG_RPS_CLR</i>	接受流程停止状态标志位清除
<i>ENET_DMA_FLAG_RWT_CLR</i>	接收看门狗超时状态标志位清除
<i>ENET_DMA_FLAG_ET_CLR</i>	早发送状态标志位清除
<i>ENET_DMA_FLAG_FBE_CLR</i>	致命总线错误状态标志位清除
<i>ENET_DMA_FLAG_ER_CLR</i>	早接收状态标志位清除
<i>ENET_DMA_FLAG_AI_CLR</i>	异常中断汇总标志位清除
<i>ENET_DMA_FLAG_NI_CLR</i>	正常中断汇总标志位清除

### 枚举类型 `enet_int_enum`

表 3-251. 枚举类型 `enet_int_enum`

成员名称	功能描述
<i>ENET_MAC_INT_WUMIM</i>	WUM中断屏蔽
<i>ENET_MAC_INT_TMSTIM</i>	时间戳触发中断屏蔽
<i>ENET_MSC_INT_RFCEIM</i>	接收帧CRC错误中断屏蔽
<i>ENET_MSC_INT_RFAEIM</i>	接收帧对齐错误中断屏蔽
<i>ENET_MSC_INT_R</i>	接收“好”单播帧中断屏蔽

<i>GUFIM</i>	
<i>ENET_MSC_INT_T GFSCIM</i>	仅遇到1个冲突后发送“好”帧中断屏蔽
<i>ENET_MSC_INT_T GFMSCIM</i>	遇到1个以上冲突后发送“好”帧中断屏蔽
<i>ENET_MSC_INT_T GFIM</i>	发送“好”的帧的中断屏蔽
<i>ENET_DMA_INT_T E</i>	发送中断使能
<i>ENET_DMA_INT_T PSIE</i>	发送流程停止中断使能
<i>ENET_DMA_INT_T BUIE</i>	发送缓冲区不可用中断使能
<i>ENET_DMA_INT_T JTIE</i>	发送jabber超时中断使能
<i>ENET_DMA_INT_R OIE</i>	接收溢出中断使能
<i>ENET_DMA_INT_T UIE</i>	发送下溢中断使能
<i>ENET_DMA_INT_R E</i>	接收中断使能
<i>ENET_DMA_INT_R BUIE</i>	接收缓冲区不可用中断使能
<i>ENET_DMA_INT_R PSIE</i>	接收流程停止中断使能
<i>ENET_DMA_INT_R WTIE</i>	接收看门狗超时中断使能
<i>ENET_DMA_INT_E TIE</i>	早发送中断使能
<i>ENET_DMA_INT_F BEIE</i>	致命总线错误中断使能
<i>ENET_DMA_INT_E RIE</i>	早接收中断使能
<i>ENET_DMA_INT_AI E</i>	异常中断汇总使能
<i>ENET_DMA_INT_NI E</i>	正常中断汇总使能

#### 枚举类型 `enet_int_flag_enum`

表 3-252. 枚举类型 `enet_int_flag_enum`

成员名称	功能描述
<i>ENET_MAC_INT_F</i>	WUM中断标志位

LAG_WUM	
ENET_MAC_INT_F LAG_MSC	MSC中断标志位
ENET_MAC_INT_F LAG_MSCR	MSC接收中断标志位
ENET_MAC_INT_F LAG_MSCT	MSC发送中断标志位
ENET_MAC_INT_F LAG_TMST	时间戳触发中断标志位
ENET_MSC_INT_F LAG_RFCE	接收帧CRC错误中断标志位
ENET_MSC_INT_F LAG_RFAE	接收帧对齐错误中断标志位
ENET_MSC_INT_F LAG_RGUF	接收“好”单播帧中断标志位
ENET_MSC_INT_F LAG_TGFSC	仅遇到1个冲突后发送“好”帧中断标志位
ENET_MSC_INT_F LAG_TGFMSC	遇到1个以上冲突后发送“好”帧中断标志位
ENET_MSC_INT_F LAG_TGF	发送“好”的帧的中断标志位
ENET_DMA_INT_F LAG_TS	发送中断标志位
ENET_DMA_INT_F LAG_TPS	发送流程停止中断标志位
ENET_DMA_INT_F LAG_TBU	发送缓冲区不可用中断标志位
ENET_DMA_INT_F LAG_TJT	发送jabber超时中断标志位
ENET_DMA_INT_F LAG_RO	接收溢出中断标志位
ENET_DMA_INT_F LAG_TU	发送下溢中断标志位
ENET_DMA_INT_F LAG_RS	接收中断标志位
ENET_DMA_INT_F LAG_RBU	接收缓冲区不可用中断标志位
ENET_DMA_INT_F LAG_RPS	接收流程停止中断标志位
ENET_DMA_INT_F LAG_RWT	接收看门狗超时中断标志位
ENET_DMA_INT_F LAG_ET	早发送中断标志位

ENET_DMA_INT_F LAG_FBE	致命总线错误中断标志位
ENET_DMA_INT_F LAG_ER	早接收中断标志位
ENET_DMA_INT_F LAG_AI	异常中断汇总中断标志位
ENET_DMA_INT_F LAG_NI	正常中断汇总中断标志位
ENET_DMA_INT_F LAG_MSC	MSC中断标志位
ENET_DMA_INT_F LAG_WUM	WUM中断标志位
ENET_DMA_INT_F LAG_TST	时间戳触发中断标志位

### 枚举类型 `enet_int_flag_clear_enum`

表 3-253. 枚举类型 `enet_int_flag_clear_enum`

成员名称	功能描述
ENET_DMA_INT_F LAG_TS_CLR	发送中断标志位清除
ENET_DMA_INT_F LAG_TPS_CLR	发送流程停止中断标志位清除
ENET_DMA_INT_F LAG_TBU_CLR	发送缓冲区不可用中断标志位清除
ENET_DMA_INT_F LAG_TJT_CLR	发送jabber超时中断标志位清除
ENET_DMA_INT_F LAG_RO_CLR	接收溢出中断标志位清除
ENET_DMA_INT_F LAG_TU_CLR	发送下溢中断标志位清除
ENET_DMA_INT_F LAG_RS_CLR	接收中断标志位清除
ENET_DMA_INT_F LAG_RBU_CLR	接收缓冲区不可用中断标志位清除
ENET_DMA_INT_F LAG_RPS_CLR	接收流程停止中断标志位清除
ENET_DMA_INT_F LAG_RWT_CLR	接收看门狗超时中断标志位清除
ENET_DMA_INT_F LAG_ET_CLR	早发送中断标志位清除
ENET_DMA_INT_F LAG_FBE_CLR	致命总线错误中断标志位清除

ENET_DMA_INT_F LAG_ER_CLR	早接收中断标志位清除
ENET_DMA_INT_F LAG_AI_CLR	异常中断汇总中断标志位清除
ENET_DMA_INT_F LAG_NI_CLR	正常中断汇总中断标志位清除

### 枚举类型 enet\_desc\_reg\_enum

表 3-254. 枚举类型 enet\_desc\_reg\_enum

成员名称	功能描述
ENET_RX_DESC_T ABLE	接收描述符列表首地址
ENET_RX_CURRE NT_DESC	当前DMA控制器使用的接收描述符地址
ENET_RX_CURRE NT_BUFFER	当前DMA控制器使用的接收描述符缓冲区地址
ENET_TX_DESC_T ABLE	发送描述符列表首地址
ENET_TX_CURRE NT_DESC	当前DMA控制器使用的发送描述符地址
ENET_TX_CURRE NT_BUFFER	当前DMA控制器使用的发送描述符缓冲区地址

### 枚举类型 enet\_msc\_counter\_enum

表 3-255. 枚举类型 enet\_msc\_counter\_enum

成员名称	功能描述
ENET_MSC_TX_S CCNT	MSC 1次冲突后发送“好”帧的计数器
ENET_MSC_TX_M SCCNT	MSC 1次以上冲突后发送“好”帧的计数器
ENET_MSC_TX_T GFCNT	MSC发送“好”帧计数器
ENET_MSC_RX_R FCECNT	MSC CRC错误接收帧计数器
ENET_MSC_RX_R FAECNT	MSC对齐错误接收帧计数器
ENET_MSC_RX_R GUFCNT	MSC“好”单播帧接收帧计数器

## 枚举类型 `enet_option_enum`

表 3-256. 枚举类型 `enet_option_enum`

成员名称	功能描述
<code>FORWARD_OPTION</code>	选择配置帧通过功能相关参数
<code>DMABUS_OPTION</code>	选择配置DMA总线模式相关参数
<code>DMA_MAXBURST_OPTION</code>	选择配置DMA最大突发传输相关参数
<code>DMA_ARBITRATION_OPTION</code>	选择配置DMA仲裁相关参数
<code>STORE_OPTION</code>	选择配置存储转发模式相关参数
<code>DMA_OPTION</code>	选择配置DMA相关参数
<code>VLAN_OPTION</code>	选择配置VLAN相关参数
<code>FLOWCTL_OPTION</code>	选择配置流控相关参数
<code>HASHH_OPTION</code>	选择配置HASH_H相关参数
<code>HASHL_OPTION</code>	选择配置HASH_L相关参数
<code>FILTER_OPTION</code>	选择配置帧过滤器相关参数
<code>HALFDUPLEX_OPTION</code>	选择配置半双工模式相关参数
<code>TIMER_OPTION</code>	选择配置计数器相关参数
<code>INTERFRAMEGAP_OPTION</code>	选择配置帧间隔相关参数

## 枚举类型 `enet_mediamode_enum`

表 3-257. 枚举类型 `enet_mediamode_enum`

成员名称	功能描述
<code>ENET_AUTO_NEGOTIATION</code>	PHY自协商
<code>ENET_100M_FULLDUPLEX</code>	100Mbit/s, 全双工
<code>ENET_100M_HALFDUPLEX</code>	100Mbit/s, 半双工
<code>ENET_10M_FULLDUPLEX</code>	10Mbit/s, 全双工
<code>ENET_10M_HALFDUPLEX</code>	10Mbit/s, 半双工
<code>ENET_LOOPBACK_MODE</code>	MII模式下的回环模式

### 枚举类型 `enet_chksumconf_enum`

表 3-258. 枚举类型 `enet_chksumconf_enum`

成员名称	功能描述
<code>ENET_NO_AUTOCHECKSUM</code>	关闭IP帧校验和功能
<code>ENET_AUTOCHECKSUM_DROP_FAILFRAMES</code>	使能IP帧校验和功能
<code>ENET_AUTOCHECKSUM_ACCEPT_FAILFRAMES</code>	使能IP帧校验和功能，不丢弃仅有载荷错误的帧

### 枚举类型 `enet_frmrecept_enum`

表 3-259. 枚举类型 `enet_frmrecept_enum`

成员名称	功能描述
<code>ENET_PROMISCUOUS_MODE</code>	使能混杂模式
<code>ENET_RECEIVEALL</code>	接收所有帧
<code>ENET_BROADCAST_FRAMES_PASS</code>	接收广播帧
<code>ENET_BROADCAST_FRAMES_DROP</code>	禁止接收广播帧

### 枚举类型 `enet_registers_type_enum`

表 3-260. 枚举类型 `enet_registers_type_enum`

成员名称	功能描述
<code>ALL_MAC_REG</code>	寄存器范围从ENET_MAC_CFG到ENET_MAC_FCTH
<code>ALL_MSC_REG</code>	寄存器范围从ENET_MSC_CTL到ENET_MSC_RGUFCNT
<code>ALL_PTP_REG</code>	寄存器范围从ENET_PTP_TSCTL到ENET_PTP_PPSCTL
<code>ALL_DMA_REG</code>	寄存器范围从ENET_DMA_BCTL到ENET_DMA_CRBADDR

### 枚举类型 `enet_dmadirection_enum`

表 3-261. 枚举类型 `enet_dmadirection_enum`

成员名称	功能描述
<code>ENET_DMA_TX</code>	DMA Tx描述符
<code>ENET_DMA_RX</code>	DMA Rx描述符

### 枚举类型 `enet_phydirection_enum`

表 3-262. 枚举类型 `enet_phydirection_enum`

成员名称	功能描述
<code>ENET_PHY_WRITE</code>	向PHY寄存器写数据
<code>ENET_PHY_READ</code>	从PHY寄存器读数据

### 枚举类型 `enet_regdirection_enum`

表 3-263. 枚举类型 `enet_regdirection_enum`

成员名称	功能描述
<code>ENET_REG_READ</code>	读寄存器
<code>ENET_REG_WRITE</code>	写寄存器

### 枚举类型 `enet_macaddress_enum`

表 3-264. 枚举类型 `enet_macaddress_enum`

成员名称	功能描述
<code>ENET_MAC_ADDR</code> <code>ESS0</code>	配置MAC address 0过滤器
<code>ENET_MAC_ADDR</code> <code>ESS1</code>	配置MAC address 1过滤器
<code>ENET_MAC_ADDR</code> <code>ESS2</code>	配置MAC address 2过滤器
<code>ENET_MAC_ADDR</code> <code>ESS3</code>	配置MAC address 3过滤器

### 枚举类型 `enet_descstate_enum`

表 3-265. 枚举类型 `enet_descstate_enum`

成员名称	功能描述
<code>TXDESC_COLLISION_COUNT</code>	帧发送出去前出现的冲突次数
<code>TXDESC_BUFFER_1_ADDR</code>	发送帧的缓冲区地址
<code>RXDESC_FRAME_LENGTH</code>	接收帧长度
<code>RXDESC_BUFFER_1_SIZE</code>	接收缓冲区1大小
<code>RXDESC_BUFFER_2_SIZE</code>	接收缓冲区2大小
<code>RXDESC_BUFFER_1_ADDR</code>	接收帧的缓冲区地址

枚举类型 `enet_msc_preset_enum`表 3-266. 枚举类型 `enet_msc_preset_enum`

成员名称	功能描述
<code>ENET_MSC_PRES ET_NONE</code>	关闭MSC计数器预设功能
<code>ENET_MSC_PRES ET_HALF</code>	预设为最大值一半
<code>ENET_MSC_PRES ET_FULL</code>	预设为近似全值

函数 `enet_deinit`

函数`enet_deinit`描述见下表：

表 3-267. 函数 `enet_deinit`

函数名称	<code>enet_deinit</code>
函数原型	<code>void enet_deinit(void);</code>
功能描述	复位ENET模块及相关软件初始化所需结构体
先决条件	-
被调用函数	<code>rcu_periph_reset_enable()/rcu_periph_reset_disable()/enet_initpara_reset()</code>
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize the ENET */
```

```
enet_deinit( );
```

函数 `enet_initpara_config`

函数`enet_initpara_config`描述见下表：

表 3-268. 函数 `enet_initpara_config`

函数名称	<code>enet_initpara_config</code>
函数原型	<code>void enet_initpara_config(enet_option_enum option, uint32_t para)</code>
功能描述	配置ENET模块的各类不常用功能。当 <code>enet_init()</code> 函数无法满足所需实现功能时调用，必须在 <code>enet_init()</code> 函数之前调用
先决条件	<code>enet_initpara_reset(void)</code>
被调用函数	-
输入参数{in}	

option	ENET模块功能选项，根据选择需要使用不同参数进行配置，参考 <a href="#">表3-256</a> 。 <a href="#">枚举类型enet_option_enum</a> ，下列参数仅可选择一个
FORWARD_OPTION	选择配置帧通过功能相关参数
DMABUS_OPTION	选择配置DMA总线模式相关参数
DMA_MAXBURST_OPTION	选择配置DMA最大突发传输相关参数
DMA_ARBITRATION_OPTION	选择配置DMA仲裁相关参数
STORE_OPTION	选择配置存储转发模式相关参数
DMA_OPTION	选择配置DMA相关参数
VLAN_OPTION	选择配置VLAN相关参数
FLOWCTL_OPTION	选择配置流控相关参数
HASHH_OPTION	选择配置HASH_H相关参数
HASHL_OPTION	选择配置HASH_L相关参数
FILTER_OPTION	选择配置帧过滤器相关参数
HALFDUPLEX_OPTION	选择配置半双工模式相关参数
TIMER_OPTION	选择配置计数器相关参数
INTERFRAMEGAP_OPTION	选择配置帧间隔相关参数
输入参数{in}	
para (该参数值需根据option参数对应值进行选择)	下列参数值均可以被配置 例如：para = (value1   value2   value3...)
当option参数值为FORWARD_OPTION时	
value1	ENET_AUTO_PADCRC_DROP_ENABLE / ENET_AUTO_PADCRC_DROP_DISABLE
value2	ENET_TYPEFRAME_CRC_DROP_ENABLE / ENET_TYPEFRAME_CRC_DROP_DISABLE
value3	ENET_FORWARD_ERRFRAMES_ENABLE / ENET_FORWARD_ERRFRAMES_DISABLE
value4	ENET_FORWARD_UNDEFSZ_GOODFRAMES_ENABLE / ENET_FORWARD_UNDEFSZ_GOODFRAMES_DISABLE
当option参数值为DMABUS_OPTION时	
value1	ENET_ADDRESS_ALIGN_ENABLE / ENET_ADDRESS_ALIGN_DISABLE
value2	ENET_FIXED_BURST_ENABLE / ENET_FIXED_BURST_DISABLE
value3	ENET_MIXED_BURST_ENABLE / ENET_MIXED_BURST_DISABLE
当option参数值为DMA_MAXBURST_OPTION时	
value1	ENET_RXDP_1BEAT / ENET_RXDP_2BEAT / ENET_RXDP_4BEAT / ENET_RXDP_8BEAT / ENET_RXDP_16BEAT / ENET_RXDP_32BEAT / ENET_RXDP_4xPGBL_4BEAT / ENET_RXDP_4xPGBL_8BEAT /

	<i>ENET_RXDP_4xPGBL_16BEAT / ENET_RXDP_4xPGBL_32BEAT / ENET_RXDP_4xPGBL_64BEAT / ENET_RXDP_4xPGBL_128BEAT</i>
value2	<i>ENET_PGBL_1BEAT / ENET_PGBL_2BEAT / ENET_PGBL_4BEAT / ENET_PGBL_8BEAT / ENET_PGBL_16BEAT / ENET_PGBL_32BEAT / ENET_PGBL_4xPGBL_4BEAT / ENET_PGBL_4xPGBL_8BEAT / ENET_PGBL_4xPGBL_16BEAT / ENET_PGBL_4xPGBL_32BEAT / ENET_PGBL_4xPGBL_64BEAT / ENET_PGBL_4xPGBL_128BEAT</i>
value3	<i>ENET_RXTX_DIFFERENT_PGBL / ENET_RXTX_SAME_PGBL</i>
当 <b>option</b> 参数值为DMA_ARBITRATION_OPTION时	
value1	<i>ENET_ARBITRATION_RXPRIORTX</i>
value2	<i>ENET_ARBITRATION_RXTX_1_1 / ENET_ARBITRATION_RXTX_2_1 / ENET_ARBITRATION_RXTX_3_1 / ENET_ARBITRATION_RXTX_4_1</i>
当 <b>option</b> 参数值为STORE_OPTION时	
value1	<i>ENET_RX_MODE_STOREFORWARD / ENET_RX_MODE_CUTTHROUGH</i>
value2	<i>ENET_TX_MODE_STOREFORWARD / ENET_TX_MODE_CUTTHROUGH</i>
value3	<i>ENET_RX_THRESHOLD_64BYTES / ENET_RX_THRESHOLD_32BYTES / ENET_RX_THRESHOLD_96BYTES / ENET_RX_THRESHOLD_128BYTES</i>
value4	<i>ENET_TX_THRESHOLD_64BYTES / ENET_TX_THRESHOLD_128BYTES / ENET_TX_THRESHOLD_192BYTES / ENET_TX_THRESHOLD_256BYTES / ENET_TX_THRESHOLD_40BYTES / ENET_TX_THRESHOLD_32BYTES / ENET_TX_THRESHOLD_24BYTES / ENET_TX_THRESHOLD_16BYTES</i>
当 <b>option</b> 参数值为DMA_OPTION时	
value1	<i>ENET_FLUSH_RXFRAME_ENABLE / ENET_FLUSH_RXFRAME_DISABLE</i>
value2	<i>ENET_SECONDFRAME_OPT_ENABLE / ENET_SECONDFRAME_OPT_DISABLE</i>
value3	<i>ENET_ENHANCED_DESCRIPTOR / ENET_NORMAL_DESCRIPTOR</i>
当 <b>option</b> 参数值为VLAN_OPTION时	
value1	<i>ENET_VLANTAGCOMPARISON_12BIT / ENET_VLANTAGCOMPARISON_16BIT</i>
value2	<i>MAC_VLT_VLTI(regval)</i>
当 <b>option</b> 参数值为FLOWCTL_OPTION时	
value1	<i>MAC_FCTL_PTM(regval)</i>
value2	<i>ENET_ZERO_QUANTA_PAUSE_ENABLE / ENET_ZERO_QUANTA_PAUSE_DISABLE</i>
value3	<i>ENET_PAUSETIME_MINUS4 / ENET_PAUSETIME_MINUS28 / ENET_PAUSETIME_MINUS144 / ENET_PAUSETIME_MINUS256</i>
value4	<i>ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDetect / ENET_UNIQUE_PAUSEDetect</i>
value5	<i>ENET_RX_FLOWCONTROL_ENABLE / ENET_RX_FLOWCONTROL_DISABLE</i>
value6	<i>ENET_TX_FLOWCONTROL_ENABLE / ENET_TX_FLOWCONTROL_DISABLE</i>
当 <b>option</b> 参数值为HASHH_OPTION时	

value1	0x0~0xFFFF FFFFU
当 <b>option</b> 参数值为HASHL_OPTION时	
value1	0x0~0xFFFF FFFFU
当 <b>option</b> 参数值为FILTER_OPTION时	
value1	ENET_SRC_FILTER_NORMAL_ENABLE / ENET_SRC_FILTER_INVERSE_ENABLE / ENET_SRC_FILTER_DISABLE
value2	ENET_DEST_FILTER_INVERSE_ENABLE / ENET_DEST_FILTER_INVERSE_DISABLE
value3	ENET_MULTICAST_FILTER_HASH_OR_PERFECT / ENET_MULTICAST_FILTER_HASH / ENET_MULTICAST_FILTER_PERFECT / ENET_MULTICAST_FILTER_NONE
value4	ENET_UNICAST_FILTER_EITHER / ENET_UNICAST_FILTER_HASH / ENET_UNICAST_FILTER_PERFECT
value5	ENET_PCFRM_PREVENT_ALL / ENET_PCFRM_PREVENT_PAUSEFRAME / ENET_PCFRM_FORWARD_ALL / ENET_PCFRM_FORWARD_FILTERED
当 <b>option</b> 参数值为HALFDUPLEX_OPTION时	
value1	ENET_CARRIERSENSE_ENABLE / ENET_CARRIERSENSE_DISABLE
value2	ENET_RECEIVEOWN_ENABLE / ENET_RECEIVEOWN_DISABLE
value3	ENET_RETRYTRANSMISSION_ENABLE / ENET_RETRYTRANSMISSION_DISABLE
value4	ENET_BACKOFFLIMIT_10 / ENET_BACKOFFLIMIT_8 / ENET_BACKOFFLIMIT_4 / ENET_BACKOFFLIMIT_1
value5	ENET_DEFERRALCHECK_ENABLE / ENET_DEFERRALCHECK_DISABLE
当 <b>option</b> 参数值为TIMER_OPTION时	
value1	ENET_WATCHDOG_ENABLE / ENET_WATCHDOG_DISABLE
value2	ENET_JABBER_ENABLE / ENET_JABBER_DISABLE
当 <b>option</b> 参数值为INTERFRAMEGAP_OPTION时	
value1	ENET_INTERFRAMEGAP_96BIT / ENET_INTERFRAMEGAP_88BIT / ENET_INTERFRAMEGAP_80BIT / ENET_INTERFRAMEGAP_72BIT / ENET_INTERFRAMEGAP_64BIT / ENET_INTERFRAMEGAP_56BIT / ENET_INTERFRAMEGAP_48BIT / ENET_INTERFRAMEGAP_40BIT
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config DMA option of the ENET */
```

```
enet_initpara_reset();
```

```
enet_initpara_config(DMA_OPTION, ENET_FLUSH_RXFRAME_ENABLE | ENET_SECONDFRAME_OPT_ENABLE | ENET_NORMAL_DESCRIPTOR);
```

函数 **enet\_init**

函数 **enet\_init** 描述见下表：

表 3-269. 函数 **enet\_init**

函数名称	enet_init
函数原型	ErrStatus enet_init(enet_mediamode_enum mediamode, enet_chksumconf_enum checksum, enet_frmrecept_enum recept);
功能描述	ENET模块初始化，配置用户最关心的功能
先决条件	enet_deinit ()
被调用函数	enet_phy_config()/enet_phy_write_read()/enet_default_init()
输入参数{in}	
mediamode	ENET通讯方式配置，参考 <a href="#">表3-257. 枚举类型enet_mediamode_enum</a> ，仅可选择唯一参数
ENET_AUTO_NEGOTIATION	PHY自协商
ENET_100M_FULLDUPLEX	100Mbit/s，全双工
ENET_100M_HALFDUPLEX	100Mbit/s，半双工
ENET_10M_FULLDUPLEX	10Mbit/s，全双工
ENET_10M_HALFDUPLEX	10Mbit/s，半双工
ENET_LOOPBACKMODE	MII模式下的回环模式
输入参数{in}	
checksum	IP帧数据校验和功能，参考 <a href="#">表3-258. 枚举类型enet_chksumconf_enum</a> ，仅可选择唯一参数
ENET_NO_AUTOCHECKSUM	关闭IP帧校验和功能
ENET_AUTOCHECKSUM_DROP_FAILFRAMES	使能IP帧校验和功能
ENET_AUTOCHECKSUM_ACCEPT_FAILFRAMES	使能IP帧校验和功能，不丢弃仅有载荷错误的帧
输入参数{in}	
recept	帧过滤功能，参考 <a href="#">表3-259. 枚举类型enet_frmrecept_enum</a> ，仅可选择唯一参数
ENET_PROMISCUOUS_MODE	使能混杂模式
ENET_RECEIVEALL	接收所有帧
ENET_BROADCAST	接收广播帧

<code>_FRAMES_PASS</code>	
<code>ENET_BROADCAST</code> <code>_FRAMES_DROP</code>	禁止接收广播帧
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR或SUCCESS

例如：

```
/* initialize ENET peripheral */
```

```
ErrStatus enet_init_status;
```

```
enet_init_status = enet_init(ENET_AUTO_NEGOTIATION, ENET_AUTOCHECKSUM_DROP_FAILFRAMES, ENET_BROADCAST_FRAMES_PASS);
```

### 函数 `enet_software_reset`

函数 `enet_software_reset` 描述见下表：

**表 3-270. 函数 `enet_software_reset`**

函数名称	<code>enet_software_reset</code>
函数原型	<code>ErrStatus enet_software_reset(void);</code>
功能描述	复位ENET寄存器，并检测CLK_TX/CLK_RX信号
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR或SUCCESS

例如：

```
/* reset all core internal registers located in CLK_TX and CLK_RX */
```

```
ErrStatus reval_state = ERROR;
```

```
reval_state = enet_software_reset();
```

### 函数 `enet_rxframe_size_get`

函数 `enet_rxframe_size_get` 描述见下表：

**表 3-271. 函数 `enet_rxframe_size_get`**

函数名称	<code>enet_rxframe_size_get</code>
函数原型	<code>uint32_t enet_rxframe_size_get(void);</code>

功能描述	检测接收帧是否有错误，正确时返回帧长度
先决条件	-
被调用函数	enet_rxframe_drop()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	取值范围：0x0 - 0x3FFF

例如：

```
/* check receive frame valid */

uint32_t reval;

reval = enet_rxframe_size_get();
```

### 函数 enet\_descriptors\_chain\_init

函数enet\_descriptors\_chain\_init描述见下表：

表 3-272. 函数 enet\_descriptors\_chain\_init

函数名称	enet_descriptors_chain_init
函数原型	void enet_descriptors_chain_init(enet_dmadirection_enum direction);
功能描述	初始化DMA接收/发送描述符为链模式
先决条件	-
被调用函数	-
输入参数{in}	
direction	想要初始化的描述符类型，参考 <a href="#">表3-261. 枚举类型 enet_dmadirection_enum</a> ， 下列参数仅可选择一个
ENET_DMA_TX	DMA Tx描述符
ENET_DMA_RX	DMA Rx描述符
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the DMA Tx/Rx descriptors's parameters in chain mode */

enet_descriptors_chain_init(ENET_DMA_TX);
```

### 函数 enet\_descriptors\_ring\_init

函数enet\_descriptors\_ring\_init描述见下表：

表 3-273. 函数 `enet_descriptors_ring_init`

函数名称	<code>enet_descriptors_ring_init</code>
函数原型	<code>void enet_descriptors_ring_init(enet_dmadirection_enum direction);</code>
功能描述	初始化DMA接收/发送描述符为环模式
先决条件	-
被调用函数	-
输入参数{in}	
<b>direction</b>	想要初始化的描述符类型，参考 <a href="#">表3-261. 枚举类型</a> <a href="#">enet_dmadirection_enum</a> , 下列参数仅可选择一个
<code>ENET_DMA_TX</code>	DMA Tx描述符
<code>ENET_DMA_RX</code>	DMA Rx描述符
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the DMA Tx/Rx descriptors's parameters in ring mode */
```

```
enet_descriptors_ring_init(ENET_DMA_TX);
```

### 函数 `enet_frame_receive`

函数`enet_frame_receive`描述见下表：

表 3-274. 函数 `enet_frame_receive`

函数名称	<code>enet_frame_receive</code>
函数原型	<code>ErrStatus enet_frame_receive(uint8_t *buffer, uint32_t bufsize);</code>
功能描述	处理当前接收到的帧，并将当前描述符中存储的接收帧数据拷贝到指定区域
先决条件	-
被调用函数	-
输入参数{in}	
<b>bufsize</b>	缓冲区长度，范围(0~1524)
输出参数{out}	
<b>buffer</b>	接收帧数据的缓冲区地址指针。如果输入NULL，用户需要在调用该函数之前将数据拷贝到用户缓冲区内
返回值	
<b>ErrStatus</b>	ERROR或SUCCESS

例如：

```
/* transfer received frame data to application buffer */
```

```
uint8_t data_buffer[1500];
```

```
uint32_t data_size;
```

```
enet_frame_receive(data_buffer, &data_size);
```

### 函数 `enet_frame_transmit`

函数`enet_frame_transmit`描述见下表：

表 3-275. 函数 `enet_frame_transmit`

函数名称	<code>enet_frame_transmit</code>
函数原型	<code>ErrStatus enet_frame_transmit(uint8_t *buffer, uint32_t length);</code>
功能描述	将指定区域内的数据拷贝到当前发送描述符中，并发送
先决条件	-
被调用函数	-
输入参数{in}	
<b>buffer</b>	等待发送的帧数据缓冲区地址指针。如果输入NULL，用户需要在调用该函数之前将待发送数据拷贝到描述符指定的位置
输入参数{in}	
<b>length</b>	待发送数据长度，范围(0~1524)
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR或SUCCESS

例如：

```
/* transfer buffer data of application */
```

```
uint8_t data_buffer[1500];
```

```
uint32_t data_size = 800;
```

```
enet_frame_transmit (data_buffer, data_size);
```

### 函数 `enet_transmit_checksum_config`

函数`enet_transmit_checksum_config`描述见下表：

表 3-276. 函数 `enet_transmit_checksum_config`

函数名称	<code>enet_transmit_checksum_config</code>
函数原型	<code>ErrStatus enet_transmit_checksum_config(enet_descriptors_struct *desc, uint32_t checksum);</code>
功能描述	配置发送帧校验和模式
先决条件	-
被调用函数	-
输入参数{in}	
<b>desc</b>	需要配置的描述符地址指针，结构体成员介绍参考 <a href="#">表3-247. 结构体 <code>enet_descriptors_struct</code></a>

输入参数{in}	
<b>checksum</b>	IP帧校验和配置，下列参数仅可选择一个
<i>ENET_CHECKSUM_DISABLE</i>	禁能校验和自动插入
<i>ENET_CHECKSUM_IPV4HEADER</i>	仅使能IP头校验和计算和插入
<i>ENET_CHECKSUM_TCPUDPICMP_SEGMENT</i>	TCP/UDP/ICMP校验和（除去伪报头）计算和插入
<i>ENET_CHECKSUM_TCPUDPICMP_FULL</i>	TCP/UDP/ICMP校验和计算和插入
输出参数{out}	
返回值	
<b>ErrStatus</b>	ERROR或SUCCESS

例如：

```
/* configure the transmit IP frame checksum offload calculation and insertion */
enet_descriptors_struct rx_desc;
enet_transmit_checksum_config(rx_desc, ENET_CHECKSUM_TCPUDPICMP_FULL);
```

### 函数 enet\_enable

函数enet\_enable描述见下表：

表 3-277. 函数 enet\_enable

函数名称	enet_enable
函数原型	void enet_enable(void);
功能描述	ENET Tx/Rx功能使能（包括ENET外设内的MAC和DMA模块）
先决条件	-
被调用函数	enet_tx_enable()/enet_rx_enable()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the ENET */
enet_enable();
```

函数 **enet\_disable**

函数enet\_disable描述见下表:

表 3-278. 函数 **enet\_disable**

函数名称	enet_disable
函数原型	void enet_disable(void);
功能描述	ENET Tx/Rx功能禁能（包括ENET外设内的MAC和DMA模块）
先决条件	-
被调用函数	enet_tx_disable()/enet_rx_disable()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the ENET */
```

```
enet_disable();
```

函数 **enet\_mac\_address\_set**

函数enet\_mac\_address\_set描述见下表:

表 3-279. 函数 **enet\_mac\_address\_set**

函数名称	enet_mac_address_set
函数原型	void enet_mac_address_set(enet_macaddress_enum mac_addr, uint8_t paddr[]);
功能描述	配置MAC地址
先决条件	-
被调用函数	-
输入参数{in}	
mac_addr	选择何组MAC地址将被配置, 参考 <a href="#">表3-264. 枚举类型 enet_macaddress_enum</a> , 下列参数仅可选择一个
ENET_MAC_ADDRESSES0	配置MAC address 0过滤器
ENET_MAC_ADDRESSES1	配置MAC address 1过滤器
ENET_MAC_ADDRESSES2	配置MAC address 2过滤器
ENET_MAC_ADDRESSES3	配置MAC address 3过滤器
输入参数{in}	

<b>paddr</b>	存储MAC地址的缓冲区指针，小端存储 例如MAC地址为aa:bb:cc:dd:ee:22，缓冲区内数据为{22, ee, dd, cc, bb, aa}
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config mac address */
netif->hwaddr[0] = 0x02;
netif->hwaddr[1] = 0xaa;
netif->hwaddr[2] = 0xbb;
netif->hwaddr[3] = 0xcc;
netif->hwaddr[4] = 0xdd;
netif->hwaddr[5] = 0xee;

enet_mac_address_set(ENET_MAC_ADDRESS0, netif->hwaddr);
```

### 函数 enet\_mac\_address\_get

函数enet\_mac\_address\_get描述见下表：

表 3-280. 函数 enet\_mac\_address\_get

函数名称	enet_mac_address_get
函数原型	ErrStatus enet_mac_address_get(enet_macaddress_enum mac_addr, uint8_t paddr[], uint8_t bufsize)
功能描述	获取MAC地址
先决条件	-
被调用函数	-
输入参数{in}	
<b>mac_addr</b>	选择何组MAC地址将被配置，参考 <a href="#">表3-264. 枚举类型 enet_macaddress_enum</a> ，下列参数仅可选择一个
ENET_MAC_ADDRES0	配置MAC address 0过滤器
ENET_MAC_ADDRES1	配置MAC address 1过滤器
ENET_MAC_ADDRES2	配置MAC address 2过滤器
ENET_MAC_ADDRES3	配置MAC address 3过滤器
输出参数{out}	

<b>paddr</b>	存储MAC地址的缓冲区指针，小端存储 例如MAC地址为aa:bb:cc:dd:ee:22，缓冲区内数据为{22, ee, dd, cc, bb, aa}
<b>输入参数{in}</b>	
<b>bufsize</b>	缓存大小，6 - 255
<b>返回值</b>	
-	-

例如：

```
/* get mac address */
```

```
enet_mac_address_get (ENET_MAC_ADDRESS0, netif->hwaddr, 0x100);
```

### 函数 enet\_flag\_get

函数enet\_flag\_get描述见下表：

表 3-281. 函数 enet\_flag\_get

<b>函数名称</b>	enet_flag_get
<b>函数原型</b>	FlagStatus enet_flag_get(enet_flag_enum enet_flag);
<b>功能描述</b>	获取ENET模块MAC/MSC/PTP/DMA状态标志位
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>enet_flag</b>	ENET状态标志位，参考 <a href="#">表3-249. 枚举类型enet_flag_enum</a> ，下列参数仅可选一个
ENET_MAC_FLAG_MPKR	接收到魔术帧标志位
ENET_MAC_FLAG_WUFR	接收到唤醒帧标志位
ENET_MAC_FLAG_FLOWCONTROL	流控状态标志位
ENET_MAC_FLAG_WUM	WUM状态标志位
ENET_MAC_FLAG_MSC	MSC状态标志位
ENET_MAC_FLAG_MSCR	MSC接收状态标志位
ENET_MAC_FLAG_MSCT	MSC发送状态标志位
ENET_MAC_FLAG_TMST	时间戳触发状态标志位
ENET_PTP_FLAG_TSSCO	时间戳秒计数溢出标志位
ENET_PTP_FLAG_T	目标时间匹配标志位

<i>TM</i>	
<i>ENET_MSC_FLAG_RFCE</i>	接收帧CRC错误标志位
<i>ENET_MSC_FLAG_RFAE</i>	接收帧对齐错误标志位
<i>ENET_MSC_FLAG_RGUF</i>	接收到“好”的单播帧标志位
<i>ENET_MSC_FLAG_TGFSC</i>	发送“好”的帧时仅遇到1个冲突标志位
<i>ENET_MSC_FLAG_TGFMSC</i>	发送“好”的帧时遇到1个以上冲突
<i>ENET_MSC_FLAG_TGF</i>	发送“好”的帧标志位
<i>ENET_DMA_FLAG_TS</i>	发送状态标志位
<i>ENET_DMA_FLAG_TPS</i>	发送流程停止状态标志位
<i>ENET_DMA_FLAG_TBU</i>	发送缓冲区不可用状态标志位
<i>ENET_DMA_FLAG_TJT</i>	发送jabber超时状态标志位
<i>ENET_DMA_FLAG_RO</i>	接收溢出状态标志位
<i>ENET_DMA_FLAG_TU</i>	发送下溢状态标志位
<i>ENET_DMA_FLAG_RS</i>	接收状态标志位
<i>ENET_DMA_FLAG_RBU</i>	接收缓冲区不可用状态标志位
<i>ENET_DMA_FLAG_RPS</i>	接受流程停止状态标志位
<i>ENET_DMA_FLAG_RWT</i>	接收看门狗超时状态标志位
<i>ENET_DMA_FLAG_ET</i>	早发送状态标志位
<i>ENET_DMA_FLAG_FBE</i>	致命总线错误状态标志位
<i>ENET_DMA_FLAG_ER</i>	早接收状态标志位
<i>ENET_DMA_FLAG_AI</i>	异常中断汇总标志位
<i>ENET_DMA_FLAG_NI</i>	正常中断汇总标志位

ENET_DMA_FLAG_ EB_DMA_ERROR	DMA错误标志位
ENET_DMA_FLAG_ EB_TRANSFER_ER ROR	发送错误标志位
ENET_DMA_FLAG_ EB_ACCESS_ERRO R	DMA访问错误标志位
ENET_DMA_FLAG_ MSC	MSC状态标志位
ENET_DMA_FLAG_ WUM	WUM状态标志位
ENET_DMA_FLAG_ TST	时间戳触发状态标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* check whether the specified flag bit is set */
```

```
enet_flag_get (ENET_DMA_FLAG_RS);
```

### 函数 enet\_flag\_clear

函数enet\_flag\_clear描述见下表：

表 3-282. 函数 enet\_flag\_clear

函数名称	enet_flag_clear
函数原型	void enet_flag_clear(enet_flag_clear_enum enet_flag);
功能描述	清除ENET状态标志位
先决条件	-
被调用函数	-
输入参数{in}	
enet_flag	ENET模块DMA标志位清除，参考 <a href="#">表3-250. 枚举类型enet_flag_clear_enum</a> ， 下列参数仅可选择一个
ENET_DMA_FLAG_ TS_CLR	发送状态标志位清除
ENET_DMA_FLAG_ TPS_CLR	发送流程停止状态标志位清除
ENET_DMA_FLAG_ TBU_CLR	发送缓冲区不可用状态标志位清除
ENET_DMA_FLAG_	发送jabber超时状态标志位清除

TJT_CLR	
ENET_DMA_FLAG_RO_CLR	接收溢出状态标志位清除
ENET_DMA_FLAG_TU_CLR	发送下溢状态标志位清除
ENET_DMA_FLAG_RS_CLR	接收状态标志位清除
ENET_DMA_FLAG_RBU_CLR	接收缓冲区不可用状态标志位清除
ENET_DMA_FLAG_RPS_CLR	接受流程停止状态标志位清除
ENET_DMA_FLAG_RWT_CLR	接收看门狗超时状态标志位清除
ENET_DMA_FLAG_ET_CLR	早发送状态标志位清除
ENET_DMA_FLAG_FBE_CLR	致命总线错误状态标志位清除
ENET_DMA_FLAG_ER_CLR	早接收状态标志位清除
ENET_DMA_FLAG_AI_CLR	异常中断汇总标志位清除
ENET_DMA_FLAG_NI_CLR	正常中断汇总标志位清除
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the specified flag bit */
```

```
enet_flag_clear(ENET_DMA_FLAG_RS_CLR);
```

### 函数 enet\_interrupt\_enable

函数enet\_interrupt\_enable描述见下表：

表 3-283. 函数 enet\_interrupt\_enable

函数名称	enet_interrupt_enable
函数原型	void enet_interrupt_enable(enet_int_enum enet_int);
功能描述	使能ENET模块MAC/MSC/DMA中断
先决条件	-
被调用函数	-
输入参数{in}	

enet_int	ENET中断，参考 <a href="#">表3-251. 枚举类型enet_int_enum</a> ，下列参数仅可选择一个
ENET_MAC_INT_WUMIM	WUM中断屏蔽
ENET_MAC_INT_TSTMIM	时间戳触发中断屏蔽
ENET_MSC_INT_RFCEIM	接收帧CRC错误中断屏蔽
ENET_MSC_INT_RFAEIM	接收帧对齐错误中断屏蔽
ENET_MSC_INT_RGUFIM	接收“好”单播帧中断屏蔽
ENET_MSC_INT_TGFSCIM	仅遇到1个冲突后发送“好”帧中断屏蔽
ENET_MSC_INT_TGFMSCIM	遇到1个以上冲突后发送“好”帧中断屏蔽
ENET_MSC_INT_TGFIM	发送“好”的帧的中断屏蔽
ENET_DMA_INT_TIE	发送中断使能
ENET_DMA_INT_TPSIE	发送流程停止中断使能
ENET_DMA_INT_TBUIE	发送缓冲区不可用中断使能
ENET_DMA_INT_TJTIE	发送jabber超时中断使能
ENET_DMA_INT_ROIE	接收溢出中断使能
ENET_DMA_INT_TUIE	发送下溢中断使能
ENET_DMA_INT_RIE	接收中断使能
ENET_DMA_INT_RBUIE	接收缓冲区不可用中断使能
ENET_DMA_INT_RPSIE	接收流程停止中断使能
ENET_DMA_INT_RWTIE	接收看门狗超时中断使能
ENET_DMA_INT_ETIE	早发送中断使能
ENET_DMA_INT_FBEIE	致命总线错误中断使能
ENET_DMA_INT_ERIE	早接收中断使能

<i>ENET_DMA_INT_AIE</i>	异常中断汇总使能
<i>ENET_DMA_INT_NIE</i>	正常中断汇总使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable normal interrupt summary */
enet_interrupt_enable(ENET_DMA_INT_NIE);
```

### 函数 **enet\_interrupt\_disable**

函数 **enet\_interrupt\_disable** 描述见下表：

**表 3-284. 函数 **enet\_interrupt\_disable****

函数名称	enet_interrupt_disable
函数原型	void enet_interrupt_disable(enet_int_enum enet_int);
功能描述	禁能ENET模块MAC/MSC/DMA中断
先决条件	-
被调用函数	-
输入参数{in}	
<b>enet_int</b>	ENET中断，参考 <a href="#">表3-251. 枚举类型enet_int_enum</a> ，下列参数仅可选择一个
<i>ENET_MAC_INT_WUMIM</i>	WUM中断屏蔽
<i>ENET_MAC_INT_TSTM</i>	时间戳触发中断屏蔽
<i>ENET_MSC_INT_RFCEIM</i>	接收帧CRC错误中断屏蔽
<i>ENET_MSC_INT_RFAEIM</i>	接收帧对齐错误中断屏蔽
<i>ENET_MSC_INT_RGUFIM</i>	接收“好”单播帧中断屏蔽
<i>ENET_MSC_INT_TGFSCIM</i>	仅遇到1个冲突后发送“好”帧中断屏蔽
<i>ENET_MSC_INT_TGFMSCIM</i>	遇到1个以上冲突后发送“好”帧中断屏蔽
<i>ENET_MSC_INT_TGFIM</i>	发送“好”的帧的中断屏蔽
<i>ENET_DMA_INT_TIE</i>	发送中断使能

ENET_DMA_INT_TPSIE	发送流程停止中断使能
ENET_DMA_INT_TBUIE	发送缓冲区不可用中断使能
ENET_DMA_INT_TJTIE	发送jabber超时中断使能
ENET_DMA_INT_ROIE	接收溢出中断使能
ENET_DMA_INT_TUIE	发送下溢中断使能
ENET_DMA_INT_RIE	接收中断使能
ENET_DMA_INT_RBUIE	接收缓冲区不可用中断使能
ENET_DMA_INT_RPSIE	接收流程停止中断使能
ENET_DMA_INT_RWTIE	接收看门狗超时中断使能
ENET_DMA_INT_ETIE	早发送中断使能
ENET_DMA_INT_FBEIE	致命总线错误中断使能
ENET_DMA_INT_ERIE	早接收中断使能
ENET_DMA_INT_AIE	异常中断汇总使能
ENET_DMA_INT_NIE	正常中断汇总使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable normal interrupt summary */
enet_interrupt_disable(ENET_DMA_INT_NIE);
```

### 函数 enet\_interrupt\_flag\_get

函数enet\_interrupt\_flag\_get描述见下表：

表 3-285. 函数 enet\_interrupt\_flag\_get

函数名称	enet_interrupt_flag_get
------	-------------------------

函数原型	FlagStatus enet_interrupt_flag_get(enet_int_flag_enum int_flag);
功能描述	获取ENET模块MAC/MSC/DMA中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	ENET中断标志位，参考 <a href="#">表3-252. 枚举类型enet_int_flag_enum</a> ，下列参数仅可选择一个
ENET_MAC_INT_FL AG_WUM	WUM中断标志位
ENET_MAC_INT_FL AG_MSC	MSC中断标志位
ENET_MAC_INT_FL AG_MSCR	MSC接收中断标志位
ENET_MAC_INT_FL AG_MSCT	MSC发送中断标志位
ENET_MAC_INT_FL AG_TMST	时间戳触发中断标志位
ENET_MSC_INT_FL AG_RFCE	接收帧CRC错误中断标志位
ENET_MSC_INT_FL AG_RFAE	接收帧对齐错误中断标志位
ENET_MSC_INT_FL AG_RGUF	接收“好”单播帧中断标志位
ENET_MSC_INT_FL AG_TGFSC	仅遇到1个冲突后发送“好”帧中断标志位
ENET_MSC_INT_FL AG_TGFMSC	遇到1个以上冲突后发送“好”帧中断标志位
ENET_MSC_INT_FL AG_TGF	发送“好”的帧的中断标志位
ENET_DMA_INT_FL AG_TS	发送中断标志位
ENET_DMA_INT_FL AG_TPS	发送流程停止中断标志位
ENET_DMA_INT_FL AG_TBU	发送缓冲区不可用中断标志位
ENET_DMA_INT_FL AG_TJT	发送jabber超时中断标志位
ENET_DMA_INT_FL AG_RO	接收溢出中断标志位
ENET_DMA_INT_FL AG_TU	发送下溢中断标志位
ENET_DMA_INT_FL AG_RS	接收中断标志位

ENET_DMA_INT_FL AG_RBU	接收缓冲区不可用中断标志位
ENET_DMA_INT_FL AG_RPS	接收流程停止中断标志位
ENET_DMA_INT_FL AG_RWT	接收看门狗超时中断标志位
ENET_DMA_INT_FL AG_ET	早发送中断标志位
ENET_DMA_INT_FL AG_FBE	致命总线错误中断标志位
ENET_DMA_INT_FL AG_ER	早接收中断标志位
ENET_DMA_INT_FL AG_AI	异常中断汇总中断标志位
ENET_DMA_INT_FL AG_NI	正常中断汇总中断标志位
ENET_DMA_INT_FL AG_MSC	MSC中断标志位
ENET_DMA_INT_FL AG_WUM	WUM中断标志位
ENET_DMA_INT_FL AG_TST	时间戳触发中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* check whether the specified flag bit is set or not */
enet_interrupt_flag_get(ENET_DMA_INT_FLAG_RS);
```

### 函数 enet\_interrupt\_flag\_clear

函数enet\_interrupt\_flag\_clear描述见下表：

表 3-286. 函数 enet\_interrupt\_flag\_clear

函数名称	enet_interrupt_flag_clear
函数原型	void enet_interrupt_flag_clear(enet_int_flag_clear_enum int_flag_clear);
功能描述	禁能ENET中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag_clear	ENET中断标志位清除，参考 <a href="#">表3-253. 枚举类型enet_int_flag_clear_enum</a>

		，下列参数仅可选择一个
ENET_DMA_INT_FL AG_TS_CLR		发送中断标志位清除
ENET_DMA_INT_FL AG_TPS_CLR		发送流程停止中断标志位清除
ENET_DMA_INT_FL AG_TBU_CLR		发送缓冲区不可用中断标志位清除
ENET_DMA_INT_FL AG_TJT_CLR		发送jabber超时中断标志位清除
ENET_DMA_INT_FL AG_RO_CLR		接收溢出中断标志位清除
ENET_DMA_INT_FL AG_TU_CLR		发送下溢中断标志位清除
ENET_DMA_INT_FL AG_RS_CLR		接收中断标志位清除
ENET_DMA_INT_FL AG_RBU_CLR		接收缓冲区不可用中断标志位清除
ENET_DMA_INT_FL AG_RPS_CLR		接收流程停止中断标志位清除
ENET_DMA_INT_FL AG_RWT_CLR		接收看门狗超时中断标志位清除
ENET_DMA_INT_FL AG_ET_CLR		早发送中断标志位清除
ENET_DMA_INT_FL AG_FBE_CLR		致命总线错误中断标志位清除
ENET_DMA_INT_FL AG_ER_CLR		早接收中断标志位清除
ENET_DMA_INT_FL AG_AI_CLR		异常中断汇总中断标志位清除
ENET_DMA_INT_FL AG_NI_CLR		正常中断汇总中断标志位清除
输出参数{out}		
-		-
返回值		
-		-

例如：

```
/* clear receive status flag bit */
```

```
enet_interrupt_flag_clear(ENET_DMA_INT_FLAG_RS);
```

### 函数 enet\_tx\_enable

函数enet\_tx\_enable描述见下表：

表 3-287. 函数 enet\_tx\_enable

函数名称	enet_tx_enable
函数原型	void enet_tx_enable(void);
功能描述	ENET发送功能使能（包括ENET外设内的MAC和DMA模块）
先决条件	-
被调用函数	enet_txfifo_flush()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable transport function of MAC and DMA */
```

```
enet_tx_enable();
```

### 函数 enet\_tx\_disable

函数enet\_tx\_disable描述见下表：

表 3-288. 函数 enet\_tx\_disable

函数名称	enet_tx_disable
函数原型	void enet_tx_disable(void);
功能描述	ENET发送功能禁能（包括ENET外设内的MAC和DMA模块）
先决条件	-
被调用函数	enet_txfifo_flush()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable transport function of MAC and DMA */
```

```
enet_tx_disable();
```

### 函数 enet\_rx\_enable

函数enet\_rx\_enable描述见下表：

表 3-289. 函数 enet\_rx\_enable

函数名称	enet_rx_enable
------	----------------

函数原型	void enet_rx_enable(void);
功能描述	ENET接收功能使能（包括ENET外设内的MAC和DMA模块）
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 reception function of MAC and DMA */
```

```
enet_rx_enable();
```

### 函数 enet\_rx\_disable

函数enet\_rx\_disable描述见下表：

表 3-290. 函数 enet\_rx\_disable

函数名称	enet_rx_disable
函数原型	void enet_rx_disable(void);
功能描述	ENET发送功能禁能（包括ENET外设内的MAC和DMA模块）
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 reception function of MAC and DMA */
```

```
enet_rx_disable();
```

### 函数 enet\_registers\_get

函数enet\_registers\_get描述见下表：

表 3-291. 函数 enet\_registers\_get

函数名称	enet_registers_get
函数原型	void enet_registers_get(enet_registers_type_enum type, uint32_t *preg, uint32_t num);

功能描述	获取指定范围ENET寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
type	寄存器类型，参考 <a href="#">表3-260. 枚举类型enet_registers_type_enum</a> ，下列参数仅可选择一个
ALL_MAC_REG	寄存器范围从ENET_MAC_CFG到ENET_MAC_FCTH
ALL_MSC_REG	寄存器范围从ENET_MSC_CTL到ENET_MSC_RGUFCNT
ALL_PTP_REG	寄存器范围从ENET_PTP_TSCTL到ENET_PTP_PPSCTL
ALL_DMA_REG	寄存器范围从ENET_DMA_BCTL到ENET_DMA_CRBADDR
输入参数{in}	
num	想要获取的寄存器个数，范围(0~54)
输出参数{out}	
preg	存储寄存器值的应用缓冲区指针
返回值	
-	-

例如：

```
/* get all mac registers value */
uint32_t register_buffer[5];
enet_registers_get(ALL_MAC_REG, 5, register_buffer);
```

### 函数 enet\_debug\_status\_get

函数enet\_debug\_status\_get描述见下表：

表 3-292. 函数 enet\_debug\_status\_get

函数名称	enet_debug_status_get
函数原型	uint32_t enet_debug_status_get(uint32_t mac_debug);
功能描述	获取ENET调试状态信息
先决条件	-
被调用函数	-
输入参数{in}	
mac_debug	ENET调试信息选项，下列参数仅可选择一个
ENET_MAC_RECEIVER_NOT_IDLE	MAC接收器非空
ENET_RX_ASYNCERONOUS_FIFO_STATUS	异步接收FIFO状态
ENET_RXFIFO_WRITING	接收FIFO正在执行写操作
ENET_RXFIFO_READ_STATUS	读取接收FIFO操作状态

ENET_RXFIFO_STATE	接收FIFO状态
ENET_MAC_TRANSMITTER_NOT_IDLE	MAC发送器非空
ENET_MAC_TRANSMITTER_STATUS	MAC发送器状态
ENET_PAUSE_CONDITION_STATUS	暂停条件状态
ENET_TXFIFO_READ_STATUS	读取发送FIFO操作状态
ENET_TXFIFO_WRITING	发送FIFO正在执行写操作
ENET_TXFIFO_NOT_EMPTY	发送FIFO非空
ENET_TXFIFO_FULL	发送FIFO已满
输出参数{out}	
-	-
返回值	
uint32_t	相关状态值

例如：

```
/* get debug message of RxFIFO state */
```

```
uint32_t debug_value;
```

```
debug_value = enet_debug_status_get (ENET_RXFIFO_STATE);
```

### 函数 enet\_address\_filter\_enable

函数enet\_address\_filter\_enable描述见下表：

表 3-293. 函数 enet\_address\_filter\_enable

函数名称	enet_address_filter_enable
函数原型	void enet_address_filter_enable(enet_macaddress_enum mac_addr);
功能描述	MAC地址过滤器使能
先决条件	-
被调用函数	--
输入参数{in}	
mac_addr	选择被使能的MAC地址组，参考 <a href="#">表3-264. 枚举类型 enet_macaddress_enum</a> ，下列参数仅可选择一个
ENET_MAC_ADDRESS1	使能MAC地址组1过滤器
ENET_MAC_ADDRESS2	使能MAC地址组2过滤器

ENET_MAC_ADDRE SS3	使能MAC地址组3过滤器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the MAC address 1 filter */
```

```
enet_address_filter_enable(ENET_MAC_ADDRESS1);
```

### 函数 enet\_address\_filter\_disable

函数enet\_address\_filter\_disable描述见下表：

表 3-294. 函数 enet\_address\_filter\_disable

函数名称	enet_address_filter_disable
函数原型	void enet_address_filter_disable(enet_macaddress_enum mac_addr);
功能描述	MAC地址过滤器禁能
先决条件	-
被调用函数	-
输入参数{in}	
mac_addr	选择被使能的MAC地址组，参考 <a href="#">表3-264. 枚举类型 enet_macaddress_enum</a> ，下列参数仅可选择一个
ENET_MAC_ADDRE SS1	使能MAC地址组1过滤器
ENET_MAC_ADDRE SS2	使能MAC地址组2过滤器
ENET_MAC_ADDRE SS3	使能MAC地址组3过滤器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the MAC address 1 filter */
```

```
enet_address_filter_disable(ENET_MAC_ADDRESS1);
```

### 函数 enet\_address\_filter\_config

函数enet\_address\_filter\_config描述见下表：

表 3-295. 函数 `enet_address_filter_config`

函数名称	<code>enet_address_filter_config</code>
函数原型	<code>void enet_address_filter_config(enet_macaddress_enum mac_addr, uint32_t addr_mask, uint32_t filter_type);</code>
功能描述	配置MAC地址过滤器模式
先决条件	-
被调用函数	-
输入参数{in}	
<b>mac_addr</b>	选择被使能的MAC地址组，参考 <a href="#">表3-264. 枚举类型 <code>enet_macaddress_enum</code></a> ，下列参数仅可选择一个
<code>ENET_MAC_ADDRESSES1</code>	使能MAC地址组1过滤器
<code>ENET_MAC_ADDRESSES2</code>	使能MAC地址组2过滤器
<code>ENET_MAC_ADDRESSES3</code>	使能MAC地址组3过滤器
输入参数{in}	
<b>addr_mask</b>	选择MAC地址哪些字节将被屏蔽，下列参数可以选择多个
<code>ENET_ADDRESS_MASK_BYTE0</code>	屏蔽ENET_MAC_ADDR1L[7:0] bits
<code>ENET_ADDRESS_MASK_BYTE1</code>	屏蔽ENET_MAC_ADDR1L[15:8] bits
<code>ENET_ADDRESS_MASK_BYTE2</code>	屏蔽ENET_MAC_ADDR1L[23:16] bits
<code>ENET_ADDRESS_MASK_BYTE3</code>	屏蔽ENET_MAC_ADDR1L [31:24] bits
<code>ENET_ADDRESS_MASK_BYTE4</code>	屏蔽ENET_MAC_ADDR1H [7:0] bits
<code>ENET_ADDRESS_MASK_BYTE5</code>	屏蔽ENET_MAC_ADDR1H [15:8] bits
输入参数{in}	
<b>filter_type</b>	选择MAC地址过滤器类型， 下列参数仅可选择一个
<code>ENET_ADDRESS_FILTER_SA</code>	过滤器比对接收帧MAC地址的源地址域
<code>ENET_ADDRESS_FILTER_DA</code>	过滤器比对接收帧MAC地址的目的地址域
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config the MAC address 1 filter */
```

```
enet_address_filter_config(ENET_MAC_ADDRESS1, ENET_ADDRESS_MASK_BYTE0 |
ENET_ADDRESS_MASK_BYTE1 | ENET_ADDRESS_MASK_BYTE2, ENET_ADDRESS
_FILTER_DA);
```

### 函数 enet\_phy\_config

函数enet\_phy\_config描述见下表：

表 3-296. 函数 enet\_phy\_config

函数名称	enet_phy_config
函数原型	ErrStatus enet_phy_config(void);
功能描述	PHY接口配置（配置SMI时钟并复位PHY芯片）
先决条件	-
被调用函数	rcu_clock_freq_get()/enet_phy_write_read()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如：

```
/* config PHY interface */
```

```
enet_phy_config();
```

### 函数 enet\_phy\_write\_read

函数enet\_phy\_write\_read描述见下表：

表 3-297. 函数 enet\_phy\_write\_read

函数名称	enet_phy_write_read
函数原型	ErrStatus enet_phy_write_read(enet_phydirection_enum direction, uint16_t phy_address, uint16_t phy_reg, uint16_t *pvalue);
功能描述	写/读PHY寄存器
先决条件	-
被调用函数	-
输入参数{in}	
direction	参考 <a href="#">表3-262. 枚举类型enet_phydirection_enum</a> ，下列参数仅可选择一个
ENET_PHY_WRITE	向PHY寄存器写数据
ENET_PHY_READ	从PHY寄存器读数据
输入参数{in}	
phy_address	0x0 - 0x1F
输入参数{in}	

phy_reg	0x0 - 0x1F
输入参数{in}	
pvalue	当direction选择ENET_PHY_WRITE时，表示将写入PHY寄存器的值
输出参数{out}	
pvalue-	当direction选择ENET_PHY_READ时，表示将存储PHY寄存器读出的值
返回值	
ErrStatus	ERROR或SUCCESS

例如：

```
/* write 0 to PHY BCR register */
```

```
uint16_t temp_phy = 0U;
```

```
phy_state = enet_phy_write_read(ENET_PHY_WRITE, PHY_ADDRESS, PHY_REG_BCR,
&temp_phy);
```

### 函数 enet\_phyloopback\_enable

函数enet\_phyloopback\_enable描述见下表：

表 3-298. 函数 enet\_phyloopback\_enable

函数名称	enet_phyloopback_enable
函数原型	ErrStatus enet_phyloopback_enable(void);
功能描述	使能PHY芯片回环模式
先决条件	-
被调用函数	enet_phy_write_read()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如：

```
/* enable the loopback function of PHY chip */
```

```
ErrStatus phy_state = ERROR;
```

```
phy_state = enet_phyloopback_enable();
```

### 函数 enet\_phyloopback\_disable

函数enet\_phyloopback\_disable描述见下表：

表 3-299. 函数 enet\_phyloopback\_disable

函数名称	enet_phyloopback_disable
函数原型	ErrStatus enet_phyloopback_disable(void);

功能描述	禁能PHY芯片回环模式
先决条件	-
被调用函数	enet_phy_write_read
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如：

```
/* disable the loopback function of PHY chip */
```

```
ErrStatus phy_state = ERROR;
```

```
phy_state = enet_phyloopback_disable();
```

### 函数 enet\_forward\_feature\_enable

函数enet\_forward\_feature\_enable描述见下表：

表 3-300. 函数 enet\_forward\_feature\_enable

函数名称	enet_forward_feature_enable
函数原型	void enet_forward_feature_enable(uint32_t feature);
功能描述	使能ENET帧通过相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET帧通过功能，下列参数可以选择多个
ENET_AUTO_PAD RC_DROP	自动去除接收帧填充字节和FCS域
ENET_TYPEFRAME _CRC_DROP	帧通过前自动去除FCS域最后四个字节的CRC校验和
ENET_FORWARD_ ERRFRAMES	除了过短帧外的其他错误帧都会转发给应用
ENET_FORWARD_ UNDERSZ_GOODF RAMES	帧长小于64字节但没有错误的帧将转发给应用
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the function that forwarding undersized good frames */
```

```
enet_forward_feature_enable(ENET_FORWARD_UNDEFSZ_GOODFRAMES);
```

### 函数 `enet_forward_feature_disable`

函数 `enet_forward_feature_disable` 描述见下表：

**表 3-301. 函数 `enet_forward_feature_disable`**

函数名称	<code>enet_forward_feature_disable</code>
函数原型	<code>void enet_forward_feature_disable(uint32_t feature);</code>
功能描述	禁能ENET帧通过相关功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>feature</b>	ENET帧通过功能，下列参数可以选择多个
<code>ENET_AUTO_PAD</code> <code>RC_DROP</code>	自动去除接收帧填充字节和FCS域
<code>ENET_TYPEFRAME</code> <code>_CRC_DROP</code>	帧通过前自动去除FCS域最后四个字节的CRC校验和
<code>ENET_FORWARD</code> <code>ERRFRAMES</code>	除了过短帧外的其他错误帧都会转发给应用
<code>ENET_FORWARD</code> <code>UNDEFSZ_GOODFRAMES</code>	帧长小于64字节但没有错误的帧将转发给应用
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the function that forwarding undersized good frames */
```

```
enet_forward_feature_enable(ENET_FORWARD_UNDEFSZ_GOODFRAMES);
```

### 函数 `enet_fliter_feature_enable`

函数 `enet_fliter_feature_enable` 描述见下表：

**表 3-302. 函数 `enet_fliter_feature_enable`**

函数名称	<code>enet_fliter_feature_enable</code>
函数原型	<code>void enet_fliter_feature_enable(uint32_t feature)</code>
功能描述	使能ENET帧过滤器相关功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>feature</b>	ENET过滤器功能，下列参数可以选择多个

ENET_SRC_FILTER	源地址过滤器
ENET_SRC_FILTER_INVERSE	源地址过滤结果逆转
ENET_DEST_FILTER_INVERSE	目的地址过滤结果逆转
ENET_MULTICAST_FILTER_PASS	接收多播帧
ENET_MULTICAST_FILTER_HASH_MODE	HASH多播过滤器
ENET_UNICAST_FILTER_HASH_MODE	HASH单播过滤器
ENET_FILTER_MODE_EITHER	HASH或完美过滤器功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable filter source address function */
```

```
enet_fliter_feature_enable(ENET_SRC_FILTER);
```

### 函数 enet\_fliter\_feature\_disable

函数enet\_fliter\_feature\_disable描述见下表：

表 3-303. 函数 enet\_fliter\_feature\_disable

函数名称	enet_fliter_feature_disable
函数原型	void enet_fliter_feature_disable(uint32_t feature);
功能描述	禁能ENET帧过滤器相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET过滤器功能，下列参数可以选择多个
ENET_SRC_FILTER	源地址过滤器
ENET_SRC_FILTER_INVERSE	源地址过滤结果逆转
ENET_DEST_FILTER_INVERSE	目的地址过滤结果逆转
ENET_MULTICAST_FILTER_PASS	接收多播帧
ENET_MULTICAST_FILTER_HASH_MODE	HASH多播过滤器

<i>FILTER_HASH_MODE</i>	
<i>ENET_UNICAST_FILTER_HASH_MODE</i>	HASH单播过滤器
<i>ENET_FILTER_MODE_EITHER</i>	HASH或完美过滤器功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable filter source address function */
```

```
enet_fltir_feature_enable(ENET_SRC_FILTER);
```

### 函数 **enet\_pauseframe\_generate**

函数enet\_pauseframe\_generate描述见下表：

**表 3-304. 函数 enet\_pauseframe\_generate**

函数名称	enet_pauseframe_generate
函数原型	ErrStatus enet_pauseframe_generate(void);
功能描述	生成暂停帧，使能发送流控功能后ENET模块将发送暂停帧
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如：

```
/* generate the pause frame */
```

```
ErrStatus reval;
```

```
reval = enet_pauseframe_generate();
```

### 函数 **enet\_pauseframe\_detect\_config**

函数enet\_pauseframe\_detect\_config描述见下表：

**表 3-305. 函数 enet\_pauseframe\_detect\_config**

函数名称	enet_pauseframe_detect_config
------	-------------------------------

函数原型	void enet_pauseframe_detect_config(uint32_t detect);
功能描述	配置暂停帧检测类型
先决条件	-
被调用函数	-
输入参数{in}	
detect	暂停帧检测，下列参数仅可选择一个
ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDDETECT	除了唯一多播地址的暂停帧，MAC同时还会使用MAC0地址（ENET_MAC_ADDR0H寄存器和ENET_MAC_ADDR0L寄存器）来检测暂停帧
ENET_UNIQUE_PAUSEDDETECT	MAC只接收符合IEEE802.3规范定义的唯一多播地址的暂停帧
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config the pause frame detect type as ENET_UNIQUE_PAUSEDDETECT */
```

```
enet_pauseframe_detect_config(ENET_UNIQUE_PAUSEDDETECT);
```

### 函数 enet\_pauseframe\_config

函数enet\_pauseframe\_config描述见下表：

表 3-306. 函数 enet\_pauseframe\_config

函数名称	enet_pauseframe_config
函数原型	void enet_pauseframe_config(uint32_t pausetime, uint32_t pause_threshold);
功能描述	配置暂停帧参数
先决条件	-
被调用函数	-
输入参数{in}	
pausetime	暂停控制帧时间域，范围(0~0xFFFF)
输入参数{in}	
pause_threshold	设置了自动重发暂停帧的定时器阈值。这个阈值应当大于0，小于位[31:16]定义的暂停时间。低阈值的计算公式为PTM-PLTS。例如，PTM = 0x80（128个时间间隙），PLTS = 0x1（28个时间间隙），那么在第一个暂停帧发出100(128-28)个时间间隙后，将自动重发第二个暂停帧，下列参数仅可选择一个
ENET_PAUSETIME_MINUS4	暂停时间 - 4个时间间隙
ENET_PAUSETIME_MINUS28	暂停时间 - 28个时间间隙
ENET_PAUSETIME_MINUS144	暂停时间 - 144个时间间隙

ENET_PAUSETIME _MINUS256	暂停时间 – 256个时间间隙
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config pause time minus 4 slot times */
```

```
enet_pauseframe_config(30, ENET_PAUSETIME_MINUS4);
```

### 函数 enet\_flowcontrol\_threshold\_config

函数enet\_flowcontrol\_threshold\_config描述见下表：

表 3-307. 函数 enet\_flowcontrol\_threshold\_config

函数名称	enet_flowcontrol_threshold_config
函数原型	void enet_flowcontrol_threshold_config(uint32_t deactive, uint32_t active);
功能描述	配置流控阈值
先决条件	-
被调用函数	-
输入参数{in}	
deactive	流控失效的阈值。这个值应当小于位[2:0]定义的流控激活阈值。当RxFIFO中未处理的数据低于这些位所设置的值，流控功能将自动失效，下列参数仅可选择 一个
ENET_DEACTIVE_T HRESHOLD_256BY TES	256字节
ENET_DEACTIVE_T HRESHOLD_512BY TES	512字节
ENET_DEACTIVE_T HRESHOLD_768BY TES	768字节
ENET_DEACTIVE_T HRESHOLD_1024B YTES	1024字节
ENET_DEACTIVE_T HRESHOLD_1280B YTES	1280字节
ENET_DEACTIVE_T HRESHOLD_1536B YTES	1536字节

ENET_DEACTIVE_THRESHOLD_1792BYTES	1792字节
输入参数{in}	
active	流控激活的阈值。若使能了流控功能，当RxFIFO中未处理的数据超过了这些位所设置的值，流控功能将被激活，下列参数仅可选择一个
ENET_ACTIVE_THRESHOLD_256BYTES	256字节
ENET_ACTIVE_THRESHOLD_512BYTES	512字节
ENET_ACTIVE_THRESHOLD_768BYTES	768字节
ENET_ACTIVE_THRESHOLD_1024BYTES	1024字节
ENET_ACTIVE_THRESHOLD_1280BYTES	1280字节
ENET_ACTIVE_THRESHOLD_1536BYTES	1536字节
ENET_ACTIVE_THRESHOLD_1792BYTES	1792字节
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the threshold of the flow control */
```

```
enet_flowcontrol_threshold_config(ENET_DEACTIVE_THRESHOLD_256BYTES, ENET_ACTIVE_THRESHOLD_256BYTES);
```

### 函数 enet\_flowcontrol\_feature\_enable

函数enet\_flowcontrol\_feature\_enable描述见下表：

**表 3-308. 函数 enet\_flowcontrol\_feature\_enable**

函数名称	enet_flowcontrol_feature_enable
------	---------------------------------

函数原型	void enet_flowcontrol_feature_enable(uint32_t feature);
功能描述	使能ENET流控相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET流控功能模式，下列参数可以选择多个
ENET_ZERO_QUANTA_PAUSE	零时间片暂停控制帧自动生成
ENET_TX_FLOWCONTROL	发送流控功能
ENET_RX_FLOWCONTROL	接收流控功能
ENET_BACK_PRESSURE	背压功能（仅在半双工模式下）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the flow control operation in the MAC */
```

```
enet_flowcontrol_feature_enable(ENET_ZERO_QUANTA_PAUSE);
```

### 函数 enet\_flowcontrol\_feature\_disable

函数enet\_flowcontrol\_feature\_disable描述见下表：

表 3-309. 函数 enet\_flowcontrol\_feature\_disable

函数名称	enet_flowcontrol_feature_disable
函数原型	void enet_flowcontrol_feature_disable(uint32_t feature);
功能描述	禁能ENET流控相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET流控功能模式，下列参数可以选择多个
ENET_ZERO_QUANTA_PAUSE	零时间片暂停控制帧自动生成
ENET_TX_FLOWCONTROL	发送流控功能
ENET_RX_FLOWCONTROL	接收流控功能
ENET_BACK_PRESSURE	背压功能（仅在半双工模式下）

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the automatic zero-quanta generation function */
```

```
enet_flowcontrol_feature_disable(ENET_ZERO_QUANTA_PAUSE);
```

### 函数 enet\_dmaprocess\_state\_get

函数enet\_dmaprocess\_state\_get描述见下表：

表 3-310. 函数 enet\_dmaprocess\_state\_get

函数名称	enet_dmaprocess_state_get
函数原型	uint32_t enet_dmaprocess_state_get(enet_dmadirection_enum direction);
功能描述	获取DMA发送/接收流程状态
先决条件	-
被调用函数	-
输入参数{in}	
direction	DMA传输方向
ENET_DMA_TX	DMA发送进程
ENET_DMA_RX	DMA接收进程
输出参数{out}	
-	-
返回值	
uint32_t	DMA流程状态，可取值如下： ENET_RX_STATE_STOPPED / ENET_RX_STATE_FETCHING / ENET_RX_STATE_WAITING / ENET_RX_STATE_SUSPENDED / ENET_RX_STATE_CLOSING / ENET_RX_STATE_QUEUING / ENET_TX_STATE_STOPPED / ENET_TX_STATE_FETCHING / ENET_TX_STATE_WAITING / ENET_TX_STATE_READING / ENET_TX_STATE_SUSPENDED / ENET_TX_STATE_CLOSING

例如：

```
/* get the dma receive process state */
```

```
uint32_t reval;
```

```
reval = enet_dmaprocess_state_get(ENET_DMA_RX);
```

```
if(ENET_RX_STATE_SUSPENDED == reval){
```

```
    do...
```

```
}
```

## 函数 `enet_dmaprocess_resume`

函数 `enet_dmaprocess_resume` 描述见下表：

**表 3-311. 函数 `enet_dmaprocess_resume`**

函数名称	<code>enet_dmaprocess_resume</code>
函数原型	<code>void enet_dmaprocess_resume(enet_dmadirection_enum direction);</code>
功能描述	DMA发送/接收查询使能
先决条件	-
被调用函数	-
输入参数{in}	
<b>direction</b>	描述符类型，下列参数仅可选择一个
<code>ENET_DMA_TX</code>	DMA发送进程
<code>ENET_DMA_RX</code>	DMA接收进程
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA receive process */
enet_dmaprocess_resume(ENET_DMA_RX);
```

## 函数 `enet_rxprocess_check_recovery`

函数 `enet_rxprocess_check_recovery` 描述见下表：

**表 3-312. 函数 `enet_rxprocess_check_recovery`**

函数名称	<code>enet_rxprocess_check_recovery</code>
函数原型	<code>void enet_rxprocess_check_recovery(void);</code>
功能描述	检测并恢复接收流程
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* check and recover the Rx process */
enet_rxprocess_check_recovery();
```

函数 **enet\_txfifo\_flush**

函数enet\_txfifo\_flush描述见下表：

表 3-313. 函数 **enet\_txfifo\_flush**

函数名称	enet_txfifo_flush
函数原型	ErrStatus enet_txfifo_flush(void);
功能描述	刷新ENET发送FIFO，并等待刷新操作完成
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如：

```
/* flush the ENET transmit FIFO */
```

```
ErrStatus reval = ERROR;
```

```
reval = enet_txfifo_flush();
```

函数 **enet\_current\_desc\_address\_get**

函数enet\_current\_desc\_address\_get描述见下表：

表 3-314. 函数 **enet\_current\_desc\_address\_get**

函数名称	enet_current_desc_address_get
函数原型	uint32_t enet_current_desc_address_get(enet_desc_reg_enum addr_get);
功能描述	获取当前发送/接收描述符地址、当前缓冲区地址、描述符列表首地址
先决条件	-
被调用函数	-
输入参数{in}	
addr_get	可获取的描述符地址类型，参考 <a href="#">表3-254. 枚举类型enet_desc_reg_enum</a> ，下列参数仅可选择一个
ENET_RX_DESC_TABLE	接收描述符列表首地址
ENET_RX_CURRENT_DESC	当前DMA控制器使用的接收描述符地址
ENET_RX_CURRENT_BUFFER	当前DMA控制器使用的接收描述符缓冲区地址
ENET_TX_DESC_TABLE	发送描述符列表首地址

ENET_TX_CURRENT_DESC	当前DMA控制器使用的发送描述符地址
ENET_TX_CURRENT_BUFFER	当前DMA控制器使用的发送描述符缓冲区地址
输出参数{out}	
-	-
返回值	
uint32_t	0- 0xFFFFFFFF

例如:

```
/* get the start address of the receive descriptor table */
```

```
uint32_t reval;
```

```
reval = enet_current_desc_address_get(ENET_RX_DESC_TABLE);
```

### 函数 enet\_desc\_information\_get

函数enet\_desc\_information\_get描述见下表:

表 3-315. 函数 enet\_desc\_information\_get

函数名称	enet_desc_information_get
函数原型	uint32_t enet_desc_information_get(enet_descriptors_struct *desc, enet_descstate_enum info_get);
功能描述	获取发送/接收描述符详细信息
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针, 结构体成员介绍请参考 <a href="#">表3-247. 结构体enet_descriptors_struct</a>
输入参数{in}	
info_get	可选择的描述符信息类型, 参考 <a href="#">表3-265. 枚举类型enet_descstate_enum</a> , 下列参数仅可选择一个
TXDESC_COLLISION_COUNT	帧发送出去前出现的冲突次数
TXDESC_BUFFER_1_ADDR	发送帧的缓冲区地址
RXDESC_FRAME_LENGTH	接收帧长度
RXDESC_BUFFER_1_SIZE	接收缓冲区1大小
RXDESC_BUFFER_2_SIZE	接收缓冲区2大小
RXDESC_BUFFER_1_ADDR	接收帧的缓冲区地址
输出参数{out}	

-	-
返回值	
uint32_t	描述符信息，如果返回值为0xFFFFFFFFU，说明输入参数有误

例如：

```
/* get the reception buffer 1 size */
```

```
uint32_t reval;
```

```
reval = enet_desc_information_get(rx_desc, RXDESC_BUFFER_1_SIZE);
```

### 函数 enet\_missed\_frame\_counter\_get

函数enet\_missed\_frame\_counter\_get描述见下表：

表 3-316. 函数 enet\_missed\_frame\_counter\_get

函数名称	enet_missed_frame_counter_get
函数原型	void enet_missed_frame_counter_get(uint32_t *rxfifo_drop, uint32_t *rxdma_drop);
功能描述	获取接收丢弃帧数
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
rxfifo_drop	存储由于过短帧或接收FIFO溢出而丢弃帧数的指针
输出参数{out}	
rxdma_drop	存储由于接收描述符不可用而丢弃帧数的指针
返回值	
-	-

例如：

```
/* get the number of missed frames during receiving */
```

```
uint32_t rxcnt, txcnt;
```

```
enet_missed_frame_counter_get(&rxcnt, &txcnt);
```

### 函数 enet\_desc\_flag\_get

函数enet\_desc\_flag\_get描述见下表：

表 3-317. 函数 enet\_desc\_flag\_get

函数名称	enet_desc_flag_get
函数原型	FlagStatus enet_desc_flag_get(enet_descriptors_struct *desc, uint32_t desc_flag);
功能描述	获取ENET模块DMA描述符标志位

先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 <a href="#">表3-247. 结构体enet_descriptors_struct</a>
输入参数{in}	
desc_flag (the value according to the parameter desc)	描述符标志位，下列参数仅可选择一个
当desc参数为发送描述符时	
ENET_TDES0_DB	顺延位
ENET_TDES0_UFE	数据下溢错误位
ENET_TDES0_EXD	过度顺延位
ENET_TDES0_VFRM	VLAN帧位
ENET_TDES0_ECO	过度冲突位
ENET_TDES0_LCO	延迟冲突位
ENET_TDES0_NCA	无载波位
ENET_TDES0_LCA	载波丢失位
ENET_TDES0_IPPE	IP数据错误位
ENET_TDES0_FRMF	帧清空位
ENET_TDES0_JT	Jabber超时位
ENET_TDES0_ES	错误汇总
ENET_TDES0_IPHE	IP报头错误位
ENET_TDES0_TTMSS	发送时间戳状态位
ENET_TDES0_TCHM	第二地址链表模式位
ENET_TDES0_TERM	环形发送结束模式位
ENET_TDES0_TTSN	使能发送时间戳位
ENET_TDES0_DPAD	不填充位
ENET_TDES0_DCR	不计算CRC位
ENET_TDES0_FSG	第一分块位
ENET_TDES0_LSG	最后分块位
ENET_TDES0_INTC	完成时中断位
ENET_TDES0_DAV	DAV位
当desc参数为接收描述符时	
ENET_RDES0_PCE	数据校验和错误

<i>RR</i>	
<i>ENET_RDES0_CERR</i>	CRC错误
<i>ENET_RDES0_DBERR</i>	Dribble位错误
<i>ENET_RDES0_RERR</i>	接收错误
<i>ENET_RDES0_RWD</i> <i>T</i>	接收看门狗超时
<i>ENET_RDES0_FRMT</i>	帧类型
<i>ENET_RDES0_LCO</i>	延迟冲突位
<i>ENET_RDES0_IPHERR</i>	IP帧报头校验和错误
<i>ENET_RDES0_LDES</i>	最后一个描述符
<i>ENET_RDES0_FDES</i>	第一个描述符
<i>ENET_RDES0_VTAG</i>	VLAN标签位
<i>ENET_RDES0_OERR</i>	溢出错误位
<i>ENET_RDES0_LERR</i>	长度错误位
<i>ENET_RDES0_SAF</i> <i>F</i>	未通过源地址过滤器位
<i>ENET_RDES0_DER</i> <i>R</i>	描述符错误位
<i>ENET_RDES0_ERR</i> <i>S</i>	错误汇总位
<i>ENET_RDES0_DAF</i> <i>F</i>	未通过目标地址过滤器位
<i>ENET_RDES0_DAV</i>	描述符可用位
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* get the bit flag of ENET DMA descriptor */
```

```
FlagStatus reval;
```

```
reval = enet_desc_flag_get(p_txdesc, ENET_TDES0_TCHM);
```

## 函数 enet\_desc\_flag\_set

函数enet\_desc\_flag\_set描述见下表：

**表 3-318. 函数 enet\_desc\_flag\_set**

函数名称	enet_desc_flag_set
函数原型	void enet_desc_flag_set(enet_descriptors_struct *desc, uint32_t desc_flag);
功能描述	设置ENET模块DMA描述符标志位
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 <a href="#">表3-247. 结构体enet_descriptors_struct</a>
输入参数{in}	
desc_flag (the value according to the parameter desc)	描述符标志位，下列参数仅可选择一个
当desc参数为发送描述符时	
ENET_TDES0_VFR M	VLAN帧位
ENET_TDES0_FRM F	帧清空位
ENET_TDES0_TCH M	第二地址链表模式位
ENET_TDES0_TER M	环形发送结束模式位
ENET_TDES0_TTS EN	使能发送时间戳位
ENET_TDES0_DPA D	不填充位
ENET_TDES0_DCR C	不计算CRC位
ENET_TDES0_FSG	第一分块位
ENET_TDES0_LSG	最后分块位
ENET_TDES0_INTC	完成时中断位
ENET_TDES0_DAV	DAV位
当desc参数为接收描述符时	
ENET_RDES0_DAV	描述符可用位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set VLAN frame bit flag of ENET DMA descriptor */
```

```
enet_desc_flag_set(p_txdesc, ENET_TDES0_VFRM);
```

### 函数 enet\_desc\_flag\_clear

函数enet\_desc\_flag\_clear描述见下表：

表 3-319. 函数 enet\_desc\_flag\_clear

函数名称	enet_desc_flag_clear
函数原型	void enet_desc_flag_clear(enet_descriptors_struct *desc, uint32_t desc_flag);
功能描述	清除ENET模块DMA描述符标志位
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 <a href="#">表3-247. 结构体enet_descriptors_struct</a>
输入参数{in}	
desc_flag (the value according to the parameter desc)	描述符标志位，下列参数仅可选择一个
当desc参数为发送描述符时	
ENET_TDES0_VFRM	VLAN帧位
ENET_TDES0_FRM	帧清空位
ENET_TDES0_TCH	第二地址链表模式位
ENET_TDES0_TER	环形发送结束模式位
ENET_TDES0_TTS	使能发送时间戳位
ENET_TDES0_DPA	不填充位
ENET_TDES0_DCR	不计算CRC位
ENET_TDES0_FSG	第一分块位
ENET_TDES0_LSG	最后分块位
ENET_TDES0_INTC	完成时中断位
ENET_TDES0_DAV	DAV位
当desc参数为接收描述符时	
ENET_RDES0_DAV	描述符可用位
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* clear VLAN frame bit flag of ENET DMA descriptor */
```

```
enet_desc_flag_clear(p_txdesc, ENET_TDES0_VFRM);
```

### 函数 `enet_rx_desc_immediate_receive_complete_interrupt`

函数 `enet_rx_desc_immediate_receive_complete_interrupt` 描述见下表：

**表 3-320. 函数 `enet_rx_desc_immediate_receive_complete_interrupt`**

函数名称	<code>enet_rx_desc_immediate_receive_complete_interrupt</code>
函数原型	void <code>enet_rx_desc_immediate_receive_complete_interrupt(enet_descriptors_struct *desc);</code>
功能描述	当接收完成时，立即置位 <code>ENET_DMA_STAT</code> 寄存器的 RS 位
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 <a href="#">表3-247. 结构体 <code>enet_descriptors_struct</code></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set RS bit in ENET_DMA_STAT register immediately when receiving completed */
```

```
enet_rx_desc_immediate_receive_complete_interrupt(p_rxdesc);
```

### 函数 `enet_rx_desc_delay_receive_complete_interrupt`

函数 `enet_rx_desc_delay_receive_complete_interrupt` 描述见下表：

**表 3-321. 函数 `enet_rx_desc_delay_receive_complete_interrupt`**

函数名称	<code>enet_rx_desc_delay_receive_complete_interrupt</code>
函数原型	void <code>enet_rx_desc_delay_receive_complete_interrupt(enet_descriptors_struct *desc, uint32_t delay_time);</code>
功能描述	当接收完成时，延迟指定时间再置位 <code>ENET_DMA_STAT</code> 寄存器的 RS 位
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 <a href="#">表3-247. 结构体 <code>enet_descriptors_struct</code></a>
输入参数{in}	

<b>delay_time</b>	延迟时间，实际延迟时间为256*delay_time个HCLK（0~0xFF）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* when receiving completed, RS bit in ENET_DMA_STAT register will be set after 256*16
HCLK */
```

```
enet_rx_desc_delay_receive_complete_interrupt(p_rxdesc, 0x00000010);
```

### 函数 enet\_rxframe\_drop

函数enet\_rxframe\_drop描述见下表：

表 3-322. 函数 enet\_rxframe\_drop

<b>函数名称</b>	enet_rxframe_drop
<b>函数原型</b>	void enet_rxframe_drop(void);
<b>功能描述</b>	丢弃当前接收到的帧
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* drop current receive frame */
```

```
enet_rxframe_drop( );
```

### 函数 enet\_dma\_feature\_enable

函数enet\_dma\_feature\_enable描述见下表：

表 3-323. 函数 enet\_dma\_feature\_enable

<b>函数名称</b>	enet_dma_feature_enable
<b>函数原型</b>	void enet_dma_feature_enable(uint32_t feature);
<b>功能描述</b>	使能ENET模块DMA相关功能
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>feature</b>	DMA功能，下列参数可以选择多个

ENET_NO_FLUSH_RXFRAME	描述符不可用时，RxDMA控制器清空接收帧功能
ENET_SECONDFRAME_OPT	TxDMA控制器第二帧功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable RxDMA does not flushes frames function */
```

```
enet_dma_feature_enable(ENET_NO_FLUSH_RXFRAME);
```

### 函数 enet\_dma\_feature\_disable

函数enet\_dma\_feature\_disable描述见下表：

表 3-324. 函数 enet\_dma\_feature\_disable

函数名称	enet_dma_feature_disable
函数原型	void enet_dma_feature_disable(uint32_t feature);
功能描述	禁能ENET模块DMA相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	DMA功能，下列参数可以选择多个
ENET_NO_FLUSH_RXFRAME	描述符不可用时，RxDMA控制器清空接收帧功能
ENET_SECONDFRAME_OPT	TxDMA控制器第二帧功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable RxDMA does not flushes frames function */
```

```
enet_dma_feature_disable(ENET_NO_FLUSH_RXFRAME);
```

### 函数 enet\_rx\_desc\_enhanced\_status\_get

函数enet\_rx\_desc\_enhanced\_status\_get描述见下表：

表 3-325. 函数 enet\_rx\_desc\_enhanced\_status\_get

函数名称	enet_rx_desc_enhanced_status_get
------	----------------------------------

函数原型	uint32_t enet_rx_desc_enhanced_status_get(enet_descriptors_struct *desc, uint32_t desc_status);
功能描述	获取接收描述符增强状态标志位信息
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 <a href="#">表3-247. 结构体enet_descriptors_struct</a>
输入参数{in}	
desc_status	想要获取的状态信息
ENET_RDES4_IPPLDT	IP帧有效载荷
ENET_RDES4_IPHERR	IP帧头错误
ENET_RDES4_IPPLDERR	IP帧有效载荷错误
ENET_RDES4_IPCKSB	IP帧旁路校验和
ENET_RDES4_IPF4	Ipv4帧
ENET_RDES4_IPF6	Ipv6帧
ENET_RDES4_PTPMT	PTP消息类型
ENET_RDES4_PTPOEF	PTP网络帧
ENET_RDES4_PTPVF	PTP版本格式
输出参数{out}	
-	-
返回值	
uint32_t	相应的状态值

例如：

```
/* get the IP frame payload type in ENET DMA descriptor */
```

```
uint32_t status;
```

```
status = enet_rx_desc_enhanced_status_get(p_rxdesc, ENET_RDES4_IPPLDT);
```

### 函数 enet\_desc\_select\_enhanced\_mode

函数enet\_desc\_select\_enhanced\_mode描述见下表：

表 3-326. 函数 enet\_desc\_select\_enhanced\_mode

函数名称	enet_desc_select_enhanced_mode
函数原型	void enet_desc_select_enhanced_mode(void);
功能描述	配置DMA描述符为增强型描述符

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

例如：

```
/* configure descriptor to work in enhanced mode */
```

```
enet_desc_select_enhanced_mode();
```

### 函数 `enet_ptp_enhanced_descriptors_chain_init`

函数 `enet_ptp_enhanced_descriptors_chain_init` 描述见下表：

表 3-327. 函数 `enet_ptp_enhanced_descriptors_chain_init`

函数名称	<code>enet_ptp_enhanced_descriptors_chain_init</code>
函数原型	<code>void enet_ptp_enhanced_descriptors_chain_init(enet_dmadirection_enum direction);</code>
功能描述	初始化具有PTP功能的增强型DMA接收/发送描述符为链模式
先决条件	-
被调用函数	-
输入参数{in}	
<b>direction</b>	描述符类型，下列参数仅可选择一个
<code>ENET_DMA_TX</code>	DMA Tx描述符
<code>ENET_DMA_RX</code>	DMA Rx描述符
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the DMA Tx descriptors's parameters in enhanced chain mode with ptp function */
```

```
enet_ptp_enhanced_descriptors_chain_init(ENET_DMA_TX);
```

### 函数 `enet_ptp_enhanced_descriptors_ring_init`

函数 `enet_ptp_enhanced_descriptors_ring_init` 描述见下表：

表 3-328. 函数 `enet_ptp_enhanced_descriptors_ring_init`

函数名称	<code>enet_ptp_enhanced_descriptors_ring_init</code>
函数原型	<code>void enet_ptp_enhanced_descriptors_ring_init(enet_dmadirection_enum</code>

	direction);
功能描述	初始化具有PTP功能的DMA接收/发送描述符为环模式
先决条件	-
被调用函数	-
输入参数{in}	
direction	描述符类型，下列参数仅可选择一个
ENET_DMA_TX	DMA Tx描述符
ENET_DMA_RX	DMA Rx描述符
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the DMA Rx descriptors's parameters in enhanced ring mode with ptp function */
enet_ptp_enhanced_descriptors_ring_init(ENET_DMA_RX);
```

### 函数 enet\_ptpframe\_receive\_enhanced\_mode

函数enet\_ptpframe\_receive\_enhanced\_mode描述见下表：

表 3-329. 函数 enet\_ptpframe\_receive\_enhanced\_mode

函数名称	enet_ptpframe_receive_enhanced_mode
函数原型	ErrStatus enet_ptpframe_receive_enhanced_mode(uint8_t *buffer, uint32_t bufsize, uint32_t timestamp[]);
功能描述	在PTP模式下处理当前接收到的帧，并将当前增强型描述符中存储的接收帧数据和时间戳拷贝到指定区域
先决条件	-
被调用函数	-
输入参数{in}	
bufsize	缓冲区大小
输出参数{out}	
buffer	存放数据的用户缓冲区指针，如果输入NULL，用户需要在调用该函数之前将数据拷贝到自己指定的位置
输出参数{out}	
timestamp	存放时间戳指针
返回值	
ErrStatus	SUCCESS or ERROR

例如：

```
/* receive a packet data with timestamp values to application buffer in DMA enhanced mode */
uint32_t rx_buffer[500];
```

```
uint32_t time_stamp[2];
```

```
ErrStatus status;
```

```
status = enet_ptpframe_receive_enhanced_mode (rx_buffer, 500, time_stamp);
```

## 函数 enet\_ptpframe\_transmit\_enhanced\_mode

函数enet\_ptpframe\_transmit\_enhanced\_mode描述见下表：

**表 3-330. 函数 enet\_ptpframe\_transmit\_enhanced\_mode**

函数名称	enet_ptpframe_transmit_enhanced_mode
函数原型	ErrStatus enet_ptpframe_transmit_enhanced_mode(uint8_t *buffer, uint32_t length, uint32_t timestamp[]);
功能描述	在PTP模式下将制定区域内的数据拷贝到当前增强型发送描述符中，并同时间戳一起发送
先决条件	-
被调用函数	-
输入参数{in}	
buffer	存放数据的用户缓冲区指针，如果输入NULL，用户需要在调用该函数之前将数据拷贝到指定的位置
输入参数{in}	
length	发送数据大小
输出参数{out}	
timestamp	存放时间戳的指针，如果输入为NULL，则忽略时间戳
返回值	
ErrStatus	SUCCESS or ERROR

例如：

```
/* send data and timestamp values in application buffer as a transmit packet with DMA enhanced mode */
```

```
uint32_t tx_buffer[500];
```

```
uint32_t time_stamp[2];
```

```
ErrStatus status;
```

```
status = enet_ptpframe_transmit_enhanced_mode(tx_buffer, 500, time_stamp);
```

## 函数 enet\_desc\_select\_normal\_mode

函数enet\_desc\_select\_normal\_mode描述见下表：

**表 3-331. 函数 enet\_desc\_select\_normal\_mode**

函数名称	enet_desc_select_normal_mode
函数原型	void enet_desc_select_normal_mode(void);
功能描述	配置DMA描述符为常规型描述符

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

例如：

```
/* configure descriptor to work in normal mode */
```

```
enet_desc_select_normal_mode( );
```

### 函数 enet\_ptp\_normal\_descriptors\_chain\_init

函数enet\_ptp\_normal\_descriptors\_chain\_init描述见下表：

表 3-332. 函数 enet\_ptp\_normal\_descriptors\_chain\_init

函数名称	enet_ptp_normal_descriptors_chain_init
函数原型	void enet_ptp_normal_descriptors_chain_init(enet_dmadirection_enum direction, enet_descriptors_struct *desc_ptptab);
功能描述	初始化具有PTP功能的DMA接收/发送描述符为链模式
先决条件	-
被调用函数	-
输入参数{in}	
direction	描述符类型，下列参数仅可选择一个
ENET_DMA_TX	DMA Tx描述符
ENET_DMA_RX	DMA Rx描述符
输入参数{in}	
desc_ptptab	描述符指针，结构体成员介绍请参考 <a href="#">表3-247. 结构体enet_descriptors_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the DMA Rx descriptors's parameters in normal chain mode with PTP function */
```

```
enet_descriptors_struct ptp_rxdesc_tab[ENET_RXBUF_NUM];
```

```
enet_ptp_normal_descriptors_chain_init(ENET_DMA_RX, ptp_rxdesc_tab);
```

### 函数 enet\_ptp\_normal\_descriptors\_ring\_init

函数enet\_ptp\_normal\_descriptors\_ring\_init描述见下表：

表 3-333. 函数 `enet_ptp_normal_descriptors_ring_init`

函数名称	<code>enet_ptp_normal_descriptors_ring_init</code>
函数原型	<code>void enet_ptp_normal_descriptors_ring_init(enet_dmadirection_enum direction, enet_descriptors_struct *desc_ptptab);</code>
功能描述	初始化具有PTP功能的DMA接收/发送描述符为环模式
先决条件	-
被调用函数	-
输入参数{in}	
<b>direction</b>	描述符类型，下列参数仅可选择一个
<code>ENET_DMA_TX</code>	DMA Tx描述符
<code>ENET_DMA_RX</code>	DMA Rx描述符
输入参数{in}	
<b>desc_ptptab</b>	描述符指针，结构体成员介绍请参考 <a href="#">表3-247. 结构体enet_descriptors_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the DMA Rx descriptors's parameters in normal ring mode with PTP function */
enet_descriptors_struct ptp_rxdesc_tab[ENET_RXBUF_NUM];

enet_ptp_normal_descriptors_ring_init(ENET_DMA_RX, ptp_rxdesc_tab);
```

### 函数 `enet_ptpframe_receive_normal_mode`

函数`enet_ptpframe_receive_normal_mode`描述见下表：

表 3-334. 函数 `enet_ptpframe_receive_normal_mode`

函数名称	<code>enet_ptpframe_receive_normal_mode</code>
函数原型	<code>ErrStatus enet_ptpframe_receive_normal_mode(uint8_t *buffer, uint32_t bufsize, uint32_t timestamp[]);</code>
功能描述	在PTP模式下处理当前接收到的帧，并将当前描述符中存储的接收帧数据和时 间戳拷贝到指定区域
先决条件	-
被调用函数	-
输入参数{in}	
<b>bufsize</b>	缓冲区大小
输出参数{out}	
<b>timestamp</b>	存放时间戳指针
输出参数{out}	
<b>buffer</b>	存放数据的用户缓冲区指针，如果输入NULL，用户需要在调用该函数之前将数 据拷贝到自己指定的位置
返回值	

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

例如:

```
/* receive a packet data with timestamp values to application buffer in DMA normal mode */
uint32_t rx_buffer[500];

uint32_t time_stamp[2];

ErrStatus status;

status = enet_ptpframe_receive_normal_mode (rx_buffer, 500, time_stamp);
```

### 函数 enet\_ptpframe\_transmit\_normal\_mode

函数enet\_ptpframe\_transmit\_normal\_mode描述见下表:

**表 3-335. 函数 enet\_ptpframe\_transmit\_normal\_mode**

函数名称	enet_ptpframe_transmit_normal_mode
函数原型	ErrStatus enet_ptpframe_transmit_normal_mode(uint8_t *buffer, uint32_t length, uint32_t timestamp[]);
功能描述	在PTP模式下将指定区域内的数据拷贝到当前发送描述符中，并同时时间戳一起发送
先决条件	-
被调用函数	--
输入参数{in}	
buffer	存放数据的用户缓冲区指针，如果输入NULL，用户需要在调用该函数之前将数据拷贝到指定的位置
输入参数{in}	
length	发送数据大小
输出参数{out}	
timestamp	存放时间戳的指针，如果输入为NULL，则忽略时间戳
返回值	
ErrStatus	ERROR或SUCCESS

例如:

```
/* send data and timestamp values in application buffer as a transmit packet with DMA normal mode */
uint32_t tx_buffer[500];

uint32_t time_stamp[2];

ErrStatus status;

status = enet_ptpframe_transmit_normal_mode(tx_buffer, 500, time_stamp);
```

**函数 enet\_wum\_filter\_register\_pointer\_reset**

函数enet\_wum\_filter\_register\_pointer\_reset描述见下表：

**表 3-336. 函数 enet\_wum\_filter\_register\_pointer\_reset**

函数名称	enet_wum_filter_register_pointer_reset
函数原型	void enet_wum_filter_register_pointer_reset(void);
功能描述	远程唤醒帧过滤器寄存器指针复位
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 wakeup frame filter register pointer */
```

```
enet_wum_filter_register_pointer_reset ();
```

**函数 enet\_wum\_filter\_config**

函数enet\_wum\_filter\_config描述见下表：

**表 3-337. 函数 enet\_wum\_filter\_config**

函数名称	enet_wum_filter_config
函数原型	void enet_wum_filter_config(uint32_t pdata[]);
功能描述	配置远程唤醒帧寄存器
先决条件	-
被调用函数	-
输入参数{in}	
<b>pdata</b>	存放将写入远程唤醒帧寄存器组的数据指针（总共8字节）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the remote wakeup frame registers */
```

```
uint32_t wum_data[8];
```

```
enet_wum_filter_config(wum_data);
```

**函数 enet\_wum\_feature\_enable**

函数enet\_wum\_feature\_enable描述见下表:

**表 3-338. 函数 enet\_wum\_feature\_enable**

函数名称	enet_wum_feature_enable
函数原型	void enet_wum_feature_enable(uint32_t feature);
功能描述	使能ENET模块唤醒管理相关功能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>feature</b>	下列参数可以选择多个
ENET_WUM_POWER_DOWN	掉电模式
ENET_WUM_MAGIC_PACKET_FRAME	使能接收到魔法帧的唤醒事件
ENET_WUM_WAKE_UP_FRAME	使能接收到唤醒帧的唤醒事件
ENET_WUM_GLOBAL_UNICAST	任何通过过滤器的单播帧均作为唤醒帧
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* enable power down mode */
```

```
enet_wum_feature_enable(ENET_WUM_POWER_DOWN);
```

**函数 enet\_wum\_feature\_disable**

函数enet\_wum\_feature\_disable描述见下表:

**表 3-339. 函数 enet\_wum\_feature\_disable**

函数名称	enet_wum_feature_disable
函数原型	void enet_wum_feature_disable(uint32_t feature)
功能描述	禁能ENET模块唤醒管理相关功能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>feature</b>	下列参数可以选择多个
ENET_WUM_MAGIC_PACKET_FRAME	使能接收到魔法帧的唤醒事件
ENET_WUM_WAKE	使能接收到唤醒帧的唤醒事件

<code>_UP_FRAME</code>	
<code>ENET_WUM_GLOB</code> <code>AL_UNICAST</code>	任何通过过滤器的单播帧均作为唤醒帧
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable power down mode */
```

```
enet_wum_feature_disable(ENET_WUM_POWER_DOWN);
```

### 函数 `enet_msc_counters_reset`

函数 `enet_msc_counters_reset` 描述见下表：

表 3-340. 函数 `enet_msc_counters_reset`

函数名称	<code>enet_msc_counters_reset</code>
函数原型	<code>void enet_msc_counters_reset(void)</code>
功能描述	复位MAC统计计数器组
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 MAC statistics counters */
```

```
enet_msc_counters_reset();
```

### 函数 `enet_msc_feature_enable`

函数 `enet_msc_feature_enable` 描述见下表：

表 3-341. 函数 `enet_msc_feature_enable`

函数名称	<code>enet_msc_feature_enable</code>
函数原型	<code>void enet_msc_feature_enable(uint32_t feature);</code>
功能描述	使能MAC统计计数器相关功能
先决条件	-
被调用函数	-
输入参数{in}	

feature	下列参数可以选择多个
ENET_MSC_COUNTER_STOP_ROLLOVER	计数器停止回转
ENET_MSC_RESET_ON_READ	读时复位
ENET_MSC_COUNTERS_FREEZE	MSC计数器冻结位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable counter stop rollover function */
```

```
enet_msc_feature_enable(ENET_MSC_COUNTER_STOP_ROLLOVER);
```

### 函数 enet\_msc\_feature\_disable

函数enet\_msc\_feature\_disable描述见下表：

表 3-342. 函数 enet\_msc\_feature\_disable

函数名称	enet_msc_feature_disable
函数原型	void enet_msc_feature_disable(uint32_t feature);
功能描述	禁能MAC统计计数器相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	下列参数可以选择多个
ENET_MSC_COUNTER_STOP_ROLLOVER	计数器停止回转
ENET_MSC_RESET_ON_READ	读时复位
ENET_MSC_COUNTERS_FREEZE	MSC计数器冻结位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable counter stop rollover function */
```

```
enet_msc_feature_disable(ENET_MSC_COUNTER_STOP_ROLLOVER);
```

### 函数 `enet_msc_counters_preset_config`

函数 `enet_msc_counters_preset_config` 描述见下表：

表 3-343. 函数 `enet_msc_counters_preset_config`

函数名称	<code>enet_msc_counters_preset_config</code>
函数原型	<code>void enet_msc_counters_preset_config(enet_msc_preset_enum mode);</code>
功能描述	配置MAC统计计数器的预设模式
先决条件	-
被调用函数	-
输入参数{in}	
mode	参考 <a href="#">表3-266. 枚举类型enet_msc_preset_enum</a> ，下列参数可以选择多个
<code>ENET_MSC_PRESET_NONE</code>	关闭MSC计数器预设功能
<code>ENET_MSC_PRESET_HALF</code>	预设为最大值一半
<code>ENET_MSC_PRESET_FULL</code>	预设为近似全值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* preset all MSC counters to almost-half */
```

```
enet_msc_counters_preset_config (ENET_MSC_PRESET_HALF);
```

### 函数 `enet_msc_counters_get`

函数 `enet_msc_counters_get` 描述见下表：

表 3-344. 函数 `enet_msc_counters_get`

函数名称	<code>enet_msc_counters_get</code>
函数原型	<code>uint32_t enet_msc_counters_get(enet_msc_counter_enum counter);</code>
功能描述	获取MAC相关统计计数器值
先决条件	-
被调用函数	-
输入参数{in}	
counter	MSC计数器，参考 <a href="#">表3-255. 枚举类型enet_msc_counter_enum</a> ，下列参数仅可选择一个
<code>ENET_MSC_TX_SC_CNT</code>	MSC 1次冲突后发送“好”帧的计数器

<i>ENET_MSC_TX_MS</i> <i>CCNT</i>	MSC 1次以上冲突后发送“好”帧的计数器
<i>ENET_MSC_TX_TG</i> <i>FCNT</i>	MSC发送“好”帧计数器
<i>ENET_MSC_RX_RF</i> <i>CECNT</i>	MSC CRC错误接收帧计数器
<i>ENET_MSC_RX_RF</i> <i>AECNT</i>	MSC对齐错误接收帧计数器
<i>ENET_MSC_RX_RG</i> <i>UFCNT</i>	MSC“好”单播帧接收帧计数器
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	MSC计数器值

例如：

```
/* get MSC transmitted good frames after a single collision counter value */
```

```
uint32_t reval;
```

```
reval = enet_msc_counters_get(ENET_MSC_TX_SCCNT);
```

### 函数 enet\_ptp\_subsecond\_2\_nanosecond

函数enet\_ptp\_subsecond\_2\_nanosecond描述见下表：

表 3-345. 函数 enet\_ptp\_subsecond\_2\_nanosecond

函数名称	enet_ptp_subsecond_2_nanosecond
函数原型	uint32_t enet_ptp_subsecond_2_nanosecond(uint32_t subsecond);
功能描述	将亚秒值变为纳秒值
先决条件	-
被调用函数	-
输入参数{in}	
<b>subsecond</b>	亚秒值
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	纳秒值

例如：

```
/* change subsecond to nanosecond */
```

```
uint32_t reval;
```

```
reval = enet_ptp_subsecond_2_nanosecond (2);
```

**函数 enet\_ptp\_nanosecond\_2\_subsecond**

函数enet\_ptp\_nanosecond\_2\_subsecond描述见下表：

**表 3-346. 函数 enet\_ptp\_nanosecond\_2\_subsecond**

函数名称	enet_ptp_nanosecond_2_subsecond
函数原型	uint32_t enet_ptp_nanosecond_2_subsecond(uint32_t nanosecond);
功能描述	将纳秒值变为亚秒值
先决条件	-
被调用函数	-
输入参数{in}	
nanosecond	纳秒值
输出参数{out}	
-	-
返回值	
uint32_t	亚秒值

例如：

```
/* change nanosecond to subsecond */
uint32_t reval;

reval = enet_ptp_nanosecond_2_subsecond (2);
```

**函数 enet\_ptp\_feature\_enable**

函数enet\_ptp\_feature\_enable描述见下表：

**表 3-347. 函数 enet\_ptp\_feature\_enable**

函数名称	enet_ptp_feature_enable
函数原型	void enet_ptp_feature_enable(uint32_t feature);
功能描述	使能PTP相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET模块PTP功能，下列参数可以选择多个
ENET_RXTX_TIMES TAMP	发送/接收帧时间戳
ENET_PTP_TIMEST AMP_INT	时间戳中断触发
ENET_ALL_RX_TIM ESTAMP	所有接收帧时间戳快照使能
ENET_NONTYPE_F RAME_SNAPSHOT	接收以太网帧时时间戳使能
ENET_IPV6_FRAME	接收Ipv6帧时时间戳使能

<code>_SNAPSHOT</code>	
<code>ENET_IPV4_FRAME _SNAPSHOT</code>	接收Ipv4帧时时间戳使能
<code>ENET_PTP_FRAME _USE_MACADDRESS_FILTER</code>	PTP帧MAC地址过滤使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable PTP function for all received frames */
```

```
enet_ptp_feature_enable(ENET_ALL_RX_TIMESTAMP);
```

### 函数 `enet_ptp_feature_disable`

函数 `enet_ptp_feature_disable` 描述见下表：

表 3-348. 函数 `enet_ptp_feature_disable`

函数名称	<code>enet_ptp_feature_disable</code>
函数原型	<code>void enet_ptp_feature_disable(uint32_t feature);</code>
功能描述	禁能PTP相关功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>feature</b>	ENET模块PTP功能，下列参数可以选择多个
<code>ENET_RTX_TIMES TAMP</code>	发送/接收帧时间戳
<code>ENET_PTP_TIMEST AMP_INT</code>	时间戳中断触发
<code>ENET_ALL_RX_TIM ESTAMP</code>	所有接收帧时间戳快照使能
<code>ENET_NONTYPE_F RAME_SNAPSHOT</code>	接收以太网帧时时间戳使能
<code>ENET_IPV6_FRAME _SNAPSHOT</code>	接收Ipv6帧时时间戳使能
<code>ENET_IPV4_FRAME _SNAPSHOT</code>	接收Ipv4帧时时间戳使能
<code>ENET_PTP_FRAME _USE_MACADDRESS_FILTER</code>	PTP帧MAC地址过滤使能
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* disable PTP function for all received frames */
```

```
enet_ptp_feature_disable(ENET_ALL_RX_TIMESTAMP);
```

### 函数 enet\_ptp\_timestamp\_function\_config

函数enet\_ptp\_timestamp\_function\_config描述见下表：

**表 3-349. 函数 enet\_ptp\_timestamp\_function\_config**

函数名称	enet_ptp_timestamp_function_config
函数原型	ErrStatus enet_ptp_timestamp_function_config(enet_ptp_function_enum func);
功能描述	配置PTP时间戳相关功能
先决条件	-
被调用函数	-
输入参数{in}	
func	下列参数仅可选择一个
ENET_CKNT_ORDINARY	时间戳时钟节点类型为普通时钟
ENET_CKNT_BOUNDARY	时间戳时钟节点类型为边界时钟
ENET_CKNT_END_TO_END	时间戳时钟节点类型为端对端透明时钟
ENET_CKNT_PEER_TO_PEER	时间戳时钟节点类型为点对点透明时钟
ENET_PTP_ADDEND_UPDATE	加数寄存器更新
ENET_PTP_SYSTIME_UPDATE	时间戳更新
ENET_PTP_SYSTIME_INIT	时间戳初始化
ENET_PTP_FINEMODE	精调模式更新系统时间戳
ENET_PTP_COARSEMODE	粗调模式更新系统时间戳
ENET_SUBSECOND_DIGITAL_ROLLOVER	十进制回转模式
ENET_SUBSECOND_BINARY_ROLLOVER	二进制回转模式

VER	
ENET_SNOOPING_PTP_VERSION_2	监听PTP帧版本2
ENET_SNOOPING_PTP_VERSION_1	监听PTP帧版本1
ENET_EVENT_TYPE_MESSAGES_SNA_PSHOT	只接收事件类型消息使能时间戳
ENET_ALL_TYPE_MESSAGES_SNAPSHOT	接收到除了Announce, Management和Signaling以外的所有其他类型的消息时, 时间戳快照使能
ENET_MASTER_NODE_MESSAGE_SNA_PSHOT	主节点消息时间戳快照使能
ENET_SLAVE_NODE_MESSAGE_SNAPSHOT	从节点消息时间戳快照使能
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS or ERROR

例如:

```
/* config addend register update function */
```

```
ErrStatus reval = ERROR;
```

```
reval = enet_ptp_timestamp_function_config(ENET_PTP_ADDEND_UPDATE);
```

### 函数 enet\_ptp\_subsecond\_increment\_config

函数enet\_ptp\_subsecond\_increment\_config描述见下表:

表 3-350. 函数 enet\_ptp\_subsecond\_increment\_config

函数名称	enet_ptp_subsecond_increment_config
函数原型	void enet_ptp_subsecond_increment_config(uint32_t subsecond);
功能描述	配置PTP系统时间亚秒增加值
先决条件	-
被调用函数	-
输入参数{in}	
subsecond	该值将被加到系统时间的亚秒值, 范围 (0~0xFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure 0x1F as system time subsecond increment value */
```

```
enet_ptp_subsecond_increment_config(0x1F);
```

### 函数 `enet_ptp_timestamp_addend_config`

函数 `enet_ptp_timestamp_addend_config` 描述见下表：

表 3-351. 函数 `enet_ptp_timestamp_addend_config`

函数名称	<code>enet_ptp_timestamp_addend_config</code>
函数原型	<code>void enet_ptp_timestamp_addend_config(uint32_t add);</code>
功能描述	精调模式下PTP时钟频率校准配置
先决条件	-
被调用函数	-
输入参数{in}	
<b>add</b>	通过将该值加到累加器用于时间同步，范围(0~0xFFFF FFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* added 0x1FFF to the accumulator register */
```

```
enet_ptp_timestamp_addend_config(0x1FFF);
```

### 函数 `enet_ptp_timestamp_update_config`

函数 `enet_ptp_timestamp_update_config` 描述见下表：

表 3-352. 函数 `enet_ptp_timestamp_update_config`

函数名称	<code>enet_ptp_timestamp_update_config</code>
函数原型	<code>void enet_ptp_timestamp_update_config(uint32_t sign, uint32_t second, uint32_t subsecond);</code>
功能描述	初始化时用于替换系统时间，在更新时表示在系统时间上加上或减去的秒值
先决条件	-
被调用函数	-
输入参数{in}	
<b>sign</b>	时间戳更新正或负符号位，下列参数仅可选择一个
<code>ENET_PTP_ADD_T O_TIME</code>	更新值加到系统时间
<code>ENET_PTP_SUBST RACT_FROM_TIME</code>	将系统时间减去更新值
输入参数{in}	

<b>second</b>	秒值，范围(0~0xFFFF FFFF)
<b>输入参数{in}</b>	
<b>subsecond</b>	亚秒值，精度为0.46 ns，范围(0~0xFFFF FFFF)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* initialize system time with timestamp update value */
```

```
enet_ptp_timestamp_update_config(ENET_PTP_ADD_TO_TIME, 0, 0);
```

### 函数 enet\_ptp\_expected\_time\_config

函数enet\_ptp\_expected\_time\_config描述见下表：

表 3-353. 函数 enet\_ptp\_expected\_time\_config

<b>函数名称</b>	enet_ptp_expected_time_config
<b>函数原型</b>	void enet_ptp_expected_time_config(uint32_t second, uint32_t nanosecond);
<b>功能描述</b>	配置PTP期望时间
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>second</b>	目标时间秒值，范围(0~0xFFFF FFFF)
<b>输入参数{in}</b>	
<b>nanosecond</b>	目标时间纳秒值，范围(0~0xFFFF FFFF)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the expected target time */
```

```
enet_ptp_expected_time_config(2000, 0);
```

### 函数 enet\_ptp\_system\_time\_get

函数enet\_ptp\_system\_time\_get描述见下表：

表 3-354. 函数 enet\_ptp\_system\_time\_get

<b>函数名称</b>	enet_ptp_system_time_get
<b>函数原型</b>	void enet_ptp_system_time_get(enet_ptp_systime_struct *systime_struct);
<b>功能描述</b>	获取PTP当前系统时间
<b>先决条件</b>	-

被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
sysptime_struct	PTP系统时间结构体指针，结构体成员介绍请参考 <a href="#">表3-248. 结构体enet_ptp_sysptime_struct</a>
返回值	
-	-

例如：

```
/* get the current system time */
```

```
enet_ptp_sysptime_struct sysptime;
```

```
enet_ptp_system_time_get(&sysptime);
```

### 函数 enet\_ptp\_pps\_output\_frequency\_config

函数enet\_ptp\_pps\_output\_frequency\_config描述见下表：

表 3-355. 函数 enet\_ptp\_pps\_output\_frequency\_config

函数名称	enet_ptp_pps_output_frequency_config
函数原型	void enet_ptp_pps_output_frequency_config(uint32_t freq);
功能描述	配置PPS输出频率
先决条件	-
被调用函数	-
输入参数{in}	
freq	输出频率
ENET_PPISOFC_1HZ	PPS输出1Hz
ENET_PPISOFC_2HZ	PPS输出2Hz
ENET_PPISOFC_4HZ	PPS输出4Hz
ENET_PPISOFC_8HZ	PPS输出8Hz
ENET_PPISOFC_16HZ	PPS输出16Hz
ENET_PPISOFC_32HZ	PPS输出32Hz
ENET_PPISOFC_64HZ	PPS输出64Hz
ENET_PPISOFC_128HZ	PPS输出128Hz
ENET_PPISOFC_256HZ	PPS输出256Hz

6HZ	
ENET_PPISOFC_51 2HZ	PPS输出512Hz
ENET_PPISOFC_10 24HZ	PPS输出1024Hz
ENET_PPISOFC_20 48HZ	PPS输出2048Hz
ENET_PPISOFC_40 96HZ	PPS输出4096Hz
ENET_PPISOFC_81 92HZ	PPS输出8192Hz
ENET_PPISOFC_16 384HZ	PPS输出16384Hz
ENET_PPISOFC_32 768HZ	PPS输出32768Hz
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PPS output frequency as 1Hz */
```

```
enet_ptp_pps_output_frequency_config(ENET_PPISOFC_1HZ);
```

### 函数 enet\_ptp\_start

函数The enet\_ptp\_start描述见下表：

表 3-356. 函数 enet\_ptp\_start

函数名称	enet_ptp_start
函数原型	void enet_ptp_start(int32_t updatemethod, uint32_t init_sec, uint32_t init_subsec, uint32_t carry_cfg, uint32_t accuracy_cfg);
功能描述	配置和启动PTP时间戳计数器
先决条件	-
被调用函数	-
输入参数{in}	
updatemethod	更新方式
ENET_PTP_FINEMODE	精细校正
ENET_PTP_COARSEMODE	粗校正
输入参数{in}	
init_sec	初始化系统时间秒值

输入参数{in}	
<b>init_subsec</b>	初始化系统时间亚秒值
输入参数{in}	
<b>carry_cfg</b>	增加到时间戳加数寄存器的值（使用精细校正）
输入参数{in}	
<b>accuracy_cfg</b>	系统时间增加的亚秒值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* gconfigure and start PTP timestamp counter*/
enet_ptp_start(ENET_PTP_FINEMODE, 10, 10, 10, 10);
```

### 函数 enet\_ptp\_finecorrection\_adjfreq

函数enet\_ptp\_finecorrection\_adjfreq描述见下表：

表 3-357. 函数 enet\_ptp\_finecorrection\_adjfreq

函数名称	enet_ptp_finecorrection_adjfreq
函数原型	void enet_ptp_finecorrection_adjfreq(int32_t carry_cfg);
功能描述	通过PTP时间戳加数寄存器精调系统时间
先决条件	-
被调用函数	-
输入参数{in}	
<b>carry_cfg</b>	增加到PTP加数寄存器的数值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* adjust frequency in fine method by configure addend register */
enet_ptp_finecorrection_adjfreq(10);
```

### 函数 enet\_ptp\_coarsecorrection\_systime\_update

函数enet\_ptp\_coarsecorrection\_systime\_update描述见下表：

表 3-358. 函数 enet\_ptp\_coarsecorrection\_systime\_update

函数名称	enet_ptp_coarsecorrection_systime_update
函数原型	void enet_ptp_coarsecorrection_systime_update(enet_ptp_systime_struct *systime_struct);

功能描述	粗调系统时间
先决条件	-
被调用函数	-
输入参数{in}	
sys_time_struct	初始化结构体，结构体成员参考 <a href="#">表3-248. 结构体enet_ptp_sys_time_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* update system time in coarse method */
enet_ptp_sys_time_struct sys_time_struct;
enet_ptp_coarsecorrection_sys_time_update (&sys_time_struct);
```

### 函数 enet\_ptp\_finecorrection\_settime

函数enet\_ptp\_finecorrection\_settime描述见下表：

表 3-359. 函数 enet\_ptp\_finecorrection\_settime

函数名称	enet_ptp_finecorrection_settime
函数原形	void enet_ptp_finecorrection_settime(enet_ptp_sys_time_struct * sys_time_struct);
功能描述	精调系统时间
先决条件	-
被调用函数	-
输入参数{in}	
sys_time_struct	初始化结构体，结构体成员参考 <a href="#">表3-248. 结构体enet_ptp_sys_time_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set system time in fine method */
enet_ptp_sys_time_struct sys_time_struct;
enet_ptp_finecorrection_settime (&sys_time_struct);
```

### 函数 enet\_ptp\_flag\_get

函数enet\_ptp\_flag\_get描述见下表：

表 3-360. 函数 `enet_ptp_flag_get`

函数名称	<code>enet_ptp_flag_get</code>
函数原形	<code>FlagStatus enet_ptp_flag_get(uint32_t flag);</code>
功能描述	获取PTP标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	需要检查的PTP标志位
<code>ENET_PTP_ADDEND_UPDATE</code>	加数寄存器更新
<code>ENET_PTP_SYSTIME_UPDATE</code>	时间戳更新
<code>ENET_PTP_SYSTIME_INIT</code>	时间戳初始化
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如：

```
/* get the ptp flag status */
```

```
FlagStatus status = enet_ptp_flag_get(ENET_PTP_ADDEND_UPDATE);
```

### 函数 `enet_initpara_reset`

函数`enet_initpara_reset`描述见下表：

表 3-361. 函数 `enet_initpara_reset`

函数名称	<code>enet_initpara_reset</code>
函数原型	<code>void enet_initpara_reset(void);</code>
功能描述	复位 ENET initpara struct, 需在 <code>enet_initpara_config()</code> 函数前调用
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 ENET initpara struct */
```

```
enet_initpara_reset();
```

## 3.12. EXMC

外部存储器控制器EXMC，用来访问各种片外存储器。章节[3.12.1](#)描述了EXMC的寄存器列表，章节[3.12.2](#)对EXMC库函数进行说明。

### 3.12.1. 外设寄存器说明

EXMC寄存器列表如下表所示：

**表 3-362. EXMC 寄存器**

寄存器名称	寄存器描述
EXMC_SNCTL	SRAM/NOR Flash控制寄存器
EXMC_SNTCFG	SRAM/NOR Flash时序寄存器
EXMC_SNWTCFG	SRAM/NOR Flash写时序寄存器
EXMC_NPCTL	NAND flash/PC card控制寄存器
EXMC_NPINTEN	NAND flash/PC card中断使能寄存器
EXMC_NPCTCFG	NAND flash/PC card通用空间时序寄存器
EXMC_NPATCFG	NAND flash/PC card属性空间时序寄存器
EXMC_PIOTCFG3	PC card I/O空间时序寄存器
EXMC_NECC	NAND flash ECC结果寄存器

### 3.12.2. 外设库函数说明

EXMC库函数列表如下表所示：

**表 3-363. EXMC 库函数**

库函数名称	库函数描述
exmc_norsram_deinit	复位EXMC NOR/DRAM region x
exmc_norsram_struct_para_init	初始化结构体exmc_norsram_parameter_struct
exmc_norsram_init	初始化EXMC NOR/DRAM region x
exmc_norsram_enable	使能EXMC NOR/PSRAM region x
exmc_norsram_disable	禁用EXMC NOR/PSRAM region x
exmc_norsram_page_size_config	配置DRAM页大小
exmc_nand_deinit	复位EXMC NAND bankx
exmc_nand_struct_para_init	初始化结构体exmc_norsram_parameter_struct
exmc_nand_init	初始化EXMC NAND bank x
exmc_nand_enable	使能EXMC NAND bank x
exmc_nand_disable	禁用EXMC NAND bank x
exmc_nand_ecc_config	配置EXMC NAND ECC功能
exmc_ecc_get	获取EXMC ECC值
exmc_pccard_deinit	复位EXMC PC card bank
exmc_pccard_struct_para_init	初始化结构体exmc_pccard_parameter_struct
exmc_pccard_init	初始化EXMC PC card bank

库函数名称	库函数描述
exmc_pccard_enable	使能EXMC PC card bank
exmc_pccard_disable	禁用EXMC PC card bank
exmc_interrupt_enable	使能EXMC中断
exmc_interrupt_disable	禁用EXMC中断
exmc_flag_get	获取EXMC状态
exmc_flag_clear	清除EXMC状态
exmc_interrupt_flag_get	获取EXMC中断状态
exmc_interrupt_flag_clear	清除EXMC中断状态

### 结构体 exmc\_norsram\_timing\_parameter\_struct

表 3-364. 结构体 exmc\_norsram\_timing\_parameter\_struct

成员名称	功能描述
asyn_access_mode	异步访问模式
syn_data_latency	数据延迟，同步访问模式时有效
syn_clk_division	同步时钟分频比，同步访问模式时有效
bus_latency	总线延迟
asyn_data_setup_time	数据建立时间，异步访问模式时有效
asyn_address_hold_time	地址保持时间，异步访问模式时有效
asyn_address_setup_time	地址建立时间，异步访问模式时有效

### 结构体 exmc\_norsram\_parameter\_struct

表 3-365. 结构体 exmc\_norsram\_parameter\_struct

成员名称	功能描述
norsram_region	选择EXMC NOR/SRAM region
write_mode	写模式（同步模式或者异步模式）
extended_mode	使能或者禁用扩展模式
asyn_wait	使能或者禁用异步等待功能
nwait_signal	在同步突发模式中，使能或者禁用NWAIT信号
memory_write	使能或者禁用写操作
nwait_config	配置NWAIT信号，同步访问模式时有效
wrap_burst_mode	使能或者禁用非对齐成组模式
nwait_polarity	指定NWAIT的极性
burst_mode	使能或者禁用突发模式
databus_width	指定外部存储器数据总线宽度
memory_type	指定外部存储器的类型
address_data_mux	数据线/地址线复用是否复用
read_write_timing	未用扩展模式时，读时序参数和写时序参数；或采用扩展模式时，读时序参数，

成员名称	功能描述
	结构体成员参考 <a href="#">表3-364. 结构体 <code>exmc_norsram_timing_parameter_struct</code></a>
write_timing	使用扩展模式时，写时序参数，结构体成员参考 <a href="#">表3-364. 结构体 <code>exmc_norsram_timing_parameter_struct</code></a>

### 结构体 `exmc_nand_pccard_timing_parameter_struct`

表 3-366. 结构体 `exmc_nand_pccard_timing_parameter_struct`

成员名称	功能描述
databus_hiztime	写操作时数据总线高阻时间
holdtime	地址保持时间（写操作时数据保持时间）
waittime	等待时间（保持命令的最小时间）
setuptime	地址信号的建立时间

### 结构体 `exmc_nand_parameter_struct`

表 3-367. 结构体 `exmc_nand_parameter_struct`

成员名称	功能描述
nand_bank	选择EXMC NAND bank
ecc_size	ECC块大小
atr_latency	ALE至RE的延迟
ctr_latency	CLE至RE的延迟
ecc_logic	配置ECC使能或禁用
databus_width	NAND flash数据宽度
wait_feature	配置NWAIT信号使能或禁用
common_space_timing	NAND flash通用空间时序配置，结构体成员参考 <a href="#">表3-366. 结构体 <code>exmc_nand_pccard_timing_parameter_struct</code></a>
attribute_space_timing	NAND flash属性空间时序配置，结构体成员参考 <a href="#">表3-366. 结构体 <code>exmc_nand_pccard_timing_parameter_struct</code></a>

### 结构体 `exmc_pccard_parameter_struct`

表 3-368. 结构体 `exmc_pccard_parameter_struct`

成员名称	功能描述
atr_latency	ALE至RE的延迟
ctr_latency	CLE至RE的延迟
wait_feature	配置NWAIT信号使能或禁用
common_space_timing	PC card通用空间时序配置，结构体成员参考 <a href="#">表3-366. 结构体 <code>exmc_nand_pccard_timing_parameter_struct</code></a>
attribute_space_timing	PC card属性空间时序配置，结构体成员参考 <a href="#">表3-366. 结构体 <code>exmc_nand_pccard_timing_parameter_struct</code></a>
io_space_timing	PC card I/O空间时序配置，结构体成员参考 <a href="#">表3-366. 结构体 <code>exmc_nand_pccard_timing_parameter_struct</code></a>

函数 `exmc_norsram_deinit`

函数`exmc_norsram_deinit`描述见下表：

表 3-369. 函数 `exmc_norsram_deinit`

函数名称	<code>exmc_norsram_deinit</code>
函数原型	<code>void exmc_norsram_deinit(uint32_t exmc_norsram_region);</code>
功能描述	复位NOR/SRAM region
先决条件	-
被调用函数	-
输入参数{in}	
<code>exmc_norsram_region</code>	EXMC NOR/SRAM region
<code>EXMC_BANK0_NORSRAM_REGIONx</code>	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize the EXMC NOR/SRAM region1 of bank0 */
```

```
exmc_norsram_deinit(EXMC_BANK0_NORSRAM_REGION1);
```

函数 `exmc_norsram_struct_para_init`

函数`exmc_norsram_struct_para_init`描述见下表：

表 3-370. 函数 `exmc_norsram_struct_para_init`

函数名称	<code>exmc_norsram_struct_para_init</code>
函数原型	<code>void exmc_norsram_struct_para_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);</code>
功能描述	初始化结构体 <code>exmc_norsram_parameter_struct</code>
先决条件	-
被调用函数	-
输入参数{in}	
<code>exmc_norsram_init_struct</code>	初始化结构体，结构体成员参考 <a href="#">表3-365. 结构体 <code>exmc_norsram_parameter_struct</code></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the struct nor_init_struct */
```

```
exmc_norsram_parameter_struct nor_init_struct;
```

```
exmc_norsram_struct_para_init (&nor_init_struct);
```

### 函数 exmc\_norsram\_init

函数exmc\_norsram\_init描述见下表：

**表 3-371. 函数 exmc\_norsram\_init**

函数名称	exmc_norsram_init
函数原型	void exmc_norsram_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);
功能描述	初始化NOR/SRAM region
先决条件	-
被调用函数	-
输入参数{in}	
exmc_norsram_init_struct	初始化结构体，结构体成员参考 <a href="#">表3-365. 结构体 exmc_norsram_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize EXMC NOR/SRAM bank */
```

```
exmc_norsram_parameter_struct lcd_init_struct;
```

```
exmc_norsram_timing_parameter_struct lcd_timing_init_struct;
```

```
/* configure timing parameter */
```

```
lcd_timing_init_struct.asyn_access_mode = EXMC_ACCESS_MODE_A;
```

```
lcd_timing_init_struct.syn_data_latency = EXMC_DATA_LATENCY_2_CLK;
```

```
lcd_timing_init_struct.syn_clk_division = EXMC_SYN_CLOCK_RATIO_DISABLE;
```

```
lcd_timing_init_struct.bus_latency = 1;
```

```
lcd_timing_init_struct.asyn_data_setup_time = 5;
```

```
lcd_timing_init_struct.asyn_address_hold_time = 2;
```

```
lcd_timing_init_struct.asyn_address_setup_time = 2;
```

```
/* configure EXMC bus parameters */
```

```
lcd_init_struct.norsram_region = EXMC_BANK0_NORSRAM_REGION1;
```

```

lcd_init_struct.write_mode = EXMC_ASYNC_WRITE;

lcd_init_struct.extended_mode = DISABLE;

lcd_init_struct.asyn_wait = DISABLE;

lcd_init_struct.nwait_signal = DISABLE;

lcd_init_struct.memory_write = ENABLE;

lcd_init_struct.nwait_config = EXMC_NWAIT_CONFIG_BEFORE;

lcd_init_struct.wrap_burst_mode = DISABLE;

lcd_init_struct.nwait_polarity = EXMC_NWAIT_POLARITY_LOW;

lcd_init_struct.burst_mode = DISABLE;

lcd_init_struct.databus_width = EXMC_NOR_DATABUS_WIDTH_16B;

lcd_init_struct.memory_type = EXMC_MEMORY_TYPE_SRAM;

lcd_init_struct.address_data_mux = DISABLE;

lcd_init_struct.read_write_timing = &lcd_timing_init_struct;

lcd_init_struct.write_timing = &lcd_timing_init_struct;

exmc_norsram_init(&lcd_init_struct);

```

### 函数 exmc\_norsram\_enable

函数exmc\_norsram\_enable描述见下表：

**表 3-372. 函数 exmc\_norsram\_enable**

函数名称	exmc_norsram_enable
函数原型	void exmc_norsram_enable(uint32_t exmc_norsram_region);
功能描述	使能EXMC NOR/SRAM region
先决条件	-
被调用函数	-
输入参数{in}	
exmc_norsram_region	EXMC NOR/SRAM region
EXMC_BANK0_NORSRAM_REGIONx	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the EXMC NOR/SRAM region1 of bank0 */
```

```
exmc_norsram_enable(EXMC_BANK0_NORSRAM_REGION1);
```

### 函数 **exmc\_norsram\_disable**

函数exmc\_norsram\_disable描述见下表：

**表 3-373. 函数 exmc\_norsram\_disable**

函数名称	exmc_norsram_disable
函数原型	void exmc_norsram_disable(uint32_t exmc_norsram_region);
功能描述	禁用EXMC NOR/SRAM region
先决条件	-
被调用函数	-
输入参数{in}	
exmc_norsram_region	EXMC NOR/SRAM region
EXMC_BANK0_NORSRAM_REGIONx	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the EXMC NOR/SRAM region1 of bank0 */
```

```
exmc_norsram_disable(EXMC_BANK0_NORSRAM_REGION1);
```

### 函数 **exmc\_norsram\_page\_size\_config**

函数exmc\_norsram\_page\_size\_config描述见下表：

**表 3-374. 函数 exmc\_norsram\_page\_size\_config**

函数名称	exmc_norsram_page_size_config
函数原型	void exmc_norsram_page_size_config(uint32_t page_size);
功能描述	配置CRAM页大小
先决条件	-
被调用函数	-
输入参数{in}	
page_size	CRAM页大小
EXMC_CRAM_AUTO_SPLIT	页边界自动突发分割
EXMC_CRAM_PAGE_SIZE_128_BYTES	页大小128字节

EXMC_CRAM_PAGE_SIZE_256_BYTE	页大小256字节
EXMC_CRAM_PAGE_SIZE_512_BYTE	页大小512字节
EXMC_CRAM_PAGE_SIZE_1024_BYTES	页大小1024字节
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
exmc_norsram_page_size_config (EXMC_CRAM_PAGE_SIZE_128_BYTES);
```

### 函数 exmc\_nand\_deinit

函数exmc\_nand\_deinit描述见下表:

表 3-375. 函数 exmc\_nand\_deinit

函数名称	exmc_nand_deinit
函数原型	void exmc_nand_deinit(uint32_t exmc_nand_bank);
功能描述	复位EXMC NAND bank
先决条件	-
被调用函数	-
输入参数{in}	
exmc_nand_bank	EXMC NAND bank
EXMC_BANKx_NAND	x=1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* deinitialize EXMC NOR/SRAM bank1 */
```

```
exmc_norsram_deinit(EXMC_BANK1_NAND);
```

## 函数 exmc\_nand\_struct\_para\_init

函数exmc\_nand\_struct\_para\_init描述见下表：

**表 3-376. 函数 exmc\_nand\_struct\_para\_init**

函数名称	exmc_nand_struct_para_init
函数原型	void exmc_nand_struct_para_init(exmc_nand_parameter_struct* exmc_nand_init_struct);
功能描述	初始化结构体exmc_nand_parameter_struct
先决条件	-
被调用函数	-
输入参数{in}	
exmc_nand_init_struct	初始化结构体，结构体成员参考 <a href="#">表3-367. 结构体 exmc_nand_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the struct nand_init_struct */
exmc_nand_parameter_struct nand_init_struct;
exmc_nand_struct_para_init (&nand_init_struct);
```

## 函数 exmc\_nand\_init

函数exmc\_nand\_init描述见下表：

**表 3-377. 函数 exmc\_nand\_init**

函数名称	exmc_nand_init
函数原型	void exmc_nand_init(exmc_nand_parameter_struct* exmc_nand_init_struct);
功能描述	初始化EXMC NAND bank
先决条件	-
被调用函数	-
输入参数{in}	
exmc_nand_init_struct	初始化结构体，结构体成员参考 <a href="#">表3-367. 结构体 exmc_nand_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
exmc_nand_parameter_struct nand_init_struct;
```

```

exmc_nand_pccard_timing_parameter_struct nand_timing_init_struct;

/* EXMC configuration */

nand_timing_init_struct.setuptime = 5;

nand_timing_init_struct.waittime = 4;

nand_timing_init_struct.holdtime = 2;

nand_timing_init_struct.databus_hiztime = 2;

nand_init_struct.nand_bank = EXMC_BANK1_NAND;

nand_init_struct.ecc_size = EXMC_ECC_SIZE_2048BYTES;

nand_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_HCLK;

nand_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_HCLK;

nand_init_struct.ecc_logic = ENABLE;

nand_init_struct.databus_width = EXMC_NAND_DATABUS_WIDTH_8B;

nand_init_struct.wait_feature = ENABLE;

nand_init_struct.common_space_timing = &nand_timing_init_struct;

nand_init_struct.attribute_space_timing = &nand_timing_init_struct;

exmc_nand_init(&nand_init_struct);

```

### 函数 exmc\_nand\_enable

函数exmc\_nand\_enable描述见下表：

**表 3-378. 函数 exmc\_nand\_enable**

函数名称	exmc_nand_enable
函数原型	void exmc_nand_enable(uint32_t exmc_nand_bank);
功能描述	使能EXMC NAND bank
先决条件	-
被调用函数	-
输入参数{in}	
exmc_nand_bank	EXMC NAND bank
EXMC_BANKx_NAND	x=1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable EXMC NAND bank1 */
```

```
exmc_nand_enable(EXMC_BANK1_NAND);
```

### 函数 **exmc\_nand\_disable**

函数exmc\_nand\_disable描述见下表：

**表 3-379. 函数 exmc\_nand\_disable**

函数名称	exmc_nand_disable
函数原型	exmc_nand_disable(uint32_t exmc_nand_bank);
功能描述	禁用EXMC NAND bank
先决条件	-
被调用函数	-
输入参数{in}	
<b>exmc_nand_bank</b>	EXMC NAND bank
<i>EXMC_BANKx_NAND</i>	x=1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable EXMC NAND bank1 */
```

```
exmc_nand_disable(EXMC_BANK1_NAND);
```

### 函数 **exmc\_nand\_ecc\_config**

函数exmc\_nand\_ecc\_config描述见下表：

**表 3-380. 函数 exmc\_nand\_ecc\_config**

函数名称	exmc_nand_ecc_config
函数原型	void exmc_nand_ecc_config(uint32_t exmc_nand_bank, ControlStatus newvalue);
功能描述	使能或禁用EXMC NAND ECC功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>exmc_nand_bank</b>	EXMC NAND bank
<i>EXMC_BANKx_NAND</i>	x=1,2
输入参数{in}	
<b>newvalue</b>	ENABLE或DISABLE
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable the EXMC NAND ECC function */
```

```
exmc_nand_ecc_config(EXMC_BANK1_NAND, ENABLE);
```

### 函数 **exmc\_ecc\_get**

函数exmc\_ecc\_get描述见下表：

**表 3-381. 函数 exmc\_ecc\_get**

函数名称	exmc_ecc_get
函数原型	uint32_t exmc_ecc_get(uint32_t exmc_nand_bank);
功能描述	获取EXMC NAND ECC值
先决条件	-
被调用函数	-
输入参数{in}	
exmc_nand_bank	EXMC NAND bank
EXMC_BANKx_NAND	x=1,2
输出参数{out}	
-	-
返回值	
uint32_t	ECC计算的值

例如：

```
/* get the EXMC ECC value */
```

```
uint32_t ecc_value;
```

```
ecc_value = exmc_ecc_get(EXMC_BANK1_NAND);
```

### 函数 **exmc\_pccard\_deinit**

函数exmc\_pccard\_deinit描述见下表：

**表 3-382. 函数 exmc\_pccard\_deinit**

函数名称	exmc_pccard_deinit
函数原型	void exmc_pccard_deinit(void);
功能描述	复位EXMC PC card bank
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 EXMC PC card bank */
```

```
exmc_pccard_deinit();
```

### 函数 exmc\_pccard\_struct\_para\_init

函数exmc\_pccard\_struct\_para\_init描述见下表：

表 3-383. 函数 exmc\_pccard\_struct\_para\_init

函数名称	exmc_pccard_struct_para_init
函数原型	void exmc_pccard_struct_para_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);
功能描述	初始化结构体exmc_pccard_parameter_struct
先决条件	-
被调用函数	-
输入参数{in}	
exmc_pccard_init_struct	初始化结构体，结构体成员参考 <a href="#">表3-368. 结构体 exmc_pccard_parameter_struct</a> 结构体 exmc_pccard_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the struct pccard_init_struct */
```

```
exmc_pccard_parameter_struct pccard_init_struct;
```

```
exmc_pccard_struct_para_init (&pccard_init_struct);
```

### 函数 exmc\_pccard\_init

函数exmc\_pccard\_init描述见下表：

表 3-384. 函数 exmc\_pccard\_init

函数名称	exmc_pccard_init
函数原型	void exmc_pccard_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);

功能描述	初始化EXMC PC card bank
先决条件	-
被调用函数	-
输入参数{in}	
exmc_pccard_init_struct	初始化结构体，结构体成员参考 <a href="#">表3-368. 结构体 exmc_pccard_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
exmc_pccard_parameter_struct pccard_init_struct;

exmc_nand_pccard_timing_parameter_struct pccard_timing_init_struct;

/* EXMC configuration */

pccard_timing_init_struct.setuptime = 5;

pccard_timing_init_struct.waittime = 4;

pccard_timing_init_struct.holdtime = 2;

pccard_timing_init_struct.databus_hiztime = 2;

pccard_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_HCLK;

pccard_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_HCLK;

pccard_init_struct.wait_feature = ENABLE;

pccard_init_struct.common_space_timing = &pccard_timing_init_struct;

pccard_init_struct.attribute_space_timing = &pccard_timing_init_struct;

pccard_init_struct.io_space_timing = &pccard_timing_init_struct;

exmc_pccard_init(&pccard_init_struct);
```

### 函数 exmc\_pccard\_enable

函数exmc\_pccard\_enable描述见下表：

**表 3-385. 函数 exmc\_pccard\_enable**

函数名称	exmc_pccard_enable
函数原型	void exmc_pccard_enable(void);
功能描述	使能EXMC PC card bank
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 EXMC PC card bank */
```

```
exmc_pccard_enable();
```

### 函数 exmc\_pccard\_disable

函数exmc\_pccard\_disable描述见下表：

表 3-386. 函数 exmc\_pccard\_disable

函数名称	exmc_pccard_disable
函数原型	void exmc_pccard_disable(void);
功能描述	禁用EXMC PC card bank
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 EXMC PC card bank */
```

```
exmc_pccard_disable();
```

### 函数 exmc\_interrupt\_enable

函数exmc\_interrupt\_enable描述见下表：

表 3-387. 函数 exmc\_interrupt\_enable

函数名称	exmc_interrupt_enable
函数原型	void exmc_interrupt_enable(uint32_t exmc_bank, uint32_t interrupt);
功能描述	使能EXMC中断
先决条件	-
被调用函数	-
输入参数{in}	
exmc_bank	EXMC外设

EXMC_BANK1_NAND	NAND bank1
EXMC_BANK2_NAND	NAND bank2
EXMC_BANK3_PCCARD	PC Card bank
输入参数{in}	
interrupt	EXMC中断状态
EXMC_NAND_PCCARD_INT_FLAG_RISE	上升沿中断
EXMC_NAND_PCCARD_INT_FLAG_LEVEL	高电平中断
EXMC_NAND_PCCARD_INT_FLAG_FALL	下降沿中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable rising edge interrupt and flag t*/
```

```
exmc_interrupt_enable(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

### 函数 exmc\_interrupt\_disable

函数exmc\_interrupt\_disable描述见下表：

表 3-388. 函数 exmc\_interrupt\_disable

函数名称	exmc_interrupt_disable
函数原型	void exmc_interrupt_disable(uint32_t exmc_bank, uint32_t interrupt);
功能描述	禁用EXMC中断
先决条件	-
被调用函数	-
输入参数{in}	
exmc_bank	EXMC外设
EXMC_BANK1_NAND	NAND bank1
EXMC_BANK2_NAND	NAND bank2
EXMC_BANK3_PCCARD	PC Card bank

CARD	
输入参数{in}	
interrupt	EXMC中断状态
EXMC_NAND_PCCARD_INT_FLAG_RISE	上升沿中断
EXMC_NAND_PCCARD_INT_FLAG_LEVEL	高电平中断
EXMC_NAND_PCCARD_INT_FLAG_FALL	下降沿中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable rising edge interrupt and flag */
```

```
exmc_interrupt_disable(EXMC_BANK1_NAND,  
EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

### 函数 exmc\_flag\_get

函数exmc\_flag\_get描述见下表:

表 3-389. 函数 exmc\_flag\_get

函数名称	exmc_flag_get
函数原型	FlagStatus exmc_flag_get(uint32_t exmc_bank,uint32_t flag);
功能描述	获取EXMC状态
先决条件	-
被调用函数	-
输入参数{in}	
exmc_bank	EXMC外设
EXMC_BANK1_NAND	NAND bank1
EXMC_BANK2_NAND	NAND bank2
EXMC_BANK3_PCCARD	PC Card bank
输入参数{in}	
flag	EXMC标志状态
EXMC_NAND_PCCARD	上升沿状态

<i>ARD_FLAG_RISE</i>	
<i>EXMC_NAND_PCC</i> <i>ARD_FLAG_LEVEL</i>	高电平状态
<i>EXMC_NAND_PCC</i> <i>ARD_FLAG_FALL</i>	下降沿状态
<i>EXMC_NAND_PCC</i> <i>ARD_FLAG_FIFOE</i>	FIFO空状态
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如:

```
/* check rising edge status is set or not*/
```

```
if(RESET != exmc_flag_get(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_FLAG_RISE));
```

### 函数 **exmc\_flag\_clear**

函数exmc\_flag\_clear描述见下表:

表 3-390. 函数 **exmc\_flag\_clear**

函数名称	exmc_flag_clear
函数原型	FlagStatus exmc_flag_clear (uint32_t exmc_bank,uint32_t flag);
功能描述	清除EXMC状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>exmc_bank</b>	EXMC外设
<i>EXMC_BANK1_NAND</i>	NAND bank1
<i>EXMC_BANK2_NAND</i>	NAND bank2
<i>EXMC_BANK3_PCCARD</i>	PC Card bank
输入参数{in}	
<b>flag</b>	EXMC标志状态
<i>EXMC_NAND_PCC</i> <i>ARD_FLAG_RISE</i>	上升沿状态
<i>EXMC_NAND_PCC</i> <i>ARD_FLAG_LEVEL</i>	高电平状态
<i>EXMC_NAND_PCC</i> <i>ARD_FLAG_FALL</i>	下降沿状态
<i>EXMC_NAND_PCC</i>	FIFO空状态

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

例如：

```
/* clear rising edge status */
```

```
exmc_flag_clear(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_FLAG_RISE);
```

### 函数 exmc\_interrupt\_flag\_get

函数exmc\_interrupt\_flag\_get描述见下表：

表 3-391. 函数 exmc\_interrupt\_flag\_get

函数名称	exmc_interrupt_flag_get
函数原型	FlagStatus exmc_interrupt_flag_get(uint32_t exmc_bank, uint32_t interrupt);
功能描述	获取EXMC中断状态
先决条件	-
被调用函数	-
输入参数{in}	
exmc_bank	EXMC中断状态
EXMC_BANK1_NAND	NAND bank1
EXMC_BANK2_NAND	NAND bank2
EXMC_BANK3_PCCARD	PC Card bank
输入参数{in}	
interrupt	中断
EXMC_NAND_PCCARD_INT_FLAG_RISE	上升沿中断
EXMC_NAND_PCCARD_INT_FLAG_LEVEL	高电平中断
EXMC_NAND_PCCARD_INT_FLAG_FALL	下降沿中断
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* check rising edge interrupt and flag is set or not*/
```

```
if(RESET!=exmc_interrupt_flag_get(EXMC_BANK1_NAND,EXMC_NAND_PCCARD_INT_FLAG_RISE));
```

### 函数 **exmc\_interrupt\_flag\_clear**

函数 **exmc\_interrupt\_flag\_clear** 描述见下表：

**表 3-392. 函数 **exmc\_interrupt\_flag\_clear****

函数名称	exmc_interrupt_flag_clear
函数原型	void exmc_interrupt_flag_clear(uint32_t exmc_bank,uint32_t interrupt);
功能描述	清除EXMC中断状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>exmc_bank</b>	EXMC中断状态
EXMC_BANK1_NAND	NAND bank1
EXMC_BANK2_NAND	NAND bank2
EXMC_BANK3_PCCARD	PC Card bank
输入参数{in}	
<b>interrupt</b>	中断
EXMC_NAND_PCCARD_INT_FLAG_RISE	上升沿中断
EXMC_NAND_PCCARD_INT_FLAG_LEVEL	高电平中断
EXMC_NAND_PCCARD_INT_FLAG_FALL	下降沿中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear rising edge interrupt and flag */
```

```
exmc_interrupt_flag_clear(EXMC_BANK1_NAND,EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

### 3.13. EXTI

EXTI是MCU中的中断/事件控制器，包括22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[3.13.1](#)描述了EXTI的寄存器列表，章节[3.13.1](#)对EXTI库函数进行说明。

#### 3.13.1. 外设寄存器说明

EXTI寄存器列表如下表所示：

表 3-393. 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

#### 3.13.2. 外设库函数说明

EXTI库函数列表如下表所示：

表 3-394. 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 3-395. 枚举类型 `exti_line_enum`

成员名称	功能描述
EXTI_0	EXTI中断线0

EXTI_1	EXTI中断线1
EXTI_2	EXTI中断线2
EXTI_3	EXTI中断线3
EXTI_4	EXTI中断线4
EXTI_5	EXTI中断线5
EXTI_6	EXTI中断线6
EXTI_7	EXTI中断线7
EXTI_8	EXTI中断线8
EXTI_9	EXTI中断线9
EXTI_10	EXTI中断线10
EXTI_11	EXTI中断线11
EXTI_12	EXTI中断线12
EXTI_13	EXTI中断线13
EXTI_14	EXTI中断线14
EXTI_15	EXTI中断线15
EXTI_16	EXTI中断线16
EXTI_17	EXTI中断线17
EXTI_18	EXTI中断线18
EXTI_19	EXTI中断线19
EXTI_20	EXTI中断线20
EXTI_21	EXTI中断线21

### 枚举类型 `exti_mode_enum`

表 3-396. 枚举类型 `exti_mode_enum`

成员名称	功能描述
EXTI_INTERRUPT	EXTI中断模式
EXTI_EVENT	EXTI事件模式

### 枚举类型 `exti_trig_type_enum`

表 3-397. 枚举类型 `exti_trig_type_enum`

成员名称	功能描述
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 3-398. 函数 `exti_deinit`

函数名称	<code>exti_deinit</code>
------	--------------------------

函数原形	void exti_deinit(void);
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 3-399. 函数 exti\_init

函数名称	exti_init
函数原形	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 3-395. 枚举类型 exti_line_enum</a>
输入参数{in}	
mode	EXTI 模式, 参考 <a href="#">表 3-396. 枚举类型 exti_mode_enum</a>
输入参数{in}	
trig_type	触发类型, 参考 <a href="#">表 3-397. 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 3-400. 函数 exti\_interrupt\_enable**

函数名称	exti_interrupt_enable
函数原形	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 3-395. 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 3-401. 函数 exti\_interrupt\_disable**

函数名称	exti_interrupt_disable
函数原形	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 3-395. 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 3-402. 函数 **exti\_event\_enable**

函数名称	exti_event_enable
函数原形	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3-395. 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 3-403. 函数 **exti\_event\_disable**

函数名称	exti_event_disable
函数原形	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 3-395. 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 3-404. 函数 exti\_software\_interrupt\_enable**

函数名称	exti_software_interrupt_enable
函数原形	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 3-395. 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 3-405. 函数 exti\_software\_interrupt\_disable**

函数名称	exti_software_interrupt_disable
函数原形	void exti_software_interrupt_disable(exti_line_enum linex);
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}	
linex	EXTI 线 x, 参考 <a href="#">表 3-395. 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 3-406. 函数 **exti\_flag\_get**

函数名称	exti_flag_get
函数原形	FlagStatus exti_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 3-395. 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 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 3-407. 函数 **exti\_flag\_clear**

函数名称	exti_flag_clear
函数原形	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 3-395. 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 3-408. 函数 `exti_interrupt_flag_get`

函数名称	<code>exti_interrupt_flag_get</code>
函数原形	<code>FlagStatus exti_interrupt_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 3-395. 枚举类型 <code>exti_line_enum</code></a>
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	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 3-409. 函数 `exti_interrupt_flag_clear`

函数名称	<code>exti_interrupt_flag_clear</code>
函数原形	<code>void exti_interrupt_flag_clear(exti_line_enum linex);</code>
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}	
<b>linex</b>	EXTI 线 x，参考 <a href="#">表 3-395. 枚举类型 <code>exti_line_enum</code></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
exti_interrupt_flag_clear(EXTI_0);
```

### 3.14. 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[3.14.1](#)描述了FMC的寄存器列表，章节[3.14.2](#)对FMC库函数进行说明。

#### 3.14.1. 外设寄存器说明

FMC寄存器列表如下：

表 3-410. FMC 寄存器

寄存器	描述
FMC_WS	等待状态寄存器
FMC_KEY	解锁寄存器
FMC_OBKEY	选项字节解锁寄存器
FMC_STAT	状态寄存器
FMC_CTL	控制寄存器
FMC_ADDR	地址寄存器
FMC_ECCCS	ECC控制及状态寄存器
FMC_OBSTAT	选项字节状态寄存器
FMC_WP	擦写/编程保护寄存器
FMC_PID	产品ID寄存器

#### 3.14.2. 外设库函数说明

FMC固件库函数列举如下表：

表 3-411. FMC 固件库函数

函数名称	函数描述
fmc_unlock	解锁FMC主编程块操作
fmc_lock	锁定FMC主编程块操作
fmc_wsnt_set	设置FMC等待状态计数值
fmc_prefetch_enable	使能pre-fetch
fmc_prefetch_disable	禁能pre-fetch
fmc_ibus_enable	使能IBUS缓存区
fmc_ibus_disable	禁能IBUS缓存区
fmc_dbus_enable	使能DBUS缓存区
fmc_dbus_disable	禁能DBUS缓存区
fmc_ibus_reset	复位IBUS缓存区
fmc_dbus_reset	复位DBUS缓存区
fmc_page_erase	FMC页擦除
fmc_mass_erase	FMC全片擦除
fmc_doubleword_program	在相应地址字编程
ob_unlock	解锁选项字节操作

函数名称	函数描述
ob_lock	锁定选项字节操作
ob_erase	擦除选项字节
ob_write_protection_enable	使能写保护
ob_security_protection_config	配置安全保护
ob_user_write	写用户选项字节
ob_data_program	写数据选项字节
ob_user_get	获取用户选项字节
ob_data_get	获取数据选项字节
ob_write_protection_get	获取写保护选项字节
ob_security_protection_flag_get	获取安全保护选项字节
fmc_ecc_address_get	获取发生ECC错误的地址
fmc_flag_get	检查标志位是否置位
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中断标志状态

### 枚举类型 fmc\_state\_enum

表 3-412. 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_PGERR	编程错误
FMC_PGAERR	编程对齐错误
FMC_WPERR	写保护错误
FMC_TOERR	超时错误
FMC_OB_HSPC	高保护等级

### 枚举类型 fmc\_flag\_enum

表 3-413. 枚举类型 fmc\_flag\_enum

枚举名称	枚举描述
FMC_FLAG_BUSY	Flash忙状态
FMC_FLAG_PGER R	Flash编程错误标志
FMC_FLAG_PGAE RR	Flash编程对齐错误标志
FMC_FLAG_WPER R	Flash擦写保护错误标志
FMC_FLAG_END	Flash编程完成标志

枚举名称	枚举描述
FMC_FLAG_ECCC OR	单个位检测并纠正错误标志
FMC_FLAG_ECCD ET	双位检测错误标志
FMC_FLAG_SYSE CC	系统存储ECC错误标志
FMC_FLAG_MFEC C	主闪存ECC错误标志
FMC_FLAG_OTPE CC	OTP ECC错误标志
FMC_FLAG_OBEC C	选项字节ECC错误标志
FMC_FLAG_OBEC CDET	加载选项字节到寄存器时双位检测ECC错误标志
FMC_FLAG_OBER R	选项字节错误标志

#### 枚举类型 `fmc_interrupt_flag_enum`

表 3-414. 枚举类型 `fmc_interrupt_flag_enum`

枚举名称	枚举描述
FMC_INT_FLAG_P GERR	Flash编程错误中断标志
FMC_INT_FLAG_P GAERR	Flash编程对齐错误中断标志
FMC_INT_FLAG_W PERR	Flash擦写保护错误中断标志
FMC_INT_FLAG_E ND	Flash操作结束中断标志
FMC_INT_FLAG_E CCCOR	单个位检测并纠正错误中断标志
FMC_INT_FLAG_E CCDET	双位检测错误中断标志
FMC_INT_FLAG_O BECCDET	加载选项字节到寄存器时双位检测ECC错误中断标志

#### 枚举类型 `fmc_interrupt_enum`

表 3-415. 枚举类型 `fmc_interrupt_enum`

枚举名称	枚举描述
FMC_INT_ERR	FMC错误中断
FMC_INT_END	FMC操作结束中断
FMC_INT_ECCCO	单个位检测并纠正错误中断

枚举名称	枚举描述
R	
FMC_INT_ECCDET	双位检测错误中断

### 函数 fmc\_unlock

函数fmc\_unlock描述见下表：

**表 3-416. 函数 fmc\_unlock**

函数名称	fmc_unlock
函数原型	void fmc_unlock(void);
功能描述	解锁Flash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 FMC operation */
```

```
fmc_unlock();
```

### 函数 fmc\_lock

函数fmc\_lock描述见下表：

**表 3-417. 函数 fmc\_lock**

函数名称	fmc_lock
函数原型	void fmc_lock(void);
功能描述	锁定Flash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 FMC operation */
```

fmc\_lock();

### 函数 fmc\_wscnt\_set

函数fmc\_wscnt\_set描述见下表:

表 3-418. 函数 fmc\_wscnt\_set

函数名称	fmc_wscnt_set
函数原型	void fmc_wscnt_set(uint32_t wscnt);
功能描述	设置等待状态计数值
先决条件	-
被调用函数	-
输入参数{in}	
wscnt	等待状态计数值
FMC_WAIT_STATE_0	FMC 0个等待状态
FMC_WAIT_STATE_1	FMC 1个等待状态
FMC_WAIT_STATE_2	FMC 2个等待状态
FMC_WAIT_STATE_3	FMC 3个等待状态
FMC_WAIT_STATE_4	FMC 4个等待状态
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set 1wait state */
```

```
fmc_wscnt_set (WS_WSCNT_1);
```

### 函数 fmc\_prefetch\_enable

函数fmc\_prefetch\_enable描述见下表:

表 3-419. 函数 fmc\_prefetch\_enable

函数名称	fmc_prefetch_enable
函数原型	void fmc_prefetch_enable(void);
功能描述	使能pre-fetch
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 3-420. 函数 fmc\_prefetch\_disable

函数名称	fmc_prefetch_disable
函数原型	void fmc_prefetch_disable (void);
功能描述	禁能pre-fetch
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\_ibus\_enable

函数fmc\_ibus\_enable描述见下表：

表 3-421. 函数 fmc\_ibus\_enable

函数名称	fmc_ibus_enable
函数原型	void fmc_ibus_enable(void);
功能描述	使能IBUS缓存区
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 IBUS cache */
```

```
fmc_ibus_enable();
```

### 函数 fmc\_ibus\_disable

函数fmc\_ibus\_disable描述见下表：

**表 3-422. 函数 fmc\_ibus\_disable**

函数名称	fmc_ibus_disable
函数原型	void fmc_ibus_disable(void);
功能描述	禁能IBUS缓存区
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 IBUS cache */
```

```
fmc_ibus_disable();
```

### 函数 fmc\_dbus\_enable

函数fmc\_dbus\_enable描述见下表：

**表 3-423. 函数 fmc\_dbus\_enable**

函数名称	fmc_dbus_enable
函数原型	void fmc_dbus_enable(void);
功能描述	使能DBUS缓存区
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 DBUS cache */
```

```
fmc_dbus_enable();
```

### 函数 fmc\_dbus\_disable

函数fmc\_dbus\_disable描述见下表：

表 3-424. 函数 fmc\_dbus\_disable

函数名称	fmc_dbus_disable
函数原型	void fmc_dbus_disable(void);
功能描述	禁能DBUS缓存区
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 DBUS cache */
```

```
fmc_dbus_disable();
```

### 函数 fmc\_ibus\_reset

函数fmc\_ibus\_reset描述见下表：

表 3-425. 函数 fmc\_ibus\_reset

函数名称	fmc_ibus_reset
函数原型	void fmc_ibus_reset (void);
功能描述	复位IBUS缓存区
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 IBUS cache */
```

```
fmc_ibus_reset();
```

### 函数 fmc\_dbus\_reset

函数fmc\_dbus\_reset描述见下表：

表 3-426. 函数 fmc\_dbus\_reset

函数名称	fmc_dbus_reset
函数原型	void fmc_dbus_reset(void);
功能描述	复位DBUS缓存区
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 DBUS cache */
```

```
fmc_dbus_reset();
```

### 函数 fmc\_page\_erase

函数fmc\_page\_erase描述见下表：

表 3-427. 函数 fmc\_page\_erase

函数名称	fmc_page_erase
函数原型	fmc_state_enum fmc_page_erase(uint32_t page_address);
功能描述	页擦除
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
page_address	页擦除首地址
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，详细请参考 <a href="#">表3-412. 枚举类型fmc_state_enum</a>

例如：

```
/* erase page */
```

```
fmc_unlock();
```

```
fmc_state_enum state = fmc_page_erase(0x08004000);
```

### 函数 fmc\_mass\_erase

函数fmc\_mass\_erase描述见下表:

表 3-428. 函数 fmc\_mass\_erase

函数名称	fmc_mass_erase
函数原型	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	
fmc_state_enum	FMC状态, 详细请参考 <a href="#">表3-412. 枚举类型fmc_state_enum</a>

例如:

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

```
fmc_unlock();
```

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

### 函数 fmc\_doubleword\_program

函数fmc\_doubleword\_program描述见下表:

表 3-429. 函数 fmc\_doubleword\_program

函数名称	fmc_doubleword_program
函数原型	fmc_state_enum fmc_doubleword_program(uint32_t address, uint64_t data);
功能描述	对相应地址字编程
先决条件	fmc_unlock, fmc_page_erase/fmc_mass_erase
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	

<b>fmc_state_enum</b>	FMC状态，详细请参考 <a href="#">表3-412. 枚举类型fmc_state_enum</a>
-----------------------	--

例如：

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

fmc_unlock();

fmc_page_erase(0x08004000);

fmc_state_enum state = fmc_doubleword_program(0x08004000, 0xaabbccddaabbccdd);
```

### 函数 ob\_unlock

函数ob\_unlock描述见下表：

**表 3-430. 函数 ob\_unlock**

函数名称	ob_unlock
函数原型	void ob_unlock(void);
功能描述	解锁选项字节
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：

```
/* unlock the option bytes operation */

fmc_unlock();

ob_unlock();
```

### 函数 ob\_lock

函数ob\_lock描述见下表：

**表 3-431. 函数 ob\_lock**

函数名称	ob_lock
函数原型	void ob_lock(void);
功能描述	锁定选项字节操作
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：

```
/* lock the option bytes operation */
```

```
fmc_unlock();
```

```
ob_lock();
```

### 函数 ob\_erase

函数ob\_erase描述见下表：

表 3-432. 函数 ob\_erase

函数名称	ob_erase
函数原型	fmc_state_enum ob_erase(void);
功能描述	擦除选项字节
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，详细请参考 <a href="#">表3-412. 枚举类型fmc_state_enum</a>

例如：

```
/* erase the FMC option bytes */
```

```
fmc_state_enum state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```
state = ob_erase();
```

### 函数 ob\_write\_protection\_enable

函数ob\_write\_protection\_enable描述见下表：

表 3-433. 函数 ob\_write\_protection\_enable

函数名称	ob_write_protection_enable
函数原型	fmc_state_enum ob_write_protection_enable(uint32_t ob_wp);
功能描述	使能写保护

先决条件	ob_unlock
被调用函数	-
输入参数{in}	
ob_wp	写保护单元
OB_WP_NONE	全片不保护
OB_WPx	特定写保护单元 (x= 0 ...31)
OB_WP_ALL	全片写保护
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 详细请参考 <a href="#">表3-412. 枚举类型fmc_state_enum</a>

例如:

```
/* enable write protection */

fmc_state_enum state;

fmc_unlock();

ob_unlock();

state = ob_write_protection_enable(OB_WP7);
```

### 函数 ob\_security\_protection\_config

函数ob\_security\_protection\_config描述见下表:

表 3-434. 函数 ob\_security\_protection\_config

函数名称	ob_security_protection_config
函数原型	fmc_state_enum ob_security_protection_config (uint8_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}	
ob_spc	安全保护
FMC_NSPC	无安全保护
FMC_LSPC	低等级安全保护
FMC_HSPC	号等级安全保护
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 详细请参考 <a href="#">表3-412. 枚举类型fmc_state_enum</a>

例如:

```
/* enable low security protection */
```

```
fmc_state enum state;
```

```
ob_unlock();
```

```
state = ob_security_protection_config(FMC_LSPC);
```

### 函数 ob\_user\_write

函数ob\_user\_write描述见下表:

表 3-435. 函数 ob\_user\_write

函数名称	ob_user_write
函数原型	fmc_state_enum ob_user_write(uint8_t ob_fwdgt, uint8_t ob_deepsleep, uint8_t ob_stdby, uint8_t ob_boot);
功能描述	编辑用户选项字节
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
ob_fwdgt	选项字节看门狗数值
OB_FWDGT_SW	软件看门狗
OB_FWDGT_HW	硬件看门狗
输入参数{in}	
ob_deepsleep	选项字节深度睡眠复位值
OB_DEEPSLEEP_N_RST	进入深度睡眠时不复位
OB_DEEPSLEEP_R_RST	进入深度睡眠时产生复位
输入参数{in}	
ob_stdby	选项字节待机复位值
OB_STDBY_N_RST	进入待机时不复位
OB_STDBY_R_RST	进入待机时产生复位
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 详细请参考 <a href="#">表3-412. 枚举类型fmc_state_enum</a>

例如:

```
/* configure user option byte */
```

```
fmc_state enum state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```
state = ob_user_write(OB_FWDGT_HW,OB_DEEPSLEEP_R_RST, OB_STDBY_R_RST);
```

函数 **ob\_data\_program**

函数ob\_data\_program描述见下表：

表 3-436. 函数 **ob\_data\_program**

函数名称	ob_data_program
函数原型	fmc_state_enum ob_data_program(uint32_t address, uint8_t 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}	
address	编程数据选项字节地址
输入参数{in}	
data	所编程数值
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，详细请参考 <a href="#">表3-412. 枚举类型fmc_state_enum</a>

例如：

```
/* program option bytes data */

fmc_state_enum state;

fmc_unlock();

ob_unlock();

state = ob_data_program(0x1fff804, 0x56);
```

函数 **ob\_user\_get**

函数ob\_user\_get描述见下表：

表 3-437. 函数 **ob\_user\_get**

函数名称	ob_user_get
函数原型	uint8_t ob_user_get(void);
功能描述	获取用户选项字节
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（0x0 – 0xFF）

例如：

```
/* get the FMC user option bytes */

uint8_t user;

user = ob_user_get();
```

### 函数 ob\_data\_get

函数ob\_data\_get描述见下表：

表 3-438. 函数 ob\_data\_get

函数名称	ob_data_get
函数原型	uint16_t ob_data_get(void);
功能描述	获取数据选项字节
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 FMC data option bytes */

uint16_t data;

data = ob_data_get();
```

### 函数 ob\_write\_protection\_get

函数ob\_write\_protection\_get描述见下表：

表 3-439. 函数 ob\_write\_protection\_get

函数名称	ob_write_protection_get
函数原型	uint32_t ob_write_protection_get(void);
功能描述	获取选项字节写保护数值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

<b>uint32_t</b>	选项字节写保护数值（0x0 – 0xFFFFFFFF）
-----------------	-----------------------------

例如：

```
/* get the FMC option bytes write protection */
```

```
uint32_t wp;
```

```
wp = ob_write_protection_get();
```

### 函数 **ob\_security\_protection\_flag\_get**

函数ob\_security\_protection\_flag\_get描述见下表：

**表 3-440. 函数 ob\_security\_protection\_flag\_get**

函数名称	ob_security_protection_flag_get
函数原型	FlagStatus ob_security_protection_flag_get(void);
功能描述	获取安全保护状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the FMC option bytes security protection */
```

```
FlagStatus flag;
```

```
flag = ob_security_protection_flag_get();
```

### 函数 **fmc\_ecc\_address\_get**

函数fmc\_ecc\_address\_get描述见下表：

**表 3-441. 函数 fmc\_ecc\_address\_get**

函数名称	fmc_ecc_address_get
函数原型	uint16_t fmc_ecc_address_get(void);
功能描述	获取发生ECC错误的地址
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
uint16_t	0x0000-0x3FFF

例如：

```
/* get the address where ECC error occur on */
```

```
uint16_t ecc_addr = fmc_ecc_address_get();
```

### 函数 fmc\_flag\_get

函数fmc\_flag\_get描述见下表：

**表 3-442. 函数 fmc\_flag\_get**

函数名称	fmc_flag_get
函数原型	FlagStatus fmc_flag_get(fmc_flag_enum flag);
功能描述	检查FMC标志是否置位
先决条件	-
被调用函数	-
输入参数{in}	
flag	FMC标志，详情参考枚举变量 <a href="#">表3-413. 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_FLAG_END);
```

### 函数 fmc\_flag\_clear

函数fmc\_flag\_clear描述见下表：

**表 3-443. 函数 fmc\_flag\_clear**

函数名称	fmc_flag_clear
函数原型	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，详情参考枚举变量 <a href="#">表3-413. 枚举类型fmc_flag_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear FMC end flag */  
  
fmc_flag_clear(FMC_FLAG_END);
```

### 函数 fmc\_interrupt\_enable

函数fmc\_interrupt\_enable描述见下表：

表 3-444. 函数 fmc\_interrupt\_enable

函数名称	fmc_interrupt_enable
函数原型	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，详情参考枚举变量 <a href="#">表3-415. 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_INT_END);
```

### 函数 fmc\_interrupt\_disable

函数fmc\_interrupt\_disable描述见下表：

表 3-445. 函数 fmc\_interrupt\_disable

函数名称	fmc_interrupt_disable
函数原型	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，详情参考枚举变量 <a href="#">表3-415. 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_INT_END);
```

### 函数 fmc\_interrupt\_flag\_get

函数fmc\_interrupt\_flag\_get描述见下表：

**表 3-446. 函数 fmc\_interrupt\_flag\_get**

函数名称	fmc_interrupt_flag_get
函数原型	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	中断标志，详情参考枚举变量 <a href="#">表3-414. 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_INT_FLAG_PGERR);
```

### 函数 fmc\_interrupt\_flag\_clear

函数fmc\_interrupt\_flag\_clear描述见下表：

**表 3-447. 函数 fmc\_interrupt\_flag\_clear**

函数名称	fmc_interrupt_flag_clear
函数原型	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	中断标志，详情参考枚举变量 <a href="#">表3-414. 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_INT_FLAG_PGERR);
```

### 3.15. 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[3.15.1](#)描述了FWDGT的寄存器列表，章节[3.15.2](#)对FWDGT库函数进行说明。

#### 3.15.1. 外设寄存器说明

FWDGT寄存器列表如下表所示：

表 3-448. 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

#### 3.15.2. 外设库函数说明

FWDGT库函数列表如下表所示：

表 3-449. 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和FWDGT_RLD的写操作
fwdgt_write_disable	失能对寄存器FWDGT_PSC和FWDGT_RLD的写操作
fwdgt_enable	使能FWDGT
fwdgt_prescaler_value_config	配置FWDGT预分频值
fwdgt_reload_value_config	配置FWDGT重装载值
fwdgt_config	设置FWDGT重装载值、预分频值
fwdgt_counter_reload	按照FWDGT_RLD寄存器的值重装载IWDG计数器
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 3-450. 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和FWDGT_RLD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 */
```

```
fwdgt_write_enable( );
```

### 函数 fwdgt\_write\_disable

函数fwdgt\_write\_disable描述见下表：

**表 3-451. 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	失能对寄存器FWDGT_PSC和FWDGT_RLD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 and FWDGT_RLD */
```

```
fwdgt_write_disable( );
```

### 函数 fwdgt\_enable

函数fwdgt\_enable描述见下表：

**表 3-452. 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 free watchdog timer counter */
```

```
fwdgt_enable( );
```

### 函数 **fwdgt\_prescaler\_value\_config**

函数fwdgt\_prescaler\_value\_config描述见下表:

**表 3-453. 函数 fwdgt\_prescaler\_value\_config**

函数名称	fwdgt_prescaler_value_config
函数原型	void fwdgt_prescaler_value_config (uint16_t prescaler_value);
功能描述	配置FWDGT预分频值
先决条件	-
被调用函数	-
输入参数{in}	
<b>prescaler_value</b>	指定FWDGT预分频值
<i>FWDGT_PSC_DIV4</i>	FWDGT预分频值设为4
<i>FWDGT_PSC_DIV8</i>	FWDGT预分频值设为8
<i>FWDGT_PSC_DIV16</i>	FWDGT预分频值设为16
<i>FWDGT_PSC_DIV32</i>	FWDGT预分频值设为32
<i>FWDGT_PSC_DIV64</i>	FWDGT预分频值设为64
<i>FWDGT_PSC_DIV128</i>	FWDGT预分频值设为128
<i>FWDGT_PSC_DIV256</i>	FWDGT预分频值设为256
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the FWDGT counter prescaler value */
```

```
fwdgt_prescaler_value_config(FWDGT_PSC_DIV8);
```

### 函数 **fwdgt\_reload\_value\_config**

函数fwdgt\_reload\_value\_config描述见下表:

**表 3-454. 函数 fwdgt\_reload\_value\_config**

函数名称	fwdgt_reload_value_config
函数原型	void fwdgt_reload_value_config(uint16_t reload_value);

功能描述	配置FWDGT重装载值
先决条件	-
被调用函数	-
输入参数{in}	
reload_value	指定FWDGT重装载值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the FWDGT counter reload value */
```

```
fwdgt_reload_value_config(625);
```

### 函数 fwdgt\_config

函数fwdgt\_config描述见下表：

表 3-455. 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或SUCCESS
-----------	---------------

例如:

```
/* configure FWDGT counter clock: 40KHz(IRC40K) / 64 = 0.625 KHz */
fwdgt_config(2*500, FWDGT_PSC_DIV64);
```

### 函数 fwdgt\_counter\_reload

函数fwdgt\_counter\_reload描述见下表:

表 3-456. 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IWDG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\_flag\_get

函数fwdgt\_flag\_get描述见下表:

表 3-457. 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_RU D	重装载值更新进行中
输出参数{out}	
-	-

返回值	
FlagStatus	SET or RESET

例如：

```

/* test if a prescaler value update is on going */

FlagStatus status;

status = fwdgt_flag_get (FWDGT_FLAG_PUD);

if(status == RESET)
{
    ...
}
else
{
    ...
}

```

## 3.16. GPIO

GPIO用来实现各片上设备的逻辑输入/输出功能。章节[3.16.1](#)描述了GPIO的寄存器列表，章节[3.16.2](#)对GPIO库函数进行说明。

### 3.16.1. 外设寄存器说明

GPIO寄存器列表如下表所示：

**表 3-458. GPIO 寄存器**

寄存器名称	寄存器描述
GPIOx_CTL0	GPIO端口控制寄存器0
GPIOx_CTL1	GPIO端口控制寄存器1
GPIOx_ISTAT	GPIO端口输入状态寄存器
GPIOx_OCTL	GPIO端口输出控制寄存器
GPIOx_BOP	GPIO端口位操作寄存器
GPIOx_BC	GPIO位清除寄存器
GPIOx_LOCK	GPIO端口配置锁定寄存器
GPIOx_SPD	GPIO端口位速度寄存器
AFIO_EC	AFIO事件控制寄存器
AFIO_PCF0	AFIO端口配置寄存器0
AFIO_EXTISS0	AFIO端口EXTI源选择寄存器0
AFIO_EXTISS1	AFIO端口EXTI源选择寄存器1

寄存器名称	寄存器描述
AFIO_EXTISS2	AFIO端口EXTI源选择寄存器2
AFIO_EXTISS3	AFIO端口EXTI源选择寄存器3
AFIO_PCF1	AFIO端口配置寄存器1
AFIO_CPSCTL	IO补偿控制寄存器
AFIO_PCFA	AFIO端口配置寄存器A
AFIO_PCFB	AFIO端口配置寄存器B
AFIO_PCFC	AFIO端口配置寄存器C
AFIO_PCFD	AFIO端口配置寄存器D
AFIO_PCFE	AFIO端口配置寄存器E
AFIO_PCFG	AFIO端口配置寄存器G

### 3.16.2. 外设库函数说明

GPIO库函数列表如下表所示：

**表 3-459. GPIO 库函数**

库函数名称	库函数描述
gpio_deinit	复位外设GPIOx
gpio_afio_deinit	复位AFIO
gpio_init	GPIO参数初始化
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_pin_remap_config	配置GPIO引脚重映射
gpio_afio_port_config	配置AFIO端口的备用功能
gpio_ethernet_phy_select	以太网MII或RMII PHY选择（互联型）
gpio_exti_source_select	选择哪个引脚作为EXTI源
gpio_event_output_config	配置事件输出
gpio_event_output_enable	事件输出使能
gpio_event_output_disable	事件输出除能
gpio_pin_lock	相应的引脚配置被锁定
gpio_compensation_config	配置I/O补偿单元
gpio_compensation_flag_get	检测I/O补偿单元是否准备好

#### 函数 gpio\_deinit

函数gpio\_deinit描述见下表：

表 3-460. 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,G)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset GPIOA */
gpio_deinit (GPIOA);
```

### 函数 **gpio\_afio\_deinit**

函数gpio\_afio\_deinit描述见下表：

表 3-461. 函数 **gpio\_afio\_deinit**

函数名称	gpio_afio_deinit
函数原型	void gpio_afio_deinit(void);
功能描述	复位AFIO
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 alternate function */
gpio_afio_deinit();
```

### 函数 **gpio\_init**

函数gpio\_init描述见下表：

表 3-462. 函数 gpio\_init

函数名称	gpio_init
函数原型	void gpio_init(uint32_t gpio_periph, uint32_t mode, uint32_t speed, uint32_t pin);
功能描述	GPIO参数初始化
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,G)
输入参数{in}	
gpio_mode	GPIO引脚模式
GPIO_MODE_AIN	模拟输入模式
GPIO_MODE_IN_FLOATING	浮空输入模式
GPIO_MODE_IPD	下拉输入模式
GPIO_MODE_IPU	上拉输入模式
GPIO_MODE_OUT_OD	开漏输出模式
GPIO_MODE_OUT_PP	推挽输出模式
GPIO_MODE_AF_OD	AFIO开漏输出模式
GPIO_MODE_AF_PP	AFIO推挽输出模式
输入参数{in}	
speed	GPIO输出最大速度
GPIO_OSPEED_10MHZ	最大速度为10MHZ
GPIO_OSPEED_2MHZ	最大速度为2MHZ
GPIO_OSPEED_50MHZ	最大速度为50MHZ
GPIO_OSPEED_MAX	最大速度大于50MHZ
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:

```
/* config PA0 as analog input mode*/
```

```
gpio_init (GPIOA, GPIO_MODE_AIN, GPIO_OSPEED_50MHZ, GPIO_PIN_0);
```

### 函数 gpio\_bit\_set

函数gpio\_bit\_set描述见下表:

表 3-463. 函数 gpio\_bit\_set

函数名称	gpio_bit_set
函数原型	void gpio_bit_set(uint32_t gpio_periph, uint32_t pin);
功能描述	置位GPIO引脚值
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,G)
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:

```
/* 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 3-464. 函数 gpio\_bit\_reset

函数名称	gpio_bit_reset
函数原型	void gpio_bit_reset(uint32_t gpio_periph, uint32_t pin);
功能描述	复位GPIO引脚值
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,G)

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	
-	-

例如：

```
/* reset PA0 */
```

```
gpio_bit_reset(GPIOA, GPIO_PIN_0);
```

### 函数 gpio\_bit\_write

函数gpio\_bit\_write描述见下表：

表 3-465. 函数 gpio\_bit\_write

函数名称	gpio_bit_write
函数原型	void gpio_bit_write(uint32_t gpio_periph, uint32_t pin, bit_status bit_value);
功能描述	将特定的值写入GPIO引脚
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,G)
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
输入参数{in}	
<b>bit_value</b>	设置或清除
<i>RESET</i>	清除引脚值
<i>SET</i>	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 3-466. 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,G)
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 3-467. 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GPIO引脚的输入值
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,G)
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	

<b>FlagStatus</b>	SET或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 3-468. 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GPIO端口的输入值
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,G)
输出参数{out}	
-	-
返回值	
uint16_t	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 3-469. 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,G)

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或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 3-470. 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}	
<b>gpio_periph</b>	GPIO端口
<i>GPIOx</i>	端口选择 (x = A,B,C,D,E,F,G)
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\_pin\_remap\_config

函数gpio\_pin\_remap\_config描述见下表:

表 3-471. 函数 gpio\_pin\_remap\_config

函数名称	gpio_pin_remap_config
------	-----------------------

函数原型	void gpio_pin_remap_config(uint32_t gpio_remap, ControlStatus newvalue);
功能描述	配置GPIO引脚重映射
先决条件	-
被调用函数	-
输入参数{in}	
gpio_remap	选择重映射
GPIO_SPI0_REMAP	SPI0重映射
GPIO_I2C0_REMAP	I2C0重映射
GPIO_USART0_REMAP	USART0重映射
GPIO_USART1_REMAP	USART1重映射
GPIO_USART2_PARTIAL_REMAP	USART2部分重映射
GPIO_USART2_FULL_REMAP	USART2全部重映射
GPIO_TIMER0_PARTIAL_REMAP	TIMER0部分重映射
GPIO_TIMER0_FULL_REMAP	TIMER0全部重映射
GPIO_TIMER1_PARTIAL_REMAP0	TIMER1部分重映射0
GPIO_TIMER1_PARTIAL_REMAP1	TIMER部分重映射1
GPIO_TIMER1_FULL_REMAP	TIMER1完全重映射
GPIO_TIMER2_PARTIAL_REMAP	TIMER2部分重映射
GPIO_TIMER2_FULL_REMAP	TIMER2完全重映射
GPIO_TIMER3_REMAP	TIMER3重映射
GPIO_PD01_REMAP	PD01重映射
GPIO_TIMER4CH3_I	TIMER4内部通道3重映射
GPIO_ADC0_ETRGR_T_REMAP	ADC0常规转换外部触发重映射（高密度型）
GPIO_ADC1_ETRGR_T_REMAP	ADC1常规转换外部触发重映射（高密度型）
GPIO_ENET_REMAP	以太网重映射（互联型）
GPIO_SWJ_NONJTRST_REMAP	全部的SWJ(JTAG-DP + SW-DP)，但是不包括NJTRST
GPIO_SWJ_SWDPE	JTAG-DP除能，SW-DP使能

<i>NABLE_REMAP</i>	
<i>GPIO_SWJ_DISABLE_REMAP</i>	JTAG-DP除能，SW-DP除能
<i>GPIO_SPI2_REMAP</i>	SPI2重映射
<i>GPIO_TIMER1ITR1_REMAP</i>	TIMER1内部触发1重映射（互联型）
<i>GPIO_PTP_PPS_REMAP</i>	以太网PTP PPS重映射（互联型）
<i>GPIO_TIMER8_REMAP</i>	TIMER8重映射
<i>GPIO_TIMER9_REMAP</i>	TIMER9重映射
<i>GPIO_TIMER10_REMAP</i>	TIMER10重映射
<i>GPIO_TIMER12_REMAP</i>	TIMER12重映射
<i>GPIO_TIMER13_REMAP</i>	TIMER13重映射
<i>GPIO_EXMC_NADV_REMAP</i>	EXMC_NADV 连接/断开
<i>GPIO_CTC_REMAP0</i>	CTC重映射(PD15)
<i>GPIO_CTC_REMAP1</i>	CTC重映射(PF0)
输入参数{in}	
<b>newvalue</b>	是否使能
<i>ENABLE</i>	使能
<i>DISABLE</i>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 remapping */
```

```
gpio_pin_remap_config (GPIO_SPI0_REMAP, ENABLE);
```

### 函数 **gpio\_afio\_port\_config**

函数gpio\_afio\_port\_config的描述见下表：

**表 3-472. 函数 **gpio\_afio\_port\_config****

函数名称	gpio_afio_port_config
函数原型	void gpio_afio_port_config(uint32_t afio_function, ControlStatus newvalue);
功能描述	configure AFIO port alternate function

先决条件	-
被调用函数	-
输入参数{in}	
afio_function	配置端口的备用功能
AFIO_PA2_CMP1_CFG	配置PA2备用功能到CMP1
AFIO_PA3_USBHS_CFG	配置PA3备用功能到USBHS
AFIO_PA5_USBHS_CFG	配置PA5备用功能到USBHS
AFIO_PA8_I2C2_CFG	配置PA8备用功能到I2C2
AFIO_PA8_SHRTIMER_CFG	配置PA8备用功能到SHRTIMER
AFIO_PA9_I2C2_CFG	配置PA9备用功能到I2C2
AFIO_PA9_SHRTIMER_CFG	配置PA9备用功能到SHRTIMER
AFIO_PA10_CMP5_CFG	配置PA10备用功能到CMP5
AFIO_PA10_SHRTIMER_CFG	配置PA10备用功能到SHRTIMER
AFIO_PA11_USART5_CFG	配置PA11备用功能到USART5
AFIO_PA11_SHRTIMER_CFG	配置PA11备用功能到SHRTIMER
AFIO_PA12_CMP1_CFG	配置PA12备用功能到CMP1
AFIO_PA12_USART5_CFG	配置PA12备用功能到USART5
AFIO_PA12_SHRTIMER_CFG	配置PA12备用功能到SHRTIMER
AFIO_PA15_SHRTIMER_CFG	配置PA15备用功能到SHRTIMER
AFIO_PB0_USBHS_CFG	配置PB0备用功能到USBHS
AFIO_PB1_CMP3_CFG	配置PB1备用功能到CMP3
AFIO_PB1_USBHS_CFG	配置PB1备用功能到USBHS
AFIO_PB1_SHRTIMER_CFG	配置PB1备用功能到SHRTIMER
AFIO_PB2_USBHS_CFG	配置PB2备用功能到USBHS

CFG	
AFIO_PB2_SHRTIMER_CFG	配置PB2备用功能到SHRTIMER
AFIO_PB3_SHRTIMER_CFG	配置PB3备用功能到SHRTIMER
AFIO_PB4_I2S2_CFG	配置PB4备用功能到I2S2
AFIO_PB4_I2C2_CFG	配置PB4备用功能到I2C2
AFIO_PB4_SHRTIMER_CFG	配置PB4备用功能到SHRTIMER
AFIO_PB5_I2C2_CFG	配置PB5备用功能到I2C2
AFIO_PB5_USBHS_CFG	配置PB5备用功能到USBHS
AFIO_PB5_SHRTIMER_CFG	配置PB5备用功能到SHRTIMER
AFIO_PB6_SHRTIMER_CFG	配置PB6备用功能到SHRTIMER
AFIO_PB7_SHRTIMER_CFG	配置PB7备用功能到SHRTIMER
AFIO_PB8_I2C2_CFG	配置PB8备用功能到I2C2
AFIO_PB8_SHRTIMER_CFG	配置PB8备用功能到SHRTIMER
AFIO_PB9_CMP1_CFG	配置PB9备用功能到CMP1
AFIO_PB9_SHRTIMER_CFG	配置PB9备用功能到SHRTIMER
AFIO_PB10_USBHS_CFG	配置PB10备用功能到USBHS
AFIO_PB10_SHRTIMER_CFG	配置PB10备用功能到SHRTIMER
AFIO_PB11_USBHS_CFG	配置PB11备用功能到USBHS
AFIO_PB11_SHRTIMER_CFG	配置PB11备用功能到SHRTIMER
AFIO_PB12_USBHS_CFG	配置PB12备用功能到USBHS
AFIO_PB12_SHRTIMER_CFG	配置PB12备用功能到SHRTIMER
AFIO_PB13_USBHS_CFG	配置PB13备用功能到USBHS

AFIO_PB13_SHRTIMER_CFG	配置PB13备用功能到SHRTIMER
AFIO_PB14_I2S1_CFG	配置PB14备用功能到I2S1
AFIO_PB14_SHRTIMER_CFG	配置PB14备用功能到SHRTIMER
AFIO_PB15_SHRTIMER_CFG	配置PB15备用功能到SHRTIMER
AFIO_PC0_USBHS_CFG	配置PC0备用功能到USBHS
AFIO_PC2_I2S1_CFG	配置PC2备用功能到I2S1
AFIO_PC2_USBHS_CFG	配置PC2备用功能到USBHS
AFIO_PC3_USBHS_CFG	配置PC3备用功能到USBHS
AFIO_PC6_CMP5_CFG	配置PC6备用功能到CMP5
AFIO_PC6_USART5_CFG	配置PC6备用功能到USART5
AFIO_PC6_SHRTIMER_CFG	配置PC6备用功能到SHRTIMER
AFIO_PC7_USART5_CFG	配置PC7备用功能到USART5
AFIO_PC7_SHRTIMER_CFG	配置PC7备用功能到SHRTIMER
AFIO_PC8_USART5_CFG	配置PC8备用功能到USART5
AFIO_PC8_SHRTIMER_CFG	配置PC8备用功能到SHRTIMER
AFIO_PC9_I2C2_CFG	配置PC9备用功能到I2C2
AFIO_PC9_SHRTIMER_CFG	配置PC9备用功能到SHRTIMER
AFIO_PC10_I2C2_CFG	配置PC10备用功能到I2C2
AFIO_PC11_I2S2_CFG	配置PC11备用功能到I2S2
AFIO_PC11_SHRTIMER_CFG	配置PC11备用功能到SHRTIMER
AFIO_PC12_SHRTIMER_CFG	配置PC12备用功能到SHRTIMER
AFIO_PD4_SHRTIMER	配置PD4备用功能到SHRTIMER

ER_CFG	
AFIO_PD5_SHRTIM ER_CFG	配置PD5备用功能到SHRTIMER
AFIO_PE0_SHRTIM ER_CFG	配置PE0备用功能到SHRTIMER
AFIO_PE1_SHRTIM ER_CFG	配置PE1备用功能到SHRTIMER
AFIO_PE8_CMP1_C FG	配置PE8备用功能到CMP1
AFIO_PE9_CMP3_C FG	配置PE9备用功能到CMP3
AFIO_PE10_CMP5_ CFG	配置PE10备用功能到CMP5
AFIO_PE11_CMP5_ CFG	配置PE11备用功能到CMP5
AFIO_PE12_CMP3_ CFG	配置PE12备用功能到CMP3
AFIO_PE13_CMP1_ CFG	配置PE13备用功能到CMP1
AFIO_PG6_SHRTIM ER_CFG	配置PG6备用功能到SHRTIMER_C
AFIO_PG7_USART5 _CFG	配置PG7备用功能到USART5
AFIO_PG7_SHRTIM ER_CFG	配置PG7备用功能到SHRTIMER
AFIO_PG9_USART5 _CFG	配置PG9备用功能到USART5
AFIO_PG10_SHRTI MER_CFG	配置PG10备用功能到SHRTIMER
AFIO_PG11_SHRTI MER_CFG	配置PG11备用功能到SHRTIMER
AFIO_PG12_SHRTI MER_CFG	配置PG12备用功能到SHRTIMER
AFIO_PG13_SHRTI MER_CFG	配置PG13备用功能到SHRTIMER
AFIO_PG14_USART 5_CFG	配置PG14备用功能到USART5
输入参数{in}	
newvalue	是否使能
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-

返回值	
-	-

Example:

```
/* configure PG14 alternate function to USART5 */
```

```
gpio_afio_port_config (AFIO_PG14_USART5_CFG, ENABLE);
```

### 函数 gpio\_ethernet\_phy\_select

函数gpio\_ethernet\_phy\_select描述见下表:

表 3-473. 函数 gpio\_ethernet\_phy\_select

函数名称	gpio_ethernet_phy_select
函数原型	void gpio_ethernet_phy_select(uint32_t enet_sel);
功能描述	以太网MII或RMII PHY选择（互联型）
先决条件	-
被调用函数	-
输入参数{in}	
enet_sel	以太网MII和RMII PHY选择
GPIO_ENET_PHY_MII	选择MII
GPIO_ENET_PHY_RMII	选择RMII
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ethernet MAC for connection with an RMII PHY */
```

```
gpio_ethernet_phy_select (GPIO_ENET_PHY_RMII);
```

### 函数 gpio\_exti\_source\_select

函数gpio\_exti\_source\_select描述见下表:

表 3-474. 函数 gpio\_exti\_source\_select

函数名称	gpio_exti_source_select
函数原型	void gpio_exti_source_select(uint8_t output_port, uint8_t output_pin);
功能描述	选择哪个引脚作为EXTI源
先决条件	-
被调用函数	-
输入参数{in}	
output_port	EXTI源端口

<code>GPIO_PORT_SOURCE_GPIOx</code>	EXTI源端口选择 (x = A,B,C,D,E,F,G)
输入参数{in}	
<code>output_pin</code>	源端口引脚
<code>GPIO_PIN_SOURCE_x</code>	源端口引脚选择 (x = 0..15)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* config PA0 as EXTI source */
```

```
gpio_exti_source_select (GPIO_PORT_SOURCE_GPIOA, GPIO_PIN_SOURCE_0);
```

### 函数 `gpio_event_output_config`

函数`gpio_event_output_config`描述见下表:

表 3-475. 函数 `gpio_event_output_config`

函数名称	<code>gpio_event_output_config</code>
函数原型	<code>void gpio_event_output_config(uint8_t output_port, uint8_t output_pin);</code>
功能描述	配置事件输出
先决条件	-
被调用函数	-
输入参数{in}	
<code>output_port</code>	GPIO事件输出端口
<code>GPIO_EVENT_PORT_GPIOx</code>	事件输出端口选择 (x = A,B,C,D,E)
输入参数{in}	
<code>output_pin</code>	GPIO事件输出引脚
<code>GPIO_EVENT_PIN_x</code>	事件引脚选择 (x = 0..15)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure PA0 as the output of event */
```

```
gpio_event_output_config (GPIO_EVENT_PORT_GPIOA, GPIO_EVENT_PIN_0);
```

## 函数 gpio\_event\_output\_enable

函数gpio\_event\_output\_enable描述见下表：

**表 3-476. 函数 gpio\_event\_output\_enable**

函数名称	gpio_event_output_enable
函数原型	void gpio_event_output_enable(void);
功能描述	事件输出使能
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 GPIO pin event output */
```

```
gpio_event_output_enable();
```

## 函数 gpio\_event\_output\_disable

函数gpio\_event\_output\_disable描述见下表：

**表 3-477. 函数 gpio\_event\_output\_disable**

函数名称	gpio_event_output_disable
函数原型	void gpio_event_output_disable(void);
功能描述	事件输出禁能
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 GPIO pin event output */
```

```
gpio_event_output_disable();
```

## 函数 gpio\_pin\_lock

函数gpio\_pin\_lock描述见下表：

**表 3-478. 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,G)
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\_compensation\_config

函数gpio\_compensation\_config描述见下表：

**表 3-479. 函数 gpio\_compensation\_config**

函数名称	gpio_compensation_config
函数原型	void gpio_compensation_config(uint32_t compensation);
功能描述	配置I/O补偿单元
先决条件	-
被调用函数	-
输入参数{in}	
compensation	指定I/O补偿单元模式
GPIO_COMPENSATION_ENABLE	I/O补偿单元使能
GPIO_COMPENSATION_DISABLE	I/O补偿单元禁能
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enabled I/O compensation cell */
```

```
gpio_compensation_config (GPIO_COMPENSATION_ENABLE);
```

### 函数 gpio\_compensation\_flag\_get

函数gpio\_compensation\_flag\_get描述见下表：

**表 3-480. 函数 gpio\_compensation\_flag\_get**

函数名称	gpio_compensation_flag_get
函数原型	FlagStatus gpio_compensation_flag_get(void);
功能描述	检测I/O补偿单元是否准备好
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* check the I/O compensation cell state */
```

```
FlagStatus cell_state;
```

```
cell_state = gpio_compensation_flag_get (void);
```

## 3.17. SHRTIMER

SHRTIMER具有高精度计数时钟，可用于高精度定时。它可以产生10个高精度的数字信号来灵活地控制电动机或用于电源管理应用。这10个数字信号可以独立输出，也可以耦合成5对互补信号输出。章节[3.17.1](#)描述了SHRTIMER的寄存器列表，章节[3.17.2](#)对SHRTIMER库函数进行说明。GD32EPRT系列没有SHRTIMER模块。

### 3.17.1. 外设寄存器说明

SHRTIMER寄存器列表如下表所示：

表 3-481. SHRTIMER 寄存器

寄存器名称	寄存器描述
<b>Master Timer寄存器</b>	
SHRTIMER_MTCTL0	Master_TIMER控制寄存器0
SHRTIMER_MTINTF	Master_TIMER中断标志寄存器
SHRTIMER_MTINTC	Master_TIMER中断标志清除寄存器
SHRTIMER_MTDMAINTEN	Master_TIMER DMA和中断使能寄存器
SHRTIMER_MTCNT	Master_TIMER计数器寄存器
SHRTIMER_MTCAR	Master_TIMER计数器自动重载寄存器
SHRTIMER_MTCREP	Master_TIMER重复计数寄存器
SHRTIMER_MTCMP0V	Master_TIMER比较0寄存器
SHRTIMER_MTCMP1V	Master_TIMER比较1寄存器
SHRTIMER_MTCMP2V	Master_TIMER比较2寄存器
SHRTIMER_MTCMP3V	Master_TIMER比较3寄存器
SHRTIMER_MTACTL	Master_TIMER附加控制寄存器
<b>Slave Time寄存器</b>	
SHRTIMER_STXCTL0	Slave_TIMERx控制寄存器0
SHRTIMER_STXINTF	Slave_TIMERx中断标志寄存器
SHRTIMER_STXINTC	Slave_TIMERx中断标志清除寄存器
SHRTIMER_STXDMAINTEN	Slave_TIMERx DMA和中断使能寄存器
SHRTIMER_STXCNT	Slave_TIMERx计数器寄存器
SHRTIMER_STXCAR	Slave_TIMERx计数器自动重载寄存器
SHRTIMER_STXCREP	Slave_TIMERx重复计数寄存器
SHRTIMER_STXCMP0V	Slave_TIMERx比较0寄存器
SHRTIMER_STXCMP0CP	Slave_TIMERx比较0复合寄存器
SHRTIMER_STXCMP1V	Slave_TIMERx比较1寄存器
SHRTIMER_STXCMP2V	Slave_TIMERx比较2寄存器
SHRTIMER_STXCMP3V	Slave_TIMERx比较3寄存器
SHRTIMER_STXCAP0V	Slave_TIMERx捕获0寄存器
SHRTIMER_STXCAP1V	Slave_TIMERx捕获1寄存器
SHRTIMER_STXDTCTL	Slave_TIMERx死区控制寄存器
SHRTIMER_STXCH0SET	Slave_TIMERx通道0置位请求寄存器
SHRTIMER_STXCH0RST	Slave_TIMERx通道0复位请求寄存器
SHRTIMER_STXCH1SET	Slave_TIMERx通道1置位请求寄存器
SHRTIMER_STXCH1RST	Slave_TIMERx通道1复位请求寄存器
SHRTIMER_STXEXEVFCFG 0	Slave_TIMERx外部事件滤波配置寄存器0
SHRTIMER_STXEXEVFCFG 1	Slave_TIMERx外部事件滤波配置寄存器1
SHRTIMER_STXCNTRST	Slave_TIMERx计数器复位寄存器
SHRTIMER_STXCSTL	Slave_TIMERx载波控制寄存器
SHRTIMER_STXCAP0TRG	Slave_TIMERx捕获0触发寄存器

寄存器名称	寄存器描述
SHRTIMER_STXCAP1TRG	Slave_TIMERx捕获1触发寄存器
SHRTIMER_STXCHCTL	Slave_TIMERx通道输出控制寄存器
SHRTIMER_STXFLTCTL	Slave_TIMERx故障控制寄存器
SHRTIMER_STXACTL	Slave_TIMERx附加控制寄存器
通用寄存器	
SHRTIMER_CTL0	SHRTIMER控制寄存器0
SHRTIMER_CTL1	SHRTIMER控制寄存器1
SHRTIMER_INTF	SHRTIMER中断标志寄存器
SHRTIMER_INTC	SHRTIMER中断标志清除寄存器
SHRTIMER_INTEN	SHRTIMER中断使能寄存器
SHRTIMER_CHOUTEN	SHRTIMER通道输出使能寄存器
SHRTIMER_CHOUTDIS	SHRTIMER通道输出禁能寄存器
SHRTIMER_CHOUTDISF	SHRTIMER通道输出禁能标志寄存器
SHRTIMER_BMCTL	SHRTIMER突发模式控制寄存器
SHRTIMER_BMSTRG	SHRTIMER突发模式启动触发寄存器
SHRTIMER_BMCMPV	SHRTIMER突发模式比较值寄存器
SHRTIMER_BMCAR	SHRTIMER突发模式计数器自动重载寄存器
SHRTIMER_EXEVCFG0	SHRTIMER外部事件配置寄存器0
SHRTIMER_EXEVCFG1	SHRTIMER外部事件配置寄存器1
SHRTIMER_EXEVDCTL	SHRTIMER外部事件数字滤波控制寄存器
SHRTIMER_ADCTRIGS0	SHRTIMER ADC触发源0寄存器
SHRTIMER_ADCTRIGS1	SHRTIMER ADC触发源1寄存器
SHRTIMER_ADCTRIGS2	SHRTIMER ADC触发源2寄存器
SHRTIMER_ADCTRIGS3	SHRTIMER ADC触发源3寄存器
SHRTIMER_DLLCCTL	SHRTIMER DLL校准控制寄存器
SHRTIMER_FLTINCFG0	SHRTIMER故障输入配置寄存器0
SHRTIMER_FLTINCFG1	SHRTIMER故障输入配置寄存器1
SHRTIMER_DMAUPMTR	SHRTIMER DMA更新Master_TIMER寄存器
SHRTIMER_DMAUPST0R	SHRTIMER DMA更新Slave_TIMER0寄存器
SHRTIMER_DMAUPST1R	SHRTIMER DMA更新Slave_TIMER1寄存器
SHRTIMER_DMAUPST2R	SHRTIMER DMA更新Slave_TIMER2寄存器
SHRTIMER_DMAUPST3R	SHRTIMER DMA更新Slave_TIMER3寄存器
SHRTIMER_DMAUPST4R	SHRTIMER DMA更新Slave_TIMER4寄存器
SHRTIMER_DMATB	SHRTIMER DMA传输缓冲寄存器

### 3.17.2. 外设库函数说明

SHRTIMER库函数列表如下表所示：

**表 3-482. SHRTIMER 库函数**

库函数名称	库函数描述
shrtimer_deinit	复位SHRTIMER

库函数名称	库函数描述
shrtimer_dll_calibration_start	配置和启动DLL校准
shrtimer_baseinit_struct_para_init	初始化shrtimer_baseinit_parameter_struct结构体变量
shrtimer_timers_base_init	初始化Master_TIMER/Slave_TIMER时基
shrtimer_timers_counter_enable	使能定时器的计数器
shrtimer_timers_counter_disable	禁能定时器的计数器
shrtimer_timers_update_event_enable	使能Master_TIMER/Slave_TIMER更新事件
shrtimer_timers_update_event_disable	禁能Master_TIMER/Slave_TIMER更新事件
shrtimer_software_update	软件触发Master_TIMER/Slave_TIMER更新事件
shrtimer_software_counter_reset	软件复位Master_TIMER/Slave_TIMER计数器
shrtimer_timerinit_struct_para_init	初始化shrtimer_timerinit_parameter_struct结构体变量
shrtimer_timers_waveform_init	定时器波形初始化
shrtimer_timercfg_struct_para_init	初始化shrtimer_timercfg_parameter_struct结构体变量
shrtimer_slavetimer_waveform_config	配置Slave_TIMER的波形
shrtimer_comparecfg_struct_para_init	初始化shrtimer_comparecfg_parameter_struct结构体
shrtimer_slavetimer_waveform_compare_config	配置Slave_TIMER波形的比较功能
shrtimer_channel_outputcfg_struct_para_init	初始化shrtimer_channel_outputcfg_parameter_struct结构体变量
shrtimer_slavetimer_waveform_channel_config	Slave_TIMER波形的通道配置
shrtimer_slavetimer_waveform_channel_software_request	软件产生通道置位或复位请求
shrtimer_slavetimer_waveform_channel_output_level_get	获取通道的输出电平
shrtimer_slavetimer_waveform_channel_state_get	获取通道的运行状态
shrtimer_deadtimecfg_struct_para_init	初始化shrtimer_deadtimecfg_parameter_struct结构体变量
shrtimer_slavetimer_deadtime_config	配置Slave_TIMER的死区时间
shrtimer_carriersignalcfg_struct_para_init	初始化shrtimer_carriersignalcfg_parameter_struct结构体变量
shrtimer_slavetimer_carriersignal_config	配置Slave_TIMER的载波功能
shrtimer_output_channel_enable	使能通道输出
shrtimer_output_channel_disable	禁能通道输出
shrtimer_mastertimer_compare_value_config	配置主定时器的比较寄存器值
shrtimer_mastertimer_compare_value_get	获取主定时器的比较寄存器值

库函数名称	库函数描述
shrtimer_slavetimer_compare_value_config	配置Slave_TIMER的比较寄存器值
shrtimer_slavetimer_compare_value_get	获取Slave_TIMER的比较寄存器值
shrtimer_timers_counter_value_config	配置Master_TIMER/Slave_TIMER的计数寄存器值
shrtimer_timers_counter_value_get	获取Master_TIMER/Slave_TIMER的计数寄存器值
shrtimer_timers_autoreload_value_config	配置Master_TIMER/Slave_TIMER的自动重载寄存器值
shrtimer_timers_autoreload_value_get	获取Master_TIMER/Slave_TIMER的自动重载寄存器值
shrtimer_timers_repetition_value_config	配置Master_TIMER/Slave_TIMER的重复计数寄存器值
shrtimer_timers_repetition_value_get	获取Master_TIMER/Slave_TIMER的重复计数寄存器值
shrtimer_exefilter_struct_para_init	初始化shrtimer_exefilter_parameter_struct结构体变量
shrtimer_slavetimer_exeevent_filtering_config	配置Slave_TIMER的外部事件滤波功能
shrtimer_exeeventcfg_struct_para_init	初始化shrtimer_exeeventcfg_parameter_struct结构体变量
shrtimer_exeevent_config	配置外部事件
shrtimer_exeevent_prescaler	配置外部事件的数字滤波分频
shrtimer_synccfg_struct_para_init	初始化shrtimer_synccfg_parameter_struct结构体变量
shrtimer_synchronization_config	配置SHRTIMER的同步输入/输出功能
shrtimer_faultcfg_struct_para_init	初始化shrtimer_faultcfg_parameter_struct结构体变量
shrtimer_fault_config	配置故障输入功能
shrtimer_fault_prescaler_config	配置故障输入数字滤波分频
shrtimer_fault_input_enable	故障输入使能
shrtimer_fault_input_disable	故障输入禁能
shrtimer_timers_dma_enable	使能Master_TIMER/Slave_TIMER DMA请求
shrtimer_timers_dma_disable	禁能Master_TIMER/Slave_TIMER DMA请求
shrtimer_dmamode_config	配置Master_TIMER/Slave_TIMER的DMA模式
shrtimer_bunchmode_struct_para_init	初始化shrtimer_bunchmode_parameter_struct结构体
shrtimer_bunchmode_config	配置SHRTIMER的突发模式
shrtimer_bunchmode_enable	使能突发模式
shrtimer_bunchmode_disable	禁能突发模式
shrtimer_bunchmode_flag_get	获取突发模式的运行标志
shrtimer_bunchmode_software_start	软件启动突发模式
shrtimer_slavetimer_capture_config	配置Slave_TIMER捕获源
shrtimer_slavetimer_capture_software	软件触发Slave_TIMER捕获
shrtimer_slavetimer_capture_value_read	读取捕获值
shrtimer_adctrigcfg_struct_para_init	初始化shrtimer_adctrigcfg_parameter_struct结构体变量
shrtimer_adc_trigger_config	配置ADC触发源和更新源

库函数名称	库函数描述
shrtimer_timers_flag_get	获取Master_TIMER/Slave_TIMER的标志
shrtimer_timers_flag_clear	清除Master_TIMER/Slave_TIMER的标志
shrtimer_common_flag_get	获取SHRTIMER通用标志
shrtimer_common_flag_clear	清除SHRTIMER通用标志
shrtimer_timers_interrupt_enable	使能Master_TIMER/Slave_TIMER中断
shrtimer_timers_interrupt_disable	禁能Master_TIMER/Slave_TIMER中断
shrtimer_timers_interrupt_flag_get	获取Master_TIMER/Slave_TIMER中断标志
shrtimer_timers_interrupt_flag_clear	清除Master_TIMER/Slave_TIMER中断标志
shrtimer_common_interrupt_enable	使能SHRTIMER通用中断
shrtimer_common_interrupt_disable	禁能SHRTIMER通用中断
shrtimer_common_interrupt_flag_get	获取SHRTIMER通用中断标志
shrtimer_common_interrupt_flag_clear	清除SHRTIMER通用中断标志

### 结构体 shrtimer\_baseinit\_parameter\_struct

表 3-483. 结构体 shrtimer\_baseinit\_parameter\_struct

成员名称	功能描述
period	周期值，最小值：3个tSHRTIMER_CK时钟，最大值：0xFFFF - (1个tSHRTIMER_CK)
repetitioncounter	重复计数值，0x00~0xFF
prescaler	预分频值
counter_mode	计数器运行模式

### 结构体 shrtimer\_timerinit\_parameter\_struct

表 3-484. 结构体 shrtimer\_timerinit\_parameter\_struct

成员名称	功能描述
half_mode	使能或禁能半波模式
start_sync	同步输入启动计数器
reset_sync	同步输入复位计数器
dac_trigger	DAC触发源
shadow	使能或者禁能影子寄存器
update_selection	更新事件源选择
cnt_bunch	突发模式下计数器时钟停止或者正常运行
repetition_update	重复事件产生更新事件

### 结构体 shrtimer\_timercfg\_parameter\_struct

表 3-485. 结构体 shrtimer\_timercfg\_parameter\_struct

成员名称	功能描述
balanced_mode	使能或禁能均衡模式
fault_enable	使能或禁能Slave_TIMER故障输入通道

fault_protect	使能或禁能故障保护功能
deadtime_enable	使能或禁能死区功能
delayed_idle	延时空闲模式
update_source	更新事件由其他定时器更新事件产生
cnt_reset	计数器复位源
reset_update	计数器复位产生更新事件

### 结构体 `shrtimer_comparecfg_parameter_struct`

表 3-486. 结构体 `shrtimer_comparecfg_parameter_struct`

成员名称	功能描述
compare_value	比较寄存器值，最小值：3个tSHRTIMER_CK时钟，最大值：0xFFFF - (1个tSHRTIMER_CK)
delayed_mode	配置比较延时模式（只有比较1和3寄存器有该功能）
timeout_value	超时值（在比较延时模式具有超时功能时该值有效）

### 结构体 `shrtimer_exeefilter_parameter_struct`

表 3-487. 结构体 `shrtimer_exeefilter_parameter_struct`

成员名称	功能描述
filter_mode	外部事件的滤波模式
memorized	外部事件滤波存储功能

### 结构体 `shrtimer_deadtimecfg_parameter_struct`

表 3-488. 结构体 `shrtimer_deadtimecfg_parameter_struct`

成员名称	功能描述
prescaler	死区时间发生器预分频
rising_value	死区上升沿数值，0x0000~0xFFFF
rising_sign	死区上升沿数值符号
rising_protect	死区上升数值值和符号保护
risingsign_protect	死区上升沿符号保护
falling_value	死区下降沿数值，0x0000~0xFFFF
falling_sign	死区下降沿数值符号
falling_protect	死区下降数值值和符号保护
fallingsign_protect	死区下降沿符号保护

### 结构体 `shrtimer_carriersignalcfg_parameter_struct`

表 3-489. 结构体 `shrtimer_carriersignalcfg_parameter_struct`

成员名称	功能描述
period	载波信号的周期tCSPRD，0x0~0xF， $tCSPRD = (period + 1) * 16 * tSHRTIMER\_CK$
duty_cycle	载波信号的占空比，0x0~0x7， $duty\_cycle = duty\_cycle/8$

first_pulse	第一个载波信号的脉冲宽度tCSFSTPW，0x0~0xF，tCSFSTPW = (first_pulse+1) * 16 * tSHRTIMER_CK
-------------	---

### 结构体 shrtimer\_synccfg\_parameter\_struct

表 3-490. 结构体 shrtimer\_synccfg\_parameter\_struct

成员名称	功能描述
input_source	同步输入信号源
output_source	同步输出信号源
output_polarity	同步输出信号极性（output_source为有效信号时，该配置生效）

### 结构体 shrtimer\_bunchmode\_parameter\_struct

表 3-491. 结构体 shrtimer\_bunchmode\_parameter\_struct

成员名称	功能描述
mode	突发模式运行模式（连续或者单脉冲）
clock_source	突发模式的计数时钟源
prescaler	突发模式的预分频（仅在clock_source为fSHRTIMER_CK该配置生效）
shadow	SHRTIMER_BMCMPV和SHRTIMER_BMCAR寄存器的影子寄存器使能
trigger	突发模式的启动触发事件
idle_duration	空闲时长，0x0000~0xFFFF
period	周期，0x0001~0xFFFF

### 结构体 shrtimer\_exeventcfg\_parameter\_struct

表 3-492. 结构体 shrtimer\_exeventcfg\_parameter\_struct

成员名称	功能描述
source	外部事件源
polarity	外部事件极性，（当edge为电平有效时该配置生效）
edge	外部事件的有效沿
digital_filter	外部事件的数字滤波，0x0~0xF

### 结构体 shrtimer\_faultcfg\_parameter\_struct

表 3-493. 结构体 shrtimer\_faultcfg\_parameter\_struct

成员名称	功能描述
source	故障输入源
polarity	故障输入极性
filter	故障输入数字滤波，0x0~0xF
control	使能或禁能故障输入
protect	保护故障输入配置

### 结构体 shrtimer\_adctrigcfg\_parameter\_struct

表 3-494. 结构体 shrtimer\_adctrigcfg\_parameter\_struct

成员名称	功能描述
update_source	SHRTIMER_ADCTRIGSy (y=0..3) 寄存器的更新源
trigger	ADC触发源

### 结构体 shrtimer\_channel\_outputcfg\_parameter\_struct

表 3-495. 结构体 shrtimer\_channel\_outputcfg\_parameter\_struct

成员名称	功能描述
polarity	通道输出极性
set_request	产生通道输出置位请求的事件
reset_request	产生通道输出复位请求的事件
idle_bunch	通道是否受突发模式控制
idle_state	通道输出的空闲状态
fault_state	故障时通道输出状态
carrier_mode	使能或禁能载波模式
deadtime_bunch	在突发模式进入IDLE状态前是否插入死区时间

### 函数 shrtimer\_deinit

函数shrtimer\_deinit描述见下表:

表 3-496. 函数 shrtimer\_deinit

函数名称	shrtimer_deinit
函数原型	void shrtimer_deinit(uint32_t shrtimer_periph);
功能描述	复位外设SHRTIMER
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset SHRTIMER */
shrtimer_deinit(SHRTIMER0);
```

**函数 shrtimer\_dll\_calibration\_start**

函数shrtimer\_dll\_calibration\_start描述见下表：

**表 3-497. 函数 shrtimer\_dll\_calibration\_start**

函数名称	shrtimer_dll_calibration_start
函数原型	void shrtimer_dll_calibration_start(uint32_t shrtimer_periph);
功能描述	配置和启动DLL校准
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
<b>输入参数{in}</b>	
calform	指定校准的方式
SHRTIMER_CALIBRATION_ONCE	只校准一次
SHRTIMER_CALIBRATION_1048576_PERIOD	周期性校准，校准周期为1048576 * tSHRTIMER_CK
SHRTIMER_CALIBRATION_131072_PERIOD	周期性校准，校准周期为131072 * tSHRTIMER_CK
SHRTIMER_CALIBRATION_16384_PERIOD	周期性校准，校准周期为16384 * tSHRTIMER_CK
SHRTIMER_CALIBRATION_2048_PERIOD	周期性校准，校准周期为2048 * tSHRTIMER_CK
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure and start DLL calibration */
shrtimer_dll_calibration_start(SHRTIMER0, SHRTIMER_CALIBRATION_ONCE);
```

**函数 shrtimer\_baseinit\_struct\_para\_init**

函数shrtimer\_baseinit\_struct\_para\_init描述见下表：

**表 3-498. 函数 shrtimer\_baseinit\_struct\_para\_init**

函数名称	shrtimer_baseinit_struct_para_init
函数原型	void shrtimer_baseinit_struct_para_init(shrtimer_baseinit_parameter_struct* baseinit);
功能描述	初始化shrtimer_baseinit_parameter_struct结构体变量
先决条件	-

被调用函数	-
输入参数{in}	
baseinit	shrtimer_baseinit_parameter_struct结构体指针，该结构体成员变量参考 <a href="#">表3-483. 结构体shrtimer_baseinit_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize SHRTIMER time base parameters struct with the default value */
```

```
shrtimer_baseinit_parameter_struct baseinit;
```

```
shrtimer_baseinit_struct_para_init(&baseinit);
```

### 函数 shrtimer\_timers\_base\_init

函数shrtimer\_timers\_base\_init描述见下表：

表 3-499. 函数 shrtimer\_timers\_base\_init

函数名称	shrtimer_timers_base_init
函数原型	void shrtimer_timers_base_init(uint32_t shrtimer_periph, uint32_t timer_id, shrtimer_baseinit_parameter_struct* baseinit);
功能描述	初始化Master_TIMER/Slave_TIMER时基
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	Master_TIMER和Slave_TIMER索引
SHRTIMER_MASTER_TIMER	选择Master_TIMER
SHRTIMER_SLAVE_TIMERx	选择Slave_TIMER (x=0..4)
输入参数{in}	
baseinit	shrtimer_baseinit_parameter_struct结构体指针，该结构体成员变量参考 <a href="#">表3-483. 结构体shrtimer_baseinit_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize Master_TIMER and Slave_TIMER timerbase */
```

```
shrtimer_baseinit_parameter_struct baseinit_para;
```

```
shrtimer_baseinit_struct_para_init(&baseinit_para);
```

```
baseinit_para.period = 384;
```

```
baseinit_para.prescaler = SHRTIMER_PRESCALER_MUL64;
```

```
baseinit_para.repetitioncounter = 0;
```

```
baseinit_para.counter_mode = SHRTIMER_COUNTER_MODE_CONTINUOUS;
```

```
shrtimer_timers_base_init(SHRTIMER0, SHRTIMER_SLAVE_TIMER0, &baseinit_para);
```

### 函数 shrtimer\_timers\_counter\_enable

函数shrtimer\_timers\_counter\_enable描述见下表：

表 3-500. 函数 shrtimer\_timers\_counter\_enable

函数名称	shrtimer_timers_counter_enable
函数原型	void shrtimer_timers_counter_enable(uint32_t shrtimer_periph, uint32_t cntid);
功能描述	使能定时器的计数器
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
cntid	指定计数器
SHRTIMER_MT_COUNTER	Master_TIMER的计数器
SHRTIMER_STx_COUNTER(x=0..4)	Slave_TIMERx(x=0..4)的计数器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable a counter */
```

```
void shrtimer_timers_counter_enable(SHRTIMER0, SHRTIMER_MT_COUNTER);
```

### 函数 shrtimer\_timers\_counter\_disable

函数shrtimer\_timers\_counter\_disable描述见下表：

表 3-501. 函数 `shrtimer_timers_counter_enable`

函数名称	<code>shrtimer_timers_counter_disable</code>
函数原型	<code>void shrtimer_timers_counter_disable(uint32_t shrtimer_periph, uint32_t cntid);</code>
功能描述	禁能定时器的计数器
先决条件	-
被调用函数	-
输入参数{in}	
<code>shrtimer_periph</code>	SHRTIMER外设
<code>SHRTIMERx(x=0)</code>	选择一个SHRTIMER外设
输入参数{in}	
<code>cntid</code>	指定计数器
<code>SHRTIMER_MT_COUNTER</code>	Master_TIMER的计数器
<code>SHRTIMER_STx_COUNTER(x=0..4)</code>	Slave_TIMERx(x=0..4)的计数器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable a counter */
```

```
shrtimer_timers_counter_disable(SHRTIMER0, SHRTIMER_MT_COUNTER);
```

### 函数 `shrtimer_timers_update_event_enable`

函数`shrtimer_timers_update_event_enable`描述见下表:

表 3-502. 函数 `shrtimer_timers_update_event_enable`

函数名称	<code>shrtimer_timers_update_event_enable</code>
函数原型	<code>void shrtimer_timers_update_event_enable(uint32_t shrtimer_periph, uint32_t timer_id);</code>
功能描述	使能Master_TIMER/Slave_TIMER更新事件
先决条件	-
被调用函数	-
输入参数{in}	
<code>shrtimer_periph</code>	SHRTIMER外设
<code>SHRTIMERx(x=0)</code>	选择一个SHRTIMER外设
输入参数{in}	
<code>timer_id</code>	选择SHRTIMER中的一个定时器
<code>SHRTIMER_MASTER_TIMER</code>	选择Master_TIMER

SHRTIMER_SLAVE_TIMERx(x=0..4)	选择Slave_TIMERx(x=0..4)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the Master_TIMER or Slave_TIMER update event */
```

```
shrtimer_timers_update_event_enable(SHRTIMER0, SHRTIMER_MASTER_TIMER);
```

### 函数 shrtimer\_timers\_update\_event\_disable

函数shrtimer\_timers\_update\_event\_disable描述见下表：

表 3-503. 函数 shrtimer\_timers\_update\_event\_enable

函数名称	shrtimer_timers_update_event_disable
函数原型	void shrtimer_timers_update_event_disable(uint32_t shrtimer_periph, uint32_t timer_id);
功能描述	禁能Master_TIMER/Slave_TIMER更新事件
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_MASTER_TIMER	选择Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	选择Slave_TIMERx(x=0..4)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the Master_TIMER or Slave_TIMER update event */
```

```
shrtimer_timers_update_event_disable(SHRTIMER0, SHRTIMER_MASTER_TIMER);
```

### 函数 shrtimer\_software\_update

函数shrtimer\_software\_update描述见下表：

表 3-504. 函数 `shrtimer_software_update`

函数名称	<code>shrtimer_software_update</code>
函数原型	<code>void shrtimer_software_update(uint32_t shrtimer_periph, uint32_t timersrc);</code>
功能描述	软件触发Master_TIMER与Slave_TIMER更新事件
先决条件	-
被调用函数	-
输入参数{in}	
<code>shrtimer_periph</code>	SHRTIMER外设
<code>SHRTIMERx(x=0)</code>	选择一个SHRTIMER外设
输入参数{in}	
<code>timersrc</code>	选择软件触发的定时器
<code>SHRTIMER_UPDATE_SW_MT</code>	软件触发Master_TIMER
<code>SHRTIMER_UPDATE_SW_STx(x=0..4)</code>	软件触发Slave_TIMERx (x=0..4)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* update the Master_TIMER or Slave_TIMER by software */
```

```
void shrtimer_software_update(SHRTIMER0, SHRTIMER_UPDATE_SW_MT);
```

### 函数 `shrtimer_software_counter_reset`

函数`shrtimer_software_counter_reset`描述见下表：

表 3-505. 函数 `shrtimer_software_counter_reset`

函数名称	<code>shrtimer_software_counter_reset</code>
函数原型	<code>void shrtimer_software_counter_reset(uint32_t shrtimer_periph, uint32_t timerrst);</code>
功能描述	软件复位Master_TIMER/Slave_TIMER计数器
先决条件	-
被调用函数	-
输入参数{in}	
<code>shrtimer_periph</code>	SHRTIMER外设
<code>SHRTIMERx(x=0)</code>	选择一个SHRTIMER外设
输入参数{in}	
<code>timersrc</code>	选择软件复位的定时器
<code>SHRTIMER_COUNTER_RESET_SW_MT</code>	软件复位Master_TIMER
<code>SHRTIMER_COUNTER_RESET_SW_STx(x=0..4)</code>	软件复位Slave_TIMERx (x=0..4)

<code>_RESET_SW_STx</code> ( $x=0..4$ )	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* triggers the Master_TIMER or Slave_TIMER registers update by software */
```

```
void                                shrtimer_software_counter_reset(SHRTIMER0,
SHRTIMER_COUNTER_RESET_SW_MT);
```

### 函数 shrtimer\_timerinit\_struct\_para\_init

函数shrtimer\_timerinit\_struct\_para\_init描述见下表：

表 3-506. 函数 shrtimer\_timerinit\_struct\_para\_init

函数名称	shrtimer_timerinit_struct_para_init
函数原型	void shrtimer_timerinit_struct_para_init(shrtimer_timerinit_parameter_struct* timerinit);
功能描述	初始化shrtimer_timerinit_parameter_struct结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
timerinit	shrtimer_timerinit_parameter_struct结构体指针，结构体成员变量参考 <a href="#">表3-484. 结构体shrtimer_timerinit_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize waveform mode initialization parameters struct with a default value */
```

```
shrtimer_timerinit_parameter_struct timerinit_para;
```

```
shrtimer_timerinit_struct_para_init(&timerinit_para);
```

### 函数 shrtimer\_timers\_waveform\_init

函数shrtimer\_timers\_waveform\_init描述见下表：

表 3-507. 函数 shrtimer\_timers\_waveform\_init

函数名称	shrtimer_timers_waveform_init
函数原型	void shrtimer_timers_waveform_init(uint32_t shrtimer_periph, uint32_t

	timer_id, shrtimer_timerinit_parameter_struct* timerinitpara);
功能描述	定时器波形初始化
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_MASTER_TIMER	选择Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	选择Slave_TIMERx(x=0..4)
输入参数{in}	
timerinit	shrtimer_timerinit_parameter_struct结构体指针, 结构体成员变量参考 <a href="#">表3-484. 结构体shrtimer_timerinit_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* initialize a timer to work in waveform mode */

shrtimer_timerinit_parameter_struct timerinit_para;

shrtimer_timerinit_struct_para_init(&timerinit_para);

timerinit_para.cnt_bunch = SHRTIMER_TIMERBUNCHMODE_MAINTAINCLOCK;

timerinit_para.DAC_trigger = SHRTIMER_DAC_TRIGGER_NONE;

timerinit_para.half_mode = SHRTIMER_HALFMODE_DISABLED;

timerinit_para.repetition_update = SHRTIMER_UPDATEONREPETITION_DISABLED;

timerinit_para.reset_sync = SHRTIMER_SYNCRESET_DISABLED;

timerinit_para.shadow = SHRTIMER_SHADOW_DISABLED;

timerinit_para.start_sync = SHRTIMER_SYNISTART_DISABLED;

timerinit_para.update_selection = SHRTIMER_MT_ST_UPDATE_SELECTION_INDEPENDENT;

shrtimer_timers_waveform_init(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,
&timerinit_para);

```

## 函数 shrtimer\_timercfg\_struct\_para\_init

函数shrtimer\_timercfg\_struct\_para\_init描述见下表：

**表 3-508. 函数 shrtimer\_timercfg\_struct\_para\_init**

函数名称	shrtimer_timercfg_struct_para_init
函数原型	void shrtimer_timercfg_struct_para_init(shrtimer_timercfg_parameter_struct* timercfg);
功能描述	初始化shrtimer_timercfg_parameter_struct结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
timercfg	shrtimer_timercfg_parameter_struct结构体指针，结构体成员参考 <a href="#">表3-485. 结构体shrtimer_timercfg_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize Slave_TIMER general behavior configuration struct with a default value */
shrtimer_timercfg_parameter_struct timercfg_para;
shrtimer_timercfg_struct_para_init(timercfg_para);
```

## 函数 shrtimer\_slavetimer\_waveform\_config

函数shrtimer\_slavetimer\_waveform\_config描述见下表：

**表 3-509. 函数 shrtimer\_slavetimer\_waveform\_config**

函数名称	shrtimer_slavetimer_waveform_config
函数原型	void shrtimer_slavetimer_waveform_config(uint32_t shrtimer_periph, uint32_t timer_id, shrtimer_timercfg_parameter_struct * timercfg);
功能描述	配置Slave_TIMER的波形
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	Master_TIMER和Slave_TIMER索引
SHRTIMER_MASTER_TIMER	选择Master_TIMER

SHRTIMER_SLAVE_TIMERx(x=0..4)	选择Slave_TIMER (x=0..4)
输入参数{in}	
timercfg	shrtimer_timercfg_parameter_struct结构体指针，结构体成员参考 <a href="#">表3-485. 结构体shrtimer_timercfg_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize Slave_TIMER general behavior configuration struct with a default value */
shrtimer_timercfg_parameter_struct timercfg_para;
shrtimer_timercfg_struct_para_init(&timercfg_para);
timercfg_para.balanced_mode = SHRTIMER_STXBALANCEDMODE_DISABLED;
timercfg_para.cnt_reset = SHRTIMER_STXCNT_RESET_NONE;
timercfg_para.deadtime_enable = SHRTIMER_STXDEADTIME_DISABLED;
timercfg_para.delayed_idle = SHRTIMER_STXDELAYED_IDLE_DISABLED;
timercfg_para.fault_enable = SHRTIMER_STXFAULTENABLE_NONE;
timercfg_para.fault_protect = SHRTIMER_STXFAULT_PROTECT_READWRITE;
timercfg_para.reset_update = SHRTIMER_STXUPDATEONRESET_DISABLED;
timercfg_para.update_source = SHRTIMER_STXUPDATETRIGGER_NONE;
shrtimer_slavetimer_waveform_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,
&timercfg_para);
```

### 函数 shrtimer\_comparecfg\_struct\_para\_init

函数shrtimer\_comparecfg\_struct\_para\_init描述见下表：

表 3-510. 函数 shrtimer\_comparecfg\_struct\_para\_init

函数名称	shrtimer_comparecfg_struct_para_init
函数原型	void shrtimer_comparecfg_struct_para_init(shrtimer_comparecfg_parameter_struct* comparecfg);
功能描述	初始化shrtimer_comparecfg_parameter_struct结构体
先决条件	-
被调用函数	-
输入参数{in}	
comparecfg	shrtimer_comparecfg_parameter_struct结构体指针，结构体成员参考

	<a href="#">表3-486. 结构体shrtimer_comparecfg_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize compare unit configuration struct with a default value */
```

```
shrtimer_comparecfg_parameter_struct comparecfg_para;
```

```
shrtimer_comparecfg_struct_para_init(comparecfg_para);
```

### 函数 shrtimer\_slavetimer\_waveform\_compare\_config

函数shrtimer\_slavetimer\_waveform\_compare\_config描述见下表：

**表 3-511. 函数 shrtimer\_slavetimer\_waveform\_compare\_config**

函数名称	shrtimer_slavetimer_waveform_compare_config
函数原型	void shrtimer_slavetimer_waveform_compare_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t comparex, shrtimer_comparecfg_parameter_struct* cmpcfg);
功能描述	配置Slave_TIMER波形的比较功能
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_MASTER_TIMER	选择Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	选择Slave_TIMERx(x=0..4)
输入参数{in}	
comparex	选择一个比较单元
SHRTIMER_COMPAREy (y=0..4)	比较单元y(y=0..4)
输入参数{in}	
comparecfg	shrtimer_comparecfg_parameter_struct结构体指针，结构体成员参考 <a href="#">表3-486. 结构体shrtimer_comparecfg_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the compare unit of a Slave_TIMER which work in waveform mode */

shrtimer_comparecfg_parameter_struct comparecfg_para;

shrtimer_comparecfg_struct_para_init(&comparecfg_para);

comparecfg_para.compare_value = 192;

shrtimer_slavetimer_waveform_compare_config(SHRTIMER0,
SHRTIMER_SLAVE_TIMER0, SHRTIMER_COMPARE0, &comparecfg_para);
```

### 函数 shrtimer\_channel\_outputcfg\_struct\_para\_init

函数shrtimer\_channel\_outputcfg\_struct\_para\_init描述见下表：

表 3-512. 函数 shrtimer\_channel\_outputcfg\_struct\_para\_init

函数名称	shrtimer_channel_outputcfg_struct_para_init
函数原型	void shrtimer_channel_outputcfg_struct_para_init(shrtimer_channel_outputcfg_parameter_struct * channelcfg);
功能描述	初始化shrtimer_channel_outputcfg_parameter_struct结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
channelcfg	shrtimer_channel_outputcfg_parameter_struct结构体指针，结构体成员参考 <a href="#">表3-495. 结构体shrtimer_channel_outputcfg_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize channel output configuration struct with a default value */

shrtimer_channel_outputcfg_parameter_struct outcfg_para;

shrtimer_channel_outputcfg_struct_para_init(&outcfg_para);
```

### 函数 shrtimer\_slavetimer\_waveform\_channel\_config

函数shrtimer\_slavetimer\_waveform\_channel\_config描述见下表：

表 3-513. 函数 shrtimer\_slavetimer\_waveform\_channel\_config

函数名称	shrtimer_slavetimer_waveform_channel_config
函数原型	void shrtimer_slavetimer_waveform_channel_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t channel,

	shrtimer_channel_outputcfg_parameter_struct * channelcfg;
功能描述	Slave_TIMER波形的通道配置
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_MASTER_TIMER	选择Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	选择Slave_TIMERx(x=0..4)
输入参数{in}	
channel	SHRTIMER通道选择
SHRTIMER_STx_CHy(x=0..4,y=0,1)	选择一个通道
channelcfg	
channelcfg	shrtimer_channel_outputcfg_parameter_struct结构体指针，结构体成员参考 <a href="#">表3-495. 结构体shrtimer_channel_outputcfg_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* configure the channel of a Slave_TIMER work in waveform mode */

shrtimer_channel_outputcfg_parameter_struct outcfg_para;

shrtimer_channel_outputcfg_struct_para_init(&outcfg_para);

outcfg_para.carrier_mode = SHRTIMER_CHANNEL_CARRIER_DISABLED;

outcfg_para.deadtime_bunch = SHRTIMER_CHANNEL_BUNCH_ENTRY_REGULAR;

outcfg_para.fault_state = SHRTIMER_CHANNEL_FAULTSTATE_NONE;

outcfg_para.idle_bunch = SHRTIMER_CHANNEL_BUNCH_IDLE_DISABLE;

outcfg_para.idle_state = SHRTIMER_CHANNEL_IDLESTATE_INACTIVE;

outcfg_para.polarity = SHRTIMER_CHANNEL_POLARITY_HIGH;

outcfg_para.reset_request = SHRTIMER_CHANNEL_RESET_CMP1;

outcfg_para.set_request = SHRTIMER_CHANNEL_SET_CMP0;

shrtimer_slavetimer_waveform_channel_config(SHRTIMER0,

```

SHRTIMER\_SLAVE\_TIMER0, SHRTIMER\_ST0\_CH0, &outcfg\_para);

### 函数 shrtimer\_slavetimer\_waveform\_channel\_software\_request

函数shrtimer\_slavetimer\_waveform\_channel\_software\_request描述见下表:

表 3-514. 函数 shrtimer\_slavetimer\_waveform\_channel\_software\_request

函数名称	shrtimer_slavetimer_waveform_channel_software_request
函数原型	void shrtimer_slavetimer_waveform_channel_software_request(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t channel, uint32_t request)
功能描述	软件产生通道置位或复位请求
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_MASTER_TIMER	选择Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	选择Slave_TIMERx(x=0..4)
输入参数{in}	
channel	SHRTIMER通道选择
SHRTIMER_STx_CHy(x=0..4,y=0,1)	选择一个通道
输入参数{in}	
request	置位请求
SHRTIMER_CHANNEL_SOFTWARE_SET	软件产生置位请求
SHRTIMER_CHANNEL_SOFTWARE_RESET	软件产生复位请求
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* software generates channel "set request" or "reset request" */
shrtimer_slavetimer_waveform_channel_software_request (SHRTIMER0,
SHRTIMER_SLAVE_TIMER0, SHRTIMER_ST0_CH0,
SHRTIMER_CHANNEL_SOFTWARE_SET);
```

## 函数 shrtimer\_slavetimer\_waveform\_channel\_output\_level\_get

函数shrtimer\_slavetimer\_waveform\_channel\_output\_level\_get描述见下表：

**表 3-515. 函数 shrtimer\_slavetimer\_waveform\_channel\_output\_level\_get**

函数名称	shrtimer_slavetimer_waveform_channel_output_level_get
函数原型	uint32_t shrtimer_slavetimer_waveform_channel_output_level_get(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t channel);
功能描述	获取通道的输出电平
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_MASTER_TIMER	选择Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	选择Slave_TIMERx(x=0..4)
输入参数{in}	
channel	SHRTIMER通道选择
SHRTIMER_STx_CHy(x=0..4,y=0,1)	选择一个通道
输出参数{out}	
-	-
返回值	
uint32_t	通道输出电平

例如：

```
/* get Slave_TIMER channel output level */
```

```
uint32_t output_level;
```

```
output_level = shrtimer_slavetimer_waveform_channel_output_level_get(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t channel)
```

## 函数 shrtimer\_slavetimer\_waveform\_channel\_state\_get

函数shrtimer\_slavetimer\_waveform\_channel\_state\_get描述见下表：

**表 3-516. 函数 shrtimer\_slavetimer\_waveform\_channel\_state\_get**

函数名称	shrtimer_slavetimer_waveform_channel_state_get
函数原型	uint32_t shrtimer_slavetimer_waveform_channel_state_get(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t channel)

功能描述	获取通道的运行状态
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_MASTER_TIMER	选择Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	选择Slave_TIMERx(x=0..4)
输入参数{in}	
channel	SHRTIMER通道选择
SHRTIMER_STx_CHy(x=0..4,y=0,1)	选择一个通道
输出参数{out}	
-	-
返回值	
uint32_t	SHRTIMER_CHANNEL_STATE_IDLE, SHRTIMER_CHANNEL_STATE_RUN, SHRTIMER_CHANNEL_STATE_FAULT

例如:

```
/* get Slave_TIMER channel run state */
```

```
uint32_t output_state;
```

```
output_state = shrtimer_slavetimer_waveform_channel_state_get (SHRTIMER0,  
SHRTIMER_SLAVE_TIMER0, SHRTIMER_ST0_CH0);
```

### 函数 shrtimer\_deadtimercfg\_struct\_para\_init

函数shrtimer\_deadtimercfg\_struct\_para\_init描述见下表:

表 3-517. 函数 shrtimer\_channel\_outputcfg\_struct\_para\_init

函数名称	shrtimer_deadtimercfg_struct_para_init
函数原型	void shrtimer_deadtimercfg_struct_para_init(shrtimer_deadtimercfg_parameter_struct * dtcfg);
功能描述	初始化shrtimer_deadtimercfg_struct结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
dtcfg	shrtimer_deadtimercfg_struct结构体指针, 结构体成员参考

	<a href="#">表3-488. 结构体shrtimer_deadtimecfg_parameter_struct</a> 结构体 shrtimer_deadtimecfg_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize dead time configuration struct with a default value */
```

```
shrtimer_deadtimecfg_parameter_struct deadtimecfg_para;
```

```
shrtimer_deadtimecfg_struct_para_init(&deadtimecfg_para);
```

### 函数 shrtimer\_slavetimer\_deadtime\_config

函数shrtimer\_slavetimer\_deadtime\_config描述见下表：

表 3-518. 函数 shrtimer\_slavetimer\_waveform\_channel\_software\_request

函数名称	shrtimer_slavetimer_deadtime_config
函数原型	void shrtimer_slavetimer_deadtime_config(uint32_t shrtimer_periph, uint32_t timer_id, shrtimer_deadtimecfg_parameter_struct* dtcfg)
功能描述	配置Slave_TIMER的死区时间
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_SLAVE_TIMERx(x=0..4)	选择Slave_TIMERx(x=0..4)
输入参数{in}	
dtcfg	shrtimer_deadtimecfg_parameter_struct结构体指针，结构体成员参考 <a href="#">表3-488. 结构体shrtimer_deadtimecfg_parameter_struct</a> 结构体 shrtimer_deadtimecfg_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the dead time for Slave_TIMER */
```

```
shrtimer_deadtimecfg_parameter_struct deadtimecfg_para;
```

```

shrtimer_deadtimercfg_struct_para_init(&deadtimercfg_para);

deadtimercfg_para.fallingsign_protect                                =
SHRTIMER_DEADTIME_FALLINGSIGN_PROTECT_DISABLE;

deadtimercfg_para.falling_protect                                    =
SHRTIMER_DEADTIME_FALLING_PROTECT_DISABLE;

deadtimercfg_para.falling_sign = SHRTIMER_DEADTIME_FALLINGSIGN_POSITIVE;

deadtimercfg_para.falling_value = 0x0040;

deadtimercfg_para.prescaler = SHRTIMER_DEADTIME_PRESCALER_MUL8;

deadtimercfg_para.risingsign_protect                                =
SHRTIMER_DEADTIME_RISINGSIGN_PROTECT_DISABLE;

deadtimercfg_para.rising_protect = SHRTIMER_DEADTIME_RISING_PROTECT_DISABLE;

deadtimercfg_para.rising_sign = SHRTIMER_DEADTIME_RISINGSIGN_POSITIVE;

deadtimercfg_para.rising_value = 0x0040;

shrtimer_slavetimer_deadtime_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,
&deadtimercfg_para);

```

### 函数 shrtimer\_carriersignalcfg\_struct\_para\_init

函数shrtimer\_carriersignalcfg\_struct\_para\_init描述见下表:

**表 3-519. 函数 shrtimer\_channel\_outputcfg\_struct\_para\_init**

函数名称	shrtimer_carriersignalcfg_struct_para_init
函数原型	void shrtimer_carriersignalcfg_struct_para_init(shrtimer_carriersignalcfg_parameter_struct* carriercfg);
功能描述	初始化shrtimer_carriersignalcfg_parameter_struct结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
carriercfg	shrtimer_carriersignalcfg_parameter_struct结构体指针, 结构体成员参考 <a href="#">表3-489. 结构体shrtimer_carriersignalcfg_parameter_struct</a> 结构体shrtimer_carriersignalcfg_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize carrier signal configuration struct with a default value */
```

```
shrtimer_carriersignalcfg_parameter_struct carriercfg;
```

```
shrtimer_carriersignalcfg_struct_para_init(&carriercfg);
```

### 函数 shrtimer\_slavetimer\_carriersignal\_config

函数shrtimer\_slavetimer\_carriersignal\_config描述见下表：

表 3-520. 函数 shrtimer\_slavetimer\_carriersignal\_config

函数名称	shrtimer_slavetimer_carriersignal_config
函数原型	void shrtimer_slavetimer_carriersignal_config(uint32_t shrtimer_periph, uint32_t timer_id, shrtimer_carriersignalcfg_parameter_struct* carriercfg);
功能描述	配置Slave_TIMER的载波功能
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_SLAVE_TIMERx(x=0..4)	选择Slave_TIMERx(x=0..4)
输入参数{in}	
carriercfg	shrtimer_carriersignalcfg_parameter_struct结构体指针，结构体成员参考 <a href="#">表3-489. 结构体shrtimer_carriersignalcfg_parameter_struct</a> 结构体 shrtimer_carriersignalcfg_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the carrier signal mode for Slave_TIMER */
```

```
shrtimer_carriersignalcfg_parameter_struct carriercfg;
```

```
shrtimer_carriersignalcfg_struct_para_init(&carriercfg);
```

```
shrtimer_slavetimer_carriersignal_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0, &carriercfg);
```

### 函数 shrtimer\_output\_channel\_enable

函数shrtimer\_output\_channel\_enable描述见下表：

表 3-521. 函数 shrtimer\_output\_channel\_enable

函数名称	shrtimer_output_channel_enable
------	--------------------------------

函数原型	void shrtimer_output_channel_enable(uint32_t shrtimer_periph, uint32_t chid);
功能描述	使能通道输出
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
ch_id	SHRTIMER通道
SHRTIMER_STx_CHy( x=0..4;y=0,1)	选择一个通道
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable a output channel */
```

```
void shrtimer_output_channel_enable(SHRTIMER0, SHRTIMER_ST0_CH0);
```

### 函数 shrtimer\_output\_channel\_disable

函数shrtimer\_output\_channel\_disable描述见下表：

表 3-522. 函数 shrtimer\_output\_channel\_disable

函数名称	shrtimer_output_channel_disable
函数原型	void shrtimer_output_channel_disable(uint32_t shrtimer_periph, uint32_t chid);
功能描述	禁能一个输出通道
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
ch_id	SHRTIMER通道
SHRTIMER_STx_CHy( x=0..4;y=0,1)	选择一个通道
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable a output channel */
```

```
void shrtimer_output_channel_disable(SHRTIMER0, SHRTIMER_ST0_CH0);
```

### 函数 shrtimer\_slavetimer\_compare\_value\_config

函数shrtimer\_slavetimer\_compare\_value\_config描述见下表：

**表 3-523. 函数 shrtimer\_slavetimer\_waveform\_compare\_config**

函数名称	shrtimer_slavetimer_compare_value_config
函数原型	void shrtimer_slavetimer_compare_value_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t comparex, uint32_t cmpvalue);
功能描述	配置Slave_TIMER的比较寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_SLAVE_TIMERx(x=0..4)	选择Slave_TIMERx(x=0..4)
输入参数{in}	
comparex	比较单元选择
SHRTIMER_COMPAREy (y=0..4)	选择比较单元y(y=0..4)
输入参数{in}	
cmpvalue	比较值，最小值：3个tSHRTIMER_CK，最大值：0xFFFF - (1个tSHRTIMER_CK)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the compare value in Slave_TIMER */
```

```
shrtimer_slavetimer_compare_value_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0, SHRTIMER_COMPARE0, 3);
```

### 函数 shrtimer\_slavetimer\_compare\_value\_get

函数shrtimer\_slavetimer\_compare\_value\_get描述见下表：

表 3-524. 函数 `shrtimer_slavetimer_compare_value_get`

函数名称	<code>shrtimer_slavetimer_compare_value_get</code>
函数原型	<code>uint32_t shrtimer_slavetimer_compare_value_get(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t comparex)</code>
功能描述	获取Slave_TIMER的比较寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
<code>shrtimer_periph</code>	SHRTIMER外设
<code>SHRTIMERx(x=0)</code>	选择一个SHRTIMER外设
输入参数{in}	
<code>timer_id</code>	选择SHRTIMER中的一个定时器
<code>SHRTIMER_SLAVE_TIMERx(x=0..4)</code>	选择Slave_TIMERx(x=0..4)
输入参数{in}	
<code>comparex</code>	比较单元选择
<code>SHRTIMER_COMPAREy (y=0..4)</code>	选择比较单元y(y=0..4)
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	比较值

例如:

```
/* get the compare value in Slave_TIMER */
```

```
uint32_t cmpvalue;
```

```
cmpvalue = shrtimer_slavetimer_compare_value_get (SHRTIMER0,
SHRTIMER_SLAVE_TIMER0, SHRTIMER_COMPARE0);
```

### 函数 `shrtimer_mastertimer_compare_value_config`

函数`shrtimer_mastertimer_compare_value_config`描述见下表:

表 3-525. 函数 `shrtimer_mastertimer_compare_value_config`

函数名称	<code>shrtimer_mastertimer_compare_value_config</code>
函数原型	<code>void shrtimer_mastertimer_compare_value_config(uint32_t shrtimer_periph, uint32_t comparex, uint32_t cmpvalue);</code>
功能描述	配置主定时器的比较寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
<code>shrtimer_periph</code>	SHRTIMER外设
<code>SHRTIMERx(x=0)</code>	选择一个SHRTIMER外设

输入参数{in}	
<b>comparex</b>	比较单元选择
<i>SHRTIMER_COMPAR</i> <i>Ey (y=0..4)</i>	选择比较单元y(y=0..4)
输入参数{in}	
<b>cmpvalue</b>	比较值，最小值：3个tSHRTIMER_CK，最大值：0xFFFF - (1个tSHRTIMER_CK)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the compare value in Master_TIMER */
```

```
shrtimer_mastertimer_compare_value_config(SHRTIMER0, SHRTIMER_COMPARE0, 3);
```

### 函数 shrtimer\_mastertimer\_compare\_value\_get

函数shrtimer\_mastertimer\_compare\_value\_get描述见下表：

表 3-526. 函数 shrtimer\_slavetimer\_compare\_value\_get

函数名称	shrtimer_mastertimer_compare_value_get
函数原型	uint32_t shrtimer_mastertimer_compare_value_get(uint32_t shrtimer_periph, uint32_t comparex);
功能描述	获取主定时器的比较寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
<b>shrtimer_periph</b>	SHRTIMER外设
<i>SHRTIMERx(x=0)</i>	选择一个SHRTIMER外设
输入参数{in}	
<b>comparex</b>	比较单元选择
<i>SHRTIMER_COMPAR</i> <i>Ey (y=0..4)</i>	选择比较单元y(y=0..4)
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	比较值

例如：

```
/* get the compare value in Master_TIMER */
```

```
uint32_t cmpvalue;
```

```
cmpvalue = shrtimer_mastertimer_compare_value_get(SHRTIMER0,
SHRTIMER_COMPARE0)
```

### 函数 shrtimer\_timers\_counter\_value\_config

函数shrtimer\_timers\_counter\_value\_config描述见下表：

表 3-527. 函数 shrtimer\_timers\_counter\_value\_config

函数名称	shrtimer_timers_counter_value_config
函数原型	void shrtimer_timers_counter_value_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t cntvalue);
功能描述	配置Master_TIMER/Slave_TIMER的计数寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_MASTER_TIMER	选择Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	选择Slave_TIMERx(x=0..4)
输入参数{in}	
cntvalue	计数器值，范围从0到0xffff
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the counter value in Master_TIMER and Slave_TIMER */
```

```
shrtimer_timers_counter_value_config(SHRTIMER0, SHRTIMER_MASTER_TIMER, 100);
```

### 函数 shrtimer\_timers\_counter\_value\_get

函数shrtimer\_timers\_counter\_value\_get描述见下表：

表 3-528. 函数 shrtimer\_timers\_counter\_value\_get

函数名称	shrtimer_timers_counter_value_get
函数原型	uint32_t shrtimer_timers_counter_value_get(uint32_t shrtimer_periph, uint32_t timer_id);
功能描述	获取Master_TIMER/Slave_TIMER的计数寄存器值
先决条件	-

被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_MASTER_TIMER	选择Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	选择Slave_TIMERx(x=0..4)
输出参数{out}	
-	-
返回值	
uint32_t	计数器值

例如:

```
/* get the counter value in Master_TIMER and Slave_TIMER */
```

```
uint32_t value;
```

```
value = shrtimer_timers_counter_value_get(SHRTIMER0, SHRTIMER_MASTER_TIMER);
```

### 函数 shrtimer\_timers\_autoreload\_value\_config

函数shrtimer\_timers\_autoreload\_value\_config描述见下表:

表 3-529. 函数 shrtimer\_timers\_autoreload\_value\_config

函数名称	shrtimer_timers_autoreload_value_config
函数原型	void shrtimer_timers_autoreload_value_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t carlvalue);
功能描述	配置Master_TIMER/Slave_TIMER的自动重载寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_MASTER_TIMER	选择Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	选择Slave_TIMERx(x=0..4)
输入参数{in}	
carlvalue	自动重载值, 范围从0到0xffff
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* configure the counter auto reload value in Master_TIMER and Slave_TIMER */
shrtimer_timers_autoreload_value_config(SHRTIMER0, SHRTIMER_MASTER_TIMER,
100);
```

### 函数 shrtimer\_timers\_autoreload\_value\_get

函数shrtimer\_timers\_autoreload\_value\_get描述见下表：

表 3-530. 函数 shrtimer\_timers\_autoreload\_value\_get

函数名称	shrtimer_timers_autoreload_value_get
函数原型	uint32_t shrtimer_timers_autoreload_value_get(uint32_t shrtimer_periph, uint32_t timer_id)
功能描述	获取Master_TIMER/Slave_TIMER的自动重载寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_MASTER_TIMER	选择Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	选择Slave_TIMERx(x=0..4)
输出参数{out}	
-	-
返回值	
uint32_t	自动重载值

例如：

```
/* get the counter auto reload value in Master_TIMER and Slave_TIMER */
uint32_t value;

value = shrtimer_timers_autoreload_value_get(SHRTIMER0,
SHRTIMER_MASTER_TIMER);
```

### 函数 shrtimer\_timers\_repetition\_value\_config

函数shrtimer\_timers\_repetition\_value\_config描述见下表：

表 3-531. 函数 `shrtimer_timers_repetition_value_config`

函数名称	<code>shrtimer_timers_repetition_value_config</code>
函数原型	<code>void shrtimer_timers_repetition_value_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t replvalue);</code>
功能描述	配置Master_TIMER/Slave_TIMER的重复计数寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
<code>shrtimer_periph</code>	SHRTIMER外设
<code>SHRTIMERx(x=0)</code>	选择一个SHRTIMER外设
输入参数{in}	
<code>timer_id</code>	选择SHRTIMER中的一个定时器
<code>SHRTIMER_MASTER_TIMER</code>	选择Master_TIMER
<code>SHRTIMER_SLAVE_TIMERx(x=0..4)</code>	选择Slave_TIMERx(x=0..4)
输入参数{in}	
<code>replvalue</code>	重复计数值，范围从0到0xff
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the counter repetition value in Master_TIMER and Slave_TIMER */
```

```
shrtimer_timers_repetition_value_config(SHRTIMER0, SHRTIMER_MASTER_TIMER, 100);
```

### 函数 `shrtimer_timers_repetition_value_get`

函数`shrtimer_timers_repetition_value_get`描述见下表：

表 3-532. 函数 `shrtimer_timers_repetition_value_get`

函数名称	<code>shrtimer_timers_repetition_value_get</code>
函数原型	<code>uint32_t shrtimer_timers_repetition_value_get(uint32_t shrtimer_periph, uint32_t timer_id);</code>
功能描述	获取Master_TIMER/Slave_TIMER的重复计数寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
<code>shrtimer_periph</code>	SHRTIMER外设
<code>SHRTIMERx(x=0)</code>	选择一个SHRTIMER外设
输入参数{in}	
<code>timer_id</code>	选择SHRTIMER中的一个定时器

SHRTIMER_MASTER_TIMER	选择Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	选择Slave_TIMERx(x=0..4)
输出参数{out}	
-	-
返回值	
uint32_t	重复计数值

例如：

```
/* get the counter repetition value in Master_TIMER and Slave_TIMER */
```

```
uint32_t value;
```

```
value = shrtimer_timers_repetition_value_get (SHRTIMER0, SHRTIMER_MASTER_TIMER);
```

### 函数 shrtimer\_exeefilter\_struct\_para\_init

函数shrtimer\_exeefilter\_struct\_para\_init描述见下表：

表 3-533. 函数 shrtimer\_exeefilter\_struct\_para\_init

函数名称	shrtimer_exeefilter_struct_para_init
函数原型	void shrtimer_exeefilter_struct_para_init(shrtimer_exeefilter_parameter_struct * exeefilter);
功能描述	初始化shrtimer_exeefilter_parameter_struct结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
exeefilter	shrtimer_exeefilter_parameter_struct结构体指针，结构体成员为 <a href="#">表3-487. 结构体shrtimer_exeefilter_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize external event filtering for Slave_TIMER configuration struct with a default value */
```

```
shrtimer_exeefilter_parameter_struct exeefilter;
```

```
shrtimer_exeefilter_struct_para_init(&exeefilter);
```

### 函数 shrtimer\_slavetimer\_exeevent\_filtering\_config

函数shrtimer\_slavetimer\_exeevent\_filtering\_config描述见下表：

表 3-534. 函数 `shrtimer_slavetimer_exeevent_filtering_config`

函数名称	<code>shrtimer_slavetimer_exeevent_filtering_config</code>
函数原型	<code>void shrtimer_slavetimer_exeevent_filtering_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t event_id, shrtimer_exeeventfilter_parameter_struct *exeeventfilter);</code>
功能描述	配置Slave_TIMER的外部事件滤波功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>shrtimer_periph</b>	SHRTIMER外设
<i>SHRTIMERx(x=0)</i>	选择一个SHRTIMER外设
输入参数{in}	
<b>timer_id</b>	选择SHRTIMER中的一个定时器
<i>SHRTIMER_SLAVE_TIMERx(x=0..4)</i>	选择Slave_TIMERx(x=0..4)
输入参数{in}	
<b>event_id</b>	SHRTIMER外部事件索引
<i>SHRTIMER_EXEVENT_NONE</i>	清除外部事件滤波配置
<i>SHRTIMER_EXEVENT_y(y=0..9)</i>	外部事件y(y=0..9)
输入参数{in}	
<b>exeeventfilter</b>	shrtimer_exeeventfilter_parameter_struct结构体指针，结构体成员为 <a href="#">表3-487. 结构体shrtimer_exeeventfilter_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* configure the external event filtering for Slave_TIMER (blanking, windowing) */

shrtimer_exeeventfilter_parameter_struct exeeventfilter;

shrtimer_exeeventfilter_struct_para_init(&exeeventfilter);

exeeventfilter.filter_mode = SHRTIMER_EXEVENTFILTER_BLANKINGCMP1;

exeeventfilter.memorized = SHRTIMER_EXEVMEMORIZED_DISABLE;

shrtimer_slavetimer_exeevent_filtering_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,
SHRTIMER_EXEVENT_5, &exeeventfilter);

```

### 函数 `shrtimer_exeeventcfg_struct_para_init`

函数`shrtimer_exeeventcfg_struct_para_init`描述见下表：

表 3-535. 函数 `shrtimer_exeventcfg_struct_para_init`

函数名称	<code>shrtimer_exeventcfg_struct_para_init</code>
函数原型	<code>void shrtimer_exeventcfg_struct_para_init(shrtimer_exeventcfg_parameter_struct * exeventcfg);</code>
功能描述	初始化 <code>shrtimer_exeventcfg_parameter_struct</code> 结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
<code>exeventcfg</code>	<code>shrtimer_exeventcfg_parameter_struct</code> 结构体指针，结构体成员参考 <a href="#">表3-492. 结构体shrtimer_exeventcfg_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize external event configuration struct with a default value */
```

```
shrtimer_exeventcfg_parameter_struct exeventcfg;
```

```
shrtimer_exeventcfg_struct_para_init(&exeventcfg);
```

### 函数 `shrtimer_exevent_config`

函数`shrtimer_exevent_config`描述见下表：

表 3-536. 函数 `shrtimer_exevent_config`

函数名称	<code>shrtimer_exevent_config</code>
函数原型	<code>void shrtimer_exevent_config(uint32_t shrtimer_periph, uint32_t event_id, shrtimer_exeventcfg_parameter_struct* exeventcfg);</code>
功能描述	配置外部事件
先决条件	-
被调用函数	-
输入参数{in}	
<code>shrtimer_periph</code>	SHRTIMER外设
<code>SHRTIMERx(x=0)</code>	选择一个SHRTIMER外设
输入参数{in}	
<code>event_id</code>	外部事件索引
<code>SHRTIMER_EXEVENT_NONE</code>	清除外部事件配置
<code>SHRTIMER_EXEVENT_y(y=0..9)</code>	外部事件y(y=0..9)
输入参数{in}	
<code>exeventcfg</code>	<code>shrtimer_exeventcfg_parameter_struct</code> 结构体指针，结构体成员参考

	<a href="#">表3-492. 结构体shrtimer_exeventcfg_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the an external event */

shrtimer_exeventcfg_parameter_struct exevcfg;

shrtimer_exeventcfg_struct_para_init(&exevcfg);

exevcfg.digital_filter = 0x5;

exevcfg.edge = SHRTIMER_EXEV_EDGE_RISING;

exevcfg.polarity = SHRTIMER_EXEV_EDGE_LEVEL;

exevcfg.source = SHRTIMER_EXEV_SRC0;

shrtimer_exevent_config(SHRTIMER0, SHRTIMER_EXEVENT_5, &exevcfg);
```

### 函数 shrtimer\_exevent\_prescaler

函数shrtimer\_exevent\_prescaler描述见下表：

表 3-537. 函数 shrtimer\_exevent\_prescaler

函数名称	shrtimer_exevent_prescaler
函数原型	void shrtimer_exevent_prescaler(uint32_t shrtimer_periph, uint32_t prescaler);
功能描述	配置外部事件的数字滤波分频
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
prescaler	外部事件数字滤波器时钟分频
SHRTIMER_EXEV_PRESCALER_DIVx(x=1,2,4,8)	fSHRTIMER_EXEV_FCK = fSHRTIMER_CK/x (x=1,2,4,8)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure external event digital filter clock division */
```

```
shrtimer_exevent_prescaler(SHRTIMER0, SHRTIMER_EXEV_PRESCALER_DIV1);
```

### 函数 shrtimer\_synccfg\_struct\_para\_init

函数shrtimer\_synccfg\_struct\_para\_init描述见下表：

表 3-538. 函数 shrtimer\_synccfg\_struct\_para\_init

函数名称	shrtimer_synccfg_struct_para_init
函数原型	void shrtimer_synccfg_struct_para_init(shrtimer_synccfg_parameter_struct* synccfg);
功能描述	初始化shrtimer_synccfg_parameter_struct结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
synccfg	shrtimer_synccfg_parameter_struct结构体指针，结构体成员参考 <a href="#">表3-490. 结构体shrtimer_synccfg_parameter_struct</a> 结构体shrtimer_synccfg_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize synchronization configuration struct with a default value */
```

```
shrtimer_synccfg_parameter_struct synccfg;
```

```
shrtimer_synccfg_struct_para_init (&synccfg);
```

### 函数 shrtimer\_synchronization\_config

函数shrtimer\_synchronization\_config描述见下表：

表 3-539. 函数 shrtimer\_synchronization\_config

函数名称	shrtimer_synchronization_config
函数原型	void shrtimer_synchronization_config(uint32_t shrtimer_periph, shrtimer_synccfg_parameter_struct* synccfg);
功能描述	配置SHRTIMER的同步输入/输出功能
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	

<b>synccfg</b>	shrtimer_synccfg_parameter_struct结构体指针，结构体成员参考 <a href="#">表3-490. 结构体shrtimer_synccfg_parameter_struct</a> 结构体 shrtimer_synccfg_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the synchronization input/output of the SHRTIMER */

shrtimer_synccfg_parameter_struct synccfg;

shrtimer_synccfg_struct_para_init (&synccfg);

synccfg.input_source = SHRTIMER_SYNCINPUTSOURCE_EXTERNAL;

shrtimer_synchronization_config(SHRTIMER0, &synccfg);
```

### 函数 shrtimer\_faultcfg\_struct\_para\_init

函数shrtimer\_faultcfg\_struct\_para\_init描述见下表：

**表 3-540. 函数 shrtimer\_faultcfg\_struct\_para\_init**

<b>函数名称</b>	shrtimer_faultcfg_struct_para_init
<b>函数原型</b>	void shrtimer_faultcfg_struct_para_init(shrtimer_faultcfg_parameter_struct * faultcfg);
<b>功能描述</b>	初始化shrtimer_faultcfg_parameter_struct结构体变量
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>faultcfg</b>	shrtimer_faultcfg_parameter_struct结构体指针，结构体成员参考 <a href="#">表3-493. 结构体shrtimer_faultcfg_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize fault input configuration struct with a default value */

shrtimer_faultcfg_parameter_struct faultcfg;

shrtimer_synccfg_struct_para_init (&faultcfg);
```

**函数 shrtimer\_fault\_config**

函数shrtimer\_fault\_config描述见下表：

**表 3-541. 函数 shrtimer\_fault\_config**

函数名称	shrtimer_fault_config
函数原型	void shrtimer_fault_config(uint32_t shrtimer_periph, uint32_t fault_id, shrtimer_faultcfg_parameter_struct* faultcfg);
功能描述	配置故障输入功能
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
fault_id	故障输入索引
SHRTIMER_FAULT_y(y=0..4)	选择一个故障输入
输入参数{in}	
faultcfg	shrtimer_faultcfg_parameter_struct结构体指针，结构体成员参考 <a href="#">表3-493. 结构体shrtimer_faultcfg_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the fault input */
shrtimer_faultcfg_parameter_struct faultcfg;
shrtimer_synccfg_struct_para_init (&faultcfg);
faultcfg_para.control = SHRTIMER_FAULT_CHANNEL_ENABLE;
faultcfg_para.filter = 0x0;
faultcfg_para.polarity = SHRTIMER_FAULT_POLARITY_HIGH;
faultcfg_para.protect = SHRTIMER_FAULT_PROTECT_ENABLE;
faultcfg_para.source = SHRTIMER_FAULT_SOURCE_PIN;
shrtimer_fault_config(SHRTIMER0, SHRTIMER_FAULT_0, &faultcfg);
```

**函数 shrtimer\_fault\_prescaler\_config**

函数shrtimer\_fault\_prescaler\_config描述见下表：

表 3-542. 函数 `shrtimer_fault_prescaler_config`

函数名称	<code>shrtimer_fault_prescaler_config</code>
函数原型	<code>void shrtimer_fault_prescaler_config(uint32_t shrtimer_periph, uint32_t prescaler);</code>
功能描述	配置故障输入数字滤波分频
先决条件	-
被调用函数	-
输入参数{in}	
<code>shrtimer_periph</code>	SHRTIMER外设
<code>SHRTIMERx(x=0)</code>	选择一个SHRTIMER外设
输入参数{in}	
<code>prescaler</code>	故障输入数字滤波器时钟分频
<code>SHRTIMER_FAULT_PRESCALER_DIVy(y=1, 2, 4, 8)</code>	$f_{SHRTIMER\_FLTICK} = f_{SHRTIMER\_CK}/y$ ( $y=1, 2, 4, 8$ )
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the fault input digital filter clock division */
```

```
void shrtimer_fault_prescaler_config(SHRTIMER0, SHRTIMER_FAULT_PRESCALER_DIV1);
```

### 函数 `shrtimer_fault_input_enable`

函数`shrtimer_fault_input_enable`描述见下表：

表 3-543. 函数 `shrtimer_fault_input_enable`

函数名称	<code>shrtimer_fault_input_enable</code>
函数原型	<code>void shrtimer_fault_input_enable(uint32_t shrtimer_periph, uint32_t fault_id);</code>
功能描述	故障输入使能
先决条件	-
被调用函数	-
输入参数{in}	
<code>shrtimer_periph</code>	SHRTIMER外设
<code>SHRTIMERx(x=0)</code>	选择一个SHRTIMER外设
输入参数{in}	
<code>fault_id</code>	故障输入索引
<code>SHRTIMER_FAULT_y(y=0..4)</code>	选择一个故障输入

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* fault input enable */
```

```
shrtimer_fault_input_enable(SHRTIMER0, SHRTIMER_FAULT_0);
```

### 函数 shrtimer\_fault\_input\_disable

函数shrtimer\_fault\_input\_disable描述见下表：

表 3-544. 函数 shrtimer\_fault\_input\_disable

函数名称	shrtimer_fault_input_disable
函数原型	void shrtimer_fault_input_disable(uint32_t shrtimer_periph, uint32_t fault_id);
功能描述	故障输入禁能
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
fault_id	故障输入索引
SHRTIMER_FAULT_y(y=0..4)	选择一个故障输入
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* fault input disable */
```

```
shrtimer_fault_input_disable(SHRTIMER0, SHRTIMER_FAULT_0);
```

### 函数 shrtimer\_timers\_dma\_enable

函数shrtimer\_timers\_dma\_enable描述见下表：

表 3-545. 函数 shrtimer\_timers\_dma\_enable

函数名称	shrtimer_timers_dma_enable
函数原型	void shrtimer_timers_dma_enable(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t dmareq);

功能描述	使能Master_TIMER/Slave_TIMER DMA请求
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_MASTER_TIMER	选择Master_TIMER
SHRTIMER_SLAVE_TIMERx	选择Slave_TIMERx (x=0..4)
输入参数{in}	
dmareq	DMA请求源
SHRTIMER_MT_ST_DMA_CMPy(y=0..3)	比较y(y=0..3)DMA请求 (适用于Master_TIMER和Slave_TIMER)
SHRTIMER_MT_ST_DMA_REP	重复DMA请求 (适用于Master_TIMER和Slave_TIMER)
SHRTIMER_MT_DMA_SYNID	同步输入DMA请求 (仅适用于Master_TIMER)
SHRTIMER_MT_ST_DMA_UPD	更新DMA请求 (适用于Master_TIMER和Slave_TIMER)
SHRTIMER_ST_DMA_CAPy(y=0,1)	捕获y(y=0,1)DMA请求 (仅适用于Slave_TIMER)
SHRTIMER_ST_DMA_CHyOA(y=0,1)	通道y(y=0,1)输出置位DMA请求 (仅适用于Slave_TIMER)
SHRTIMER_ST_DMA_CHyONA(y=0,1)	通道y(y=0,1)输出置位DMA请求 (仅适用于Slave_TIMER)
SHRTIMER_ST_DMA_CNTRST	计数器复位DMA请求 (仅适用于Slave_TIMER)
SHRTIMER_ST_DMA_DLYIDLE	延迟空闲模式进入DMA请求 (仅适用于Slave_TIMER)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the Master_TIMER and Slave_TIMER DMA request */
```

```
shrtimer_timers_dma_enable(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_DMA_CMP0);
```

## 函数 shrtimer\_timers\_dma\_disable

函数shrtimer\_timers\_dma\_disable描述见下表：

表 3-546. 函数 shrtimer\_timers\_dma\_enable

函数名称	shrtimer_timers_dma_disable
函数原型	void shrtimer_timers_dma_disable(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t dmareq);
功能描述	禁能Master_TIMER/Slave_TIMER DMA请求
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_MASTER_TIMER	选择Master_TIEMR
SHRTIMER_SLAVE_TIMERx	选择Slave_TIEMRx (x=0..4)
输入参数{in}	
dmareq	DMA请求源
SHRTIMER_MT_ST_DMA_CMPy(y=0..3)	比较y(y=0..3)DMA请求（适用于Master_TIMER和Slave_TIMER）
SHRTIMER_MT_ST_DMA_REP	重复DMA请求（适用于Master_TIMER和Slave_TIMER）
SHRTIMER_MT_DMA_SYNCI	同步输入DMA请求（仅适用于Master_TIMER）
SHRTIMER_MT_ST_DMA_UPD	更新DMA请求（适用于Master_TIMER和Slave_TIMER）
SHRTIMER_ST_DMA_CAPy(y=0,1)	捕获y(y=0,1)DMA请求（仅适用于Slave_TIMER）
SHRTIMER_ST_DMA_CHyOA(y=0,1)	通道y(y=0,1)输出置位DMA请求（仅适用于Slave_TIMER）
SHRTIMER_ST_DMA_CHyONA(y=0,1)	通道y(y=0,1)输出置位DMA请求（仅适用于Slave_TIMER）
SHRTIMER_ST_DMA_CNTRST	计数器复位DMA请求（仅适用于Slave_TIMER）
SHRTIMER_ST_DMA_DLYIDLE	延迟空闲模式进入DMA请求（仅适用于Slave_TIMER）
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* disable the Master_TIMER and Slave_TIMER DMA request */
```

```
shrtimer_timers_dma_disable(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_DMA_CMP0);
```

## 函数 shrtimer\_dmamode\_config

函数shrtimer\_dmamode\_config描述见下表:

表 3-547. 函数 shrtimer\_dmamode\_config

函数名称	shrtimer_dmamode_config
函数原型	void shrtimer_dmamode_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t regupdate);
功能描述	配置Master_TIMER/Slave_TIMER的DMA模式
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_MASTER_TIMER	选择Master_TIEMR
SHRTIMER_SLAVE_TIMERx	选择Slave_TIEMRx (x=0..4)
输入参数{in}	
regupdate	更新的寄存器
SHRTIMER_DMAMODE_NONE	不通过DMA模式更新寄存器 (适用于Master_TIMER和Slave_TIMER)
SHRTIMER_DMAMODE_CTL0	DMA模式更新MTCTL0, STxCTL0寄存器 (适用于Master_TIMER和Slave_TIMER)
SHRTIMER_DMAMODE_INTC	DMA模式更新MT, STx寄存器 (适用于Master_TIMER和Slave_TIMER)
SHRTIMER_DMAMODE_DMAINTEN	DMA模式更新MTINTC, STxINTC寄存器 (适用于Master_TIMER和Slave_TIMER)
SHRTIMER_DMAMODE_CNT	DMA模式更新MTCNT, STxCNT寄存器 (适用于Master_TIMER和Slave_TIMER)
SHRTIMER_DMAMODE_CAR	DMA模式更新MTCAR, STxCAR寄存器 (适用于Master_TIMER和Slave_TIMER)
SHRTIMER_DMAMODE_CREP	DMA模式更新MTCREP, STxCREP寄存器 (适用于Master_TIMER和Slave_TIMER)

SHRTIMER_DMAMOD E_CMPyV(y=0..3)	DMA模式更新MTCMPyV(y=0..3), STxCMPyV(y=0..3)寄存器 (适用于Master_TIMER和Slave_TIMER)
SHRTIMER_DMAMOD E_DTCTL	DMA模式更新STxDTCCTL寄存器 (仅适用于Slave_TIMER)
SHRTIMER_DMAMOD E_CHySET(y=0,1)	DMA模式更新STxCHySET(y=0,1)寄存器 (仅适用于Slave_TIMER)
SHRTIMER_DMAMOD E_CHyRST(y=0,1)	DMA模式更新STxCHyRST(y=0,1)寄存器 (仅适用于Slave_TIMER)
SHRTIMER_DMAMOD E_EXEVFCFGy(y=0,1)	DMA模式更新STxEXEVFCFGy(y=0,1)寄存器 (仅适用于Slave_TIMER)
SHRTIMER_DMAMOD E_CNTRST	DMA模式更新STxCNTRST寄存器 (仅适用于Slave_TIMER)
SHRTIMER_DMAMOD E_CSCTL	DMA模式更新STxCSCCTL寄存器 (仅适用于Slave_TIMER)
SHRTIMER_DMAMOD E_CHOCTL	DMA模式更新STxCHOCTL寄存器 (仅适用于Slave_TIMER)
SHRTIMER_DMAMOD E_FLTCTL	DMA模式更新STxFLTCTL寄存器 (仅适用于Slave_TIMER)
SHRTIMER_DMAMOD E_ACTL	DMA模式更新STxACTL寄存器 (仅适用于Slave_TIMER)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the Master_TIMER and Slave_TIMER DMA request */

shrtimer_dmamode_config(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_DMAMODE_NONE);
```

### 函数 shrtimer\_bunchmode\_struct\_para\_init

函数shrtimer\_bunchmode\_struct\_para\_init描述见下表:

表 3-548. 函数 shrtimer\_bunchmode\_struct\_para\_init

函数名称	shrtimer_bunchmode_struct_para_init
函数原型	void shrtimer_bunchmode_struct_para_init(shrtimer_bunchmode_parameter_struct* bmcfg);
功能描述	初始化shrtimer_bunchmode_parameter_struct结构体
先决条件	-
被调用函数	-
输入参数{in}	

<b>bmcfg</b>	shrtimer_bunchmode_parameter_struct结构体指针，结构体成员参考 <a href="#">表3-491. 结构体shrtimer_bunchmode_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize bunch mode configuration struct with a default value */
```

```
shrtimer_bunchmode_parameter_struct bmcfg;
```

```
shrtimer_bunchmode_struct_para_init(&bmcfg);
```

### 函数 shrtimer\_bunchmode\_config

函数shrtimer\_bunchmode\_config描述见下表：

**表 3-549. 函数 shrtimer\_bunchmode\_config**

函数名称	shrtimer_bunchmode_config
函数原型	void shrtimer_bunchmode_config(uint32_t shrtimer_periph, shrtimer_bunchmode_parameter_struct* bmcfg);
功能描述	配置SHRTIMER的突发模式
先决条件	-
被调用函数	-
输入参数{in}	
<b>shrtimer_periph</b>	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
<b>bmcfg</b>	shrtimer_bunchmode_parameter_struct结构体指针，结构体成员参考 <a href="#">表3-491. 结构体shrtimer_bunchmode_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure bunch mode for the SHRTIMER */
```

```
shrtimer_bunchmode_parameter_struct bmcfg;
```

```
shrtimer_bunchmode_struct_para_init(&bmcfg);
```

```
bmcfg.clock_source = SHRTIMER_BUNCHMODE_CLOCKSOURCE_ST0;
```

```
bmcfg.idle_duration = 3;
```

```
bmcfg.mode = SHRTIMER_BUNCHMODE_CONTINUOUS;
```

```

bmcfg.period = 6;

bmcfg.prescaler = SHRTIMER_BUNCHMODE_PRESCALER_DIV1;

bmcfg.shadow = SHRTIMER_BUNCHMODEPRELOAD_DISABLED;

bmcfg.trigger = SHRTIMER_BUNCHMODE_TRIGGER_SOFTWARE;

shrtimer_bunchmode_config(SHRTIMER0, &bmcfg);

```

### 函数 shrtimer\_bunchmode\_enable

函数shrtimer\_bunchmode\_enable描述见下表：

表 3-550. 函数 shrtimer\_bunchmode\_enable

函数名称	shrtimer_bunchmode_enable
函数原型	void shrtimer_bunchmode_enable(uint32_t shrtimer_periph);
功能描述	使能突发模式
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* enable bunch mode for the SHRTIMER */

shrtimer_bunchmode_enable(SHRTIMER0);

```

### 函数 shrtimer\_bunchmode\_disable

函数shrtimer\_bunchmode\_disable描述见下表：

表 3-551. 函数 shrtimer\_bunchmode\_disable

函数名称	shrtimer_bunchmode_disable
函数原型	void shrtimer_bunchmode_disable(uint32_t shrtimer_periph);
功能描述	禁能突发模式
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* disable bunch mode for the SHRTIMER */
shrtimer_bunchmode_disable(SHRTIMER0);
```

### 函数 shrtimer\_bunchmode\_flag\_get

函数shrtimer\_bunchmode\_flag\_get描述见下表：

表 3-552. 函数 shrtimer\_bunchmode\_flag\_get

函数名称	shrtimer_bunchmode_flag_get
函数原型	uint32_t shrtimer_bunchmode_flag_get(uint32_t shrtimer_periph);
功能描述	获取突发模式的运行标志
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输出参数{out}	
-	-
返回值	
uint32_t	SHRTIMER_BUNCHMODE_OPERATION_OFF或 SHRTIMER_BUNCHMODE_OPERATION_ON

例如：

```
/* get bunch mode operating flag */
uint32_t flag;
flag = shrtimer_bunchmode_flag_get(SHRTIMER0);
```

### 函数 shrtimer\_bunchmode\_software\_start

函数shrtimer\_bunchmode\_software\_start描述见下表：

表 3-553. 函数 shrtimer\_bunchmode\_software\_start

函数名称	shrtimer_bunchmode_software_start
函数原型	void shrtimer_bunchmode_software_start(uint32_t shrtimer_periph);
功能描述	软件启动突发模式
先决条件	-
被调用函数	-
输入参数{in}	

<b>shrtimer_periph</b>	SHRTIMER外设
<i>SHRTIMERx(x=0)</i>	选择一个SHRTIMER外设
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* bunch mode started by software */
```

```
shrtimer_bunchmode_software_start(SHRTIMER0);
```

### 函数 shrtimer\_slavetimer\_capture\_config

函数shrtimer\_slavetimer\_capture\_config描述见下表:

表 3-554. 函数 shrtimer\_slavetimer\_capture\_config

函数名称	shrtimer_slavetimer_capture_config
函数原型	void shrtimer_slavetimer_capture_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t capturex, uint32_t trgsources);
功能描述	配置Slave_TIMER捕获源
先决条件	-
被调用函数	-
输入参数{in}	
<b>shrtimer_periph</b>	SHRTIMER外设
<i>SHRTIMERx(x=0)</i>	选择一个SHRTIMER外设
输入参数{in}	
<b>timer_id</b>	选择SHRTIMER中的一个定时器
<i>SHRTIMER_SLAVE_TIMERx</i>	选择Slave_TIMERx(x=0..4)
输入参数{in}	
<b>capturex</b>	捕获单元索引
<i>SHRTIMER_CAPTURE_y(y=0,1)</i>	选择捕获单元y(y=0,1)
输入参数{in}	
<b>trgsources</b>	捕获触发源
<i>SHRTIMER_CAPTURE_TRIGGER_NONE</i>	清除所有捕获触发源
<i>SHRTIMER_CAPTURE_TRIGGER_UPDATE</i>	更新事件触发捕获
<i>SHRTIMER_CAPTURE_TRIGGER_EXEV_y(y=0..9)</i>	外部事件y(y=0..9)触发捕获
<i>SHRTIMER_CAPTURE</i>	Slave_TIMERy(y=0..4)的通道0(STych0_O)的输出有效电平触发捕获

TRIGGER_STy_ACTIV E(y=0..4)	
SHRTIMER_CAPTURE TRIGGER_STy_INACTI VE(y=0..4)	Slave_TIMERy(y=0..4)的通道0(STyCH0_O)的输出无效电平触发捕获
SHRTIMER_CAPTURE TRIGGER_STy_CMP0( y=0..4)	Slave_TIMERy(y=0..4)的比较0事件触发捕获
SHRTIMER_CAPTURE TRIGGER_STy_CMP1( y=0..4)	Slave_TIMERy(y=0..4)的比较1事件触发捕获
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the capture source in Slave_TIMER */
```

```
shrtimer_slavetimer_capture_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,  
SHRTIMER_CAPTURE_0, SHRTIMER_CAPTURETRIGGER_NONE);
```

### 函数 shrtimer\_slavetimer\_capture\_software

函数shrtimer\_slavetimer\_capture\_software描述见下表:

表 3-555. 函数 shrtimer\_slavetimer\_capture\_software

函数名称	shrtimer_slavetimer_capture_software
函数原型	void shrtimer_slavetimer_capture_software(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t capturex);
功能描述	软件触发Slave_TIMER捕获
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_SLAVE_TI MERx	选择Slave_TIMERx(x=0..4)
输入参数{in}	
capturex	捕获单元索引
SHRTIMER_CAPTURE _y(y=0,1)	选择捕获单元y(y=0,1)

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the capture source in Slave_TIMER */
```

```
shrtimer_slavetimer_capture_software(SHRTIMER0, SHRTIMER_SLAVE
SHRTIMER_CAPTURE_0);
```

### 函数 shrtimer\_slavetimer\_capture\_value\_read

函数shrtimer\_slavetimer\_capture\_value\_read描述见下表:

**表 3-556. 函数 shrtimer\_slavetimer\_capture\_value\_read**

函数名称	shrtimer_slavetimer_capture_value_read
函数原型	uint32_t shrtimer_slavetimer_capture_value_read(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t capturex);
功能描述	读取捕获值
先决条件	-
被调用函数	-
输入参数{in}	
<b>shrtimer_periph</b>	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
<b>timer_id</b>	选择SHRTIMER中的一个定时器
SHRTIMER_SLAVE_TIMERx	选择Slave_TIMERx(x=0..4)
输入参数{in}	
<b>capturex</b>	捕获单元索引
SHRTIMER_CAPTURE_y(y=0,1)	选择捕获单元y(y=0,1)
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	捕获值

例如:

```
/* read the capture value */
```

```
uint32_t capture_value;
```

```
capture_value = shrtimer_slavetimer_capture_value_read (SHRTIMER0,
SHRTIMER_SLAVE SHRTIMER_CAPTURE_0);
```

**函数 shrtimer\_adctrigcfg\_struct\_para\_init**

函数shrtimer\_adctrigcfg\_struct\_para\_init描述见下表:

**表 3-557. 函数 shrtimer\_adctrigcfg\_struct\_para\_init**

函数名称	shrtimer_adctrigcfg_struct_para_init
函数原型	void shrtimer_adctrigcfg_struct_para_init(shrtimer_adctrigcfg_parameter_struct* triggercfg);
功能描述	初始化shrtimer_adctrigcfg_parameter_struct结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
triggercfg	shrtimer_adctrigcfg_parameter_struct结构体指针, 结构体成员参考 <a href="#">表3-494. 结构体shrtimer_adctrigcfg_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize ADC trigger configuration struct with a default value */
```

```
shrtimer_adctrigcfg_parameter_struct triggercfg;
```

```
shrtimer_adctrigcfg_struct_para_init(&triggercfg);
```

**函数 shrtimer\_adc\_trigger\_config**

函数shrtimer\_adc\_trigger\_config描述见下表:

**表 3-558. 函数 shrtimer\_adc\_trigger\_config**

函数名称	shrtimer_adc_trigger_config
函数原型	void shrtimer_adc_trigger_config(uint32_t shrtimer_periph, uint32_t trigger_id, shrtimer_adctrigcfg_parameter_struct* triggercfg);
功能描述	配置ADC触发源和更新源
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
trigger_id	ADC触发源
SHRTIMER_ADCTRIG_y(y=0..3)	选择一个ADC触发源

输入参数{in}	
triggercfg	shrtimer_adctrigcfg_parameter_struct结构体指针，结构体成员参考 <a href="#">表3-494. 结构体shrtimer_adctrigcfg_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the trigger source to ADC and the update source */
shrtimer_adctrigcfg_parameter_struct triggercfg;

shrtimer_adctrigcfg_struct_para_init(&triggercfg);

triggercfg.update_source = SHRTIMER_ADCTRGI_UPDATE_MT;

triggercfg.trigger = SHRTIMER_ADCTRGI02_EVENT_NONE;

shrtimer_adc_trigger_config(SHRTIMER0, SHRTIMER_ADCTRIG_0, &triggercfg);
```

### 函数 shrtimer\_timers\_flag\_get

函数shrtimer\_timers\_flag\_get描述见下表：

表 3-559. 函数 shrtimer\_timers\_flag\_get

函数名称	shrtimer_timers_flag_get
函数原型	FlagStatus shrtimer_timers_flag_get(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t flag);
功能描述	获取Master_TIMER/Slave_TIMER的标志
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_MASTER_TIMER	选择Master_TIMER
SHRTIMER_SLAVE_TIMERx	选择Slave_TIMERx(x=0..4)
输入参数{in}	
flag	选择一个标志源
SHRTIMER_MT_ST_FLAG_CMPy(y=0..3)	比较y(y=0..3)中断标志（适用于Master_TIMER和Slave_TIMER）
SHRTIMER_MT_ST_F	重复中断标志（适用于Master_TIMER 和 Slave_TIMER）

LAG_REP	
SHRTIMER_MT_INT_F LAG_SYNI	同步输入中断标志（仅适用于Master_TIMER）
SHRTIMER_MT_ST_F LAG_UPD	更新中断标志（适用于Master_TIMER和Slave_TIMER）
SHRTIMER_ST_FLAG _CAPy(y=0,1)	捕获y(y=0,1)中断标志（仅适用于Slave_TIMER）
SHRTIMER_ST_FLAG _CHyOA(y=0,1)	通道y(y=0,1)输出置位请求中断标志（仅适用于Slave_TIMER）
SHRTIMER_ST_FLAG _CHyONA(y=0,1)	通道y(y=0,1)输出复位请求中断标志（仅适用于Slave_TIMER）
SHRTIMER_ST_FLAG _CNTRST	计数器复位中断标志（仅适用于Slave_TIMER）
SHRTIMER_ST_FLAG _DLYIDLE	延迟空闲模式进入中断标志（仅适用于Slave_TIMER）
SHRTIMER_ST_FLAG _CBLN	当前的均衡状态标志（仅适用于Slave_TIMER）
SHRTIMER_ST_FLAG _BLNIDLE	均衡空闲标志（仅适用于Slave_TIMER）
SHRTIMER_ST_FLAG _CHyOUT(y=0,1)	通道 y(y=0,1)输出标志（仅适用于Slave_TIMER）
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* get the Master_TIMER and Slave_TIMER flag */
```

```
FlagStatus flag = RESET;
```

```
flag = shrtimer_timers_flag_get(SHRTIMER0, SHRTIMER_MASTER_TIMER,  
SHRTIMER_MT_ST_FLAG_CMP0);
```

### 函数 shrtimer\_timers\_flag\_clear

函数shrtimer\_timers\_flag\_clear描述见下表：

表 3-560. 函数 shrtimer\_timers\_flag\_clear

函数名称	shrtimer_timers_flag_clear
函数原型	void shrtimer_timers_flag_clear(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t flag);
功能描述	clear the Master_TIMER and Slave_TIMER flag
先决条件	-
被调用函数	-

输入参数{in}	
<b>shrtimer_periph</b>	SHRTIMER外设
<i>SHRTIMERx(x=0)</i>	选择一个SHRTIMER外设
输入参数{in}	
<b>timer_id</b>	选择SHRTIMER中的一个定时器
<i>SHRTIMER_MASTER_TIMER</i>	选择Master_TIMER
<i>SHRTIMER_SLAVE_TIMERx</i>	选择Slave_TIMERx(x=0..4)
输入参数{in}	
<b>flag</b>	the flag source
<i>SHRTIMER_MT_ST_FLAG_CMPy(y=0..3)</i>	比较y(y=0..3)中断标志（适用于Master_TIMER和Slave_TIMER）
<i>SHRTIMER_MT_ST_FLAG_LAG_REP</i>	重复中断标志（适用于Master_TIMER和Slave_TIMER）
<i>SHRTIMER_MT_INT_FLAG_LAG_SYNI</i>	同步输入中断标志（仅适用于Master_TIMER）
<i>SHRTIMER_MT_ST_FLAG_LAG_UPD</i>	更新中断标志（适用于Master_TIMER和Slave_TIMER）
<i>SHRTIMER_ST_FLAG_CAPy(y=0,1)</i>	捕获y(y=0,1)中断标志（仅适用于Slave_TIMER）
<i>SHRTIMER_ST_FLAG_CHyOA(y=0,1)</i>	通道y(y=0,1)输出置位请求中断标志（仅适用于Slave_TIMER）
<i>SHRTIMER_ST_FLAG_CHyONA(y=0,1)</i>	通道y(y=0,1)输出复位请求中断标志（仅适用于Slave_TIMER）
<i>SHRTIMER_ST_FLAG_CNTRST</i>	计数器复位中断标志（仅适用于Slave_TIMER）
<i>SHRTIMER_ST_FLAG_DLYIDLE</i>	延迟空闲模式进入中断标志（仅适用于Slave_TIMER）
<i>SHRTIMER_ST_FLAG_CBLN</i>	当前的均衡状态标志（仅适用于Slave_TIMER）
<i>SHRTIMER_ST_FLAG_BLNIDLE</i>	均衡空闲标志（仅适用于Slave_TIMER）
<i>SHRTIMER_ST_FLAG_CHyOUT(y=0,1)</i>	通道y(y=0,1)输出标志（仅适用于Slave_TIMER）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the Master_TIMER and Slave_TIMER flag */
```

```
shrtimer_timers_flag_clear(SHRTIMER0, SHRTIMER_MASTER_TIMER,  
SHRTIMER_MT_ST_FLAG_CMP0);
```

### 函数 shrtimer\_common\_flag\_get

函数shrtimer\_common\_flag\_get描述见下表:

**表 3-561. 函数 shrtimer\_common\_flag\_get**

函数名称	shrtimer_common_flag_get
函数原型	FlagStatus shrtimer_common_flag_get(uint32_t shrtimer_periph, uint32_t flag);
功能描述	获取SHRTIMER通用标志
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
flag	标志源
SHRTIMER_FLAG_FLT y(y=0..3)	故障y(y=0..3)中断标志
SHRTIMER_FLAG_SY SFLT	系统故障中断标志
SHRTIMER_FLAG_DL LCAL	DLL 校准完成中断标志
SHRTIMER_FLAG_BM PER	突发模式周期中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如:

```
/* get the common flag */
```

```
FlagStatus flag = RESET;
```

```
flag = shrtimer_common_flag_get(SHRTIMER0, SHRTIMER_FLAG_FLT0);
```

### 函数 shrtimer\_common\_flag\_clear

函数shrtimer\_common\_flag\_clear描述见下表:

**表 3-562. 函数 shrtimer\_common\_flag\_clear**

函数名称	shrtimer_common_flag_clear
函数原型	void shrtimer_common_flag_clear(uint32_t shrtimer_periph, uint32_t flag);

功能描述	清除SHRTIMER通用标志
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
flag	标志源
SHRTIMER_FLAG_FLT y(y=0..3)	故障y(y=0..3)中断标志
SHRTIMER_FLAG_SY SFLT	系统故障中断标志
SHRTIMER_FLAG_DL LCAL	DLL校准完成中断标志
SHRTIMER_FLAG_BM PER	突发模式周期中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the common flag */
```

```
shrtimer_common_flag_clear(SHRTIMER0, SHRTIMER_FLAG_FLT0);
```

### 函数 shrtimer\_timers\_interrupt\_enable

函数shrtimer\_timers\_interrupt\_enable描述见下表：

表 3-563. 函数 shrtimer\_timers\_interrupt\_enable

函数名称	shrtimer_timers_interrupt_enable
函数原型	void shrtimer_timers_interrupt_enable(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t interrupt);
功能描述	使能Master_TIMER/Slave_TIMER中断
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_MASTER_ TIMER	选择Master_TIMER

SHRTIMER_SLAVE_TIMERx	选择Slave_TIMERx(x=0..4)
输入参数{in}	
interrupt	中断源
SHRTIMER_MT_ST_INT_CMPy(y=0..3)	比较y(y=0..3)中断（适用于Master_TIMER和Slave_TIMER）
SHRTIMER_MT_ST_INT_T_REP	重复中断（适用于Master_TIMER和Slave_TIMER）
SHRTIMER_MT_INT_SYNCI	同步输入中断（仅适用于Master_TIMER）
SHRTIMER_MT_ST_INT_T_UPD	更新中断标志（适用于Master_TIMER和Slave_TIMER）
SHRTIMER_ST_INT_CAPTUREy(y=0,1)	捕获y(y=0,1)中断（仅适用于Slave_TIMER）
SHRTIMER_ST_INT_HYOAy(y=0,1)	通道y(y=0,1)输出置位请求中断（仅适用于Slave_TIMER）
SHRTIMER_ST_INT_HYONAy(y=0,1)	通道y(y=0,1)输出复位请求中断（仅适用于Slave_TIMER）
SHRTIMER_ST_INT_NTRST	计数器复位中断（仅适用于Slave_TIMER）
SHRTIMER_ST_INT_DLYIDLE	延迟空闲模式进入中断（仅适用于Slave_TIMER）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the Master_TIMER and Slave_TIMER interrupt */
```

```
shrtimer_timers_interrupt_enable(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_INT_CMP0);
```

### 函数 shrtimer\_timers\_interrupt\_disable

函数shrtimer\_timers\_interrupt\_disable描述见下表：

表 3-564. 函数 shrtimer\_timers\_interrupt\_disable

函数名称	shrtimer_timers_interrupt_disable
函数原型	void shrtimer_timers_interrupt_disable(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t interrupt);
功能描述	禁能Master_TIMER/Slave_TIMER中断
先决条件	-
被调用函数	-

输入参数{in}	
<b>shrtimer_periph</b>	SHRTIMER外设
<i>SHRTIMERx(x=0)</i>	选择一个SHRTIMER外设
输入参数{in}	
<b>timer_id</b>	选择SHRTIMER中的一个定时器
<i>SHRTIMER_MASTER_TIMER</i>	选择Master_TIMER
<i>SHRTIMER_SLAVE_TIMERx</i>	选择Slave_TIMERx(x=0..4)
输入参数{in}	
<b>interrupt</b>	中断源
<i>SHRTIMER_MT_ST_INT_CMPy(y=0..3)</i>	比较y(y=0..3)中断（适用于Master_TIMER和Slave_TIMER）
<i>SHRTIMER_MT_ST_INT_T_REP</i>	重复中断（适用于Master_TIMER和Slave_TIMER）
<i>SHRTIMER_MT_INT_SYNCI</i>	同步输入中断（仅适用于Master_TIMER）
<i>SHRTIMER_MT_ST_INT_T_UPD</i>	更新中断标志（适用于Master_TIMER和Slave_TIMER）
<i>SHRTIMER_ST_INT_CAPy(y=0,1)</i>	捕获y(y=0,1)中断（仅适用于Slave_TIMER）
<i>SHRTIMER_ST_INT_CHyOA(y=0,1)</i>	通道y(y=0,1)输出置位请求中断（仅适用于Slave_TIMER）
<i>SHRTIMER_ST_INT_CHyONA(y=0,1)</i>	通道y(y=0,1)输出复位请求中断（仅适用于Slave_TIMER）
<i>SHRTIMER_ST_INT_CNTRST</i>	计数器复位中断（仅适用于Slave_TIMER）
<i>SHRTIMER_ST_INT_DLYIDLE</i>	延迟空闲模式进入中断（仅适用于Slave_TIMER）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the Master_TIMER and Slave_TIMER interrupt */
```

```
shrtimer_timers_interrupt_disable(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_INT_CMP0);
```

### 函数 shrtimer\_timers\_interrupt\_flag\_get

函数shrtimer\_timers\_interrupt\_flag\_get描述见下表：

表 3-565. 函数 `shrtimer_timers_interrupt_flag_get`

函数名称	<code>shrtimer_timers_interrupt_flag_get</code>
函数原型	<code>FlagStatus shrtimer_timers_interrupt_flag_get(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t interrupt);</code>
功能描述	获取Master_TIMER/Slave_TIMER中断标志
先决条件	-
被调用函数	-
输入参数{in}	
<code>shrtimer_periph</code>	SHRTIMER外设
<code>SHRTIMERx(x=0)</code>	选择一个SHRTIMER外设
输入参数{in}	
<code>timer_id</code>	选择SHRTIMER中的一个定时器
<code>SHRTIMER_MASTER_TIMER</code>	选择Master_TIMER
<code>SHRTIMER_SLAVE_TIMERx</code>	选择Slave_TIMERx(x=0..4)
输入参数{in}	
<code>interrupt</code>	中断源
<code>SHRTIMER_MT_ST_INT_FLAG_CMPy(y=0..3)</code>	比较y(y=0..3)中断标志（适用于Master_TIMER和Slave_TIMER）
<code>SHRTIMER_MT_ST_INT_FLAG_REP</code>	重复中断标志（适用于Master_TIMER和Slave_TIMER）
<code>SHRTIMER_MT_INT_FLAG_SYNC</code>	同步输入中断标志（仅适用于Master_TIMER）
<code>SHRTIMER_MT_ST_INT_FLAG_UPD</code>	更新中断标志（适用于Master_TIMER和Slave_TIMER）
<code>SHRTIMER_ST_INT_FLAG_CAPy(y=0,1)</code>	捕获y(y=0,1)中断标志（仅适用于Slave_TIMER）
<code>SHRTIMER_ST_INT_FLAG_CHyOA(y=0,1)</code>	通道y(y=0,1)输出置位请求中断标志（仅适用于Slave_TIMER）
<code>SHRTIMER_ST_INT_FLAG_CHyONA(y=0,1)</code>	通道y(y=0,1)输出复位请求中断标志（仅适用于Slave_TIMER）
<code>SHRTIMER_ST_INT_FLAG_CNTRST</code>	计数器复位中断标志（仅适用于Slave_TIMER）
<code>SHRTIMER_ST_INT_FLAG_DLYIDLE</code>	延迟空闲模式进入中断标志（仅适用于Slave_TIMER）
输出参数{out}	
-	-
返回值	
<code>FlagStatus</code>	SET or RESET

例如：

```
/* get the Master_TIMER and Slave_TIMER interrupt flag */
```

```
FlagStatus flag = RESET;
```

```
flag = shrtimer_timers_interrupt_flag_get(SHRTIMER0, SHRTIMER_MASTER_TIMER,  
SHRTIMER_MT_ST_INT_FLAG_CMP0);
```

### 函数 shrtimer\_timers\_interrupt\_flag\_clear

函数 shrtimer\_timers\_interrupt\_flag\_clear 描述见下表：

**表 3-566. 函数 shrtimer\_timers\_interrupt\_flag\_clear**

函数名称	shrtimer_timers_interrupt_flag_clear
函数原型	void shrtimer_timers_interrupt_flag_clear(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t interrupt);
功能描述	清除Master_TIMER/Slave_TIMER中断标志
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
timer_id	选择SHRTIMER中的一个定时器
SHRTIMER_MASTER_TIMER	选择Master_TIMER
SHRTIMER_SLAVE_TIMERx	选择Slave_TIMERx(x=0..4)
输入参数{in}	
interrupt	中断源
SHRTIMER_MT_ST_INT_FLAG_CMPy(y=0..3)	比较y(y=0..3)中断标志（适用于Master_TIMER和Slave_TIMER）
SHRTIMER_MT_ST_INT_FLAG_REP	重复中断标志（适用于Master_TIMER和Slave_TIMER）
SHRTIMER_MT_INT_FLAG_SYNC	同步输入中断标志（仅适用于Master_TIMER）
SHRTIMER_MT_ST_INT_FLAG_UPD	更新中断标志（适用于Master_TIMER和Slave_TIMER）
SHRTIMER_ST_INT_FLAG_CAPy(y=0,1)	捕获y(y=0,1)中断标志（仅适用于Slave_TIMER）
SHRTIMER_ST_INT_FLAG_CHyOA(y=0,1)	通道y(y=0,1)输出置位请求中断标志（仅适用于Slave_TIMER）
SHRTIMER_ST_INT_FLAG_CHyONA(y=0,1)	通道y(y=0,1)输出复位请求中断标志（仅适用于Slave_TIMER）
SHRTIMER_ST_INT_FLAG_CNT	计数器复位中断标志（仅适用于Slave_TIMER）

LAG_CNTRST	
SHRTIMER_ST_INT_F LAG_DLYIDLE	延迟空闲模式进入中断标志（仅适用于Slave_TIMER）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the Master_TIMER and Slave_TIMER interrupt flag */
```

```
shrtimer_timers_interrupt_flag_clear(SHRTIMER0, SHRTIMER_MASTER_TIMER,  
SHRTIMER_MT_ST_INT_FLAG_CMP0);
```

### 函数 shrtimer\_common\_interrupt\_enable

函数shrtimer\_common\_interrupt\_enable描述见下表：

表 3-567. 函数 shrtimer\_common\_interrupt\_enable

函数名称	shrtimer_common_interrupt_enable
函数原型	void shrtimer_common_interrupt_enable(uint32_t shrtimer_periph, uint32_t interrupt);
功能描述	使能SHRTIMER通用中断
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
interrupt	中断源
SHRTIMER_INT_FLTy( y=0..3)	故障y(y=0..3)中断
SHRTIMER_INT_SYSF LT	系统故障中断
SHRTIMER_INT_DLLC AL	DLL校准完成中断
SHRTIMER_INT_BMP ER	突发模式周期中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SHRTIMER common interrupt */
```

```
shrtimer_common_interrupt_enable(SHRTIMER0, SHRTIMER_INT_FLT0);
```

### 函数 shrtimer\_common\_interrupt\_disable

函数shrtimer\_common\_interrupt\_disable描述见下表:

表 3-568. 函数 shrtimer\_common\_interrupt\_disable

函数名称	shrtimer_common_interrupt_disable
函数原型	void shrtimer_common_interrupt_disable(uint32_t shrtimer_periph, uint32_t interrupt);
功能描述	禁能SHRTIMER通用中断
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
interrupt	中断源
SHRTIMER_INT_FLTy( y=0..3)	故障y(y=0..3)中断
SHRTIMER_INT_SYSF LT	系统故障中断
SHRTIMER_INT_DLLC AL	DLL校准完成中断
SHRTIMER_INT_BMP ER	突发模式周期中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SHRTIMER common interrupt */
```

```
shrtimer_common_interrupt_disable(SHRTIMER0, SHRTIMER_INT_FLT0);
```

### 函数 shrtimer\_common\_interrupt\_flag\_get

函数shrtimer\_common\_interrupt\_flag\_get描述见下表:

表 3-569. 函数 shrtimer\_common\_interrupt\_flag\_get

函数名称	shrtimer_common_interrupt_flag_get
函数原型	FlagStatus shrtimer_common_interrupt_flag_get(uint32_t shrtimer_periph, uint32_t interrupt);

功能描述	获取SHRTIMER通用中断标志
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
interrupt	中断标志源
SHRTIMER_INT _FLAG_FLTy(y=0..3)	故障y(y=0..3)中断标志
SHRTIMER_INT _FLAG_SYSFLT	系统故障中断标志
SHRTIMER_INT _FLAG_DLLCAL	DLL校准完成中断标志
SHRTIMER_INT _FLAG_BMPER	突发模式周期中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如:

```
/* get the common interrupt flag */
```

```
FlagStatus flag = RESET;
```

```
flag = shrtimer_common_interrupt_flag_get(SHRTIMER0, SHRTIMER_INT_FLAG_FLT0);
```

### 函数 shrtimer\_common\_interrupt\_flag\_clear

函数shrtimer\_common\_interrupt\_flag\_clear描述见下表:

表 3-570. 函数 shrtimer\_common\_interrupt\_flag\_clear

函数名称	shrtimer_common_interrupt_flag_clear
函数原型	void shrtimer_common_interrupt_flag_clear(uint32_t shrtimer_periph, uint32_t interrupt);
功能描述	清除SHRTIMER通用中断标志
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_periph	SHRTIMER外设
SHRTIMERx(x=0)	选择一个SHRTIMER外设
输入参数{in}	
interrupt	中断标志源
SHRTIMER_INT	故障y(y=0..3)中断标志

<code>_FLAG_FLTy(y=0..3)</code>	
<code>SHRTIMER_INT_FLAG_SYSFLT</code>	系统故障中断标志
<code>SHRTIMER_INT_FLAG_DLLCAL</code>	DLL校准完成中断标志
<code>SHRTIMER_INT_FLAG_BMPER</code>	突发模式周期中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the common interrupt flag */
```

```
shrtimer_common_interrupt_flag_clear(SHRTIMER0, SHRTIMER_INT_FLAG_FLT0);
```

## 3.18. 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[3.18.1](#)描述了I2C的寄存器列表，章节[3.18.2](#)对I2C库函数进行说明。

### 3.18.1. 外设寄存器说明

I2C寄存器列表如下表所示：

表 3-571. 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_DATA	传输缓冲区寄存器
I2C_STAT0	传输状态寄存器0
I2C_STAT1	传输状态寄存器1
I2C_CKCFG	时钟配置寄存器
I2C_RT	上升时间寄存器
I2C_SAMCS	SAM控制状态寄存器
I2C_CTL2	控制寄存器2
I2C_CS	控制状态寄存器
I2C_STATC	状态清除寄存器
I2C2_CTL0	I2C2控制寄存器0
I2C2_CTL1	I2C2控制寄存器1

寄存器名称	寄存器描述
I2C2_SADDR0	I2C2从机地址寄存器0
I2C2_SADDR1	I2C2从机地址寄存器1
I2C2_TIMING	时序寄存器
I2C2_TIMEOUT	超时寄存器
I2C2_STAT	状态寄存器
I2C2_STATC	状态清除寄存器
I2C2_PEC	PEC寄存器
I2C2_RDATA	接收数据寄存器
I2C2_TDATA	发送数据寄存器

### 3.18.2. 外设库函数说明

I2C库函数列表如下表所示：

**表 3-572. I2C 库函数**

库函数名称	库函数描述
i2c_deinit	复位外设I2C
i2c_enable	使能I2C模块
i2c_disable	关闭I2C模块
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_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_data_transmit	发送数据
i2c_data_receive	接收数据
i2c_pec_transfer	传输PEC值使能
i2c_pec_enable	报文错误校验使能
i2c_pec_disable	报文错误校验使禁能
i2c_pec_value_get	获取报文错误校验值
i2c_clock_config	配置I2C时钟
i2c_mode_addr_config	配置I2C地址
i2c_smbus_type_config	SMBus类型选择
i2c_ack_config	是否发送ACK
i2c_ackpos_config	ACK位置配置
i2c_master_addressing	主机发送从机地址
i2c_dualaddr_enable	双地址模式使能
i2c_dualaddr_disable	双地址模式禁能
i2c_dma_config	配置I2C DMA模式
i2c_dma_last_transfer_config	配置下一个DMA EOT是最后传输
i2c_software_reset_config	配置I2C软件复位

库函数名称	库函数描述
i2c_smbus_alert_config	配置通过SMBA引脚发送警告
i2c_smbus_arp_config	配置SMBus下ARP协议
i2c_sam_enable	使能SAM_V接口
i2c_sam_disable	关闭SAM_V接口
i2c_sam_timeout_enable	使能SAM_V接口超时检测
i2c_sam_timeout_disable	关闭SAM_V接口超时检测
i2c_start_early_termination_mode_config	配置提前终止模式
i2c_timeout_calculation_enable	使能I2C超时计数
i2c_timeout_calculation_disable	禁能I2C超时计数
i2c_record_received_slave_address_enable	I2C接收到的从机地址记录在I2C_DATA寄存器中
i2c_record_received_slave_address_disable	I2C接收到的从机地址不记录在I2C_DATA寄存器中
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_status_clear_enable	状态寄存器清除使能
i2c_status_clear_disable	状态寄存器清除禁能
i2c_status_bit_clear	清除I2C状态寄存器位
i2c_flag_get	获取I2C标志位
i2c_flag_clear	清除I2C标志位
i2c_interrupt_enable	使能I2C中断
i2c_interrupt_disable	禁能I2C中断
i2c_interrupt_flag_get	获取I2C中断标志位
i2c_interrupt_flag_clear	清除I2C中断标志位
i2c_timing_config	配置时序参数
i2c_digital_noise_filter_config	配置数字噪声滤波器
i2c_analog_noise_filter_enable	使能模拟噪声滤波器
i2c_analog_noise_filter_disable	禁能模拟噪声滤波器
i2c_wakeup_from_deepsleep_enable	使能从Deep-sleep模式中唤醒
i2c_wakeup_from_deepsleep_disable	禁止从Deep-sleep模式中唤醒
i2c_master_clock_config	配置主机模式下SCL高低电平周期
i2c2_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_address_config	配置I2C从机地址
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

库函数名称	库函数描述
i2c_second_address_disable	禁能I2C从机第二个地址
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_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
i2c2_dma_enable	使能发送/接收模式下DMA
i2c2_dma_disable	禁能发送/接收模式下DMA
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
i2c2_flag_get	获取I2C标志位
i2c2_flag_clear	清除I2C标志位
i2c2_interrupt_enable	使能I2C中断
i2c2_interrupt_disable	禁能I2C中断
i2c2_interrupt_flag_get	获取I2C中断标志位
i2c2_interrupt_flag_clear	清除I2C中断标志位

## 枚举类型 i2c\_flag\_enum

表 3-573. 枚举类型 i2c\_flag\_enum

成员名称	功能描述
I2C_FLAG_SBSEND	起始位是否发送
I2C_FLAG_ADDSEND	主机模式下地址是否发送/从机模式下地址是否匹配
I2C_FLAG_BTC	字节传输完成
I2C_FLAG_ADD10SEND	主机模式下10位地址地址头发送完成
I2C_FLAG_STPDET	从机模式下监测到STOP结束位
I2C_FLAG_RBNE	接收期间I2C_DATA非空
I2C_FLAG_TBNE	发送期间I2C_DATA为空

成员名称	功能描述
I2C_FLAG_BERR	总线错误，表示I2C总线上发生了预料之外的START起始位或STOP结束位
I2C_FLAG_LOSTARB	主机模式下仲裁丢失
I2C_FLAG_AERR	应答错误
I2C_FLAG_OUERR	当禁用SCL拉低功能后，在从机模式下发生了过载或欠载事件
I2C_FLAG_PECERR	接收数据时PEC错误
I2C_FLAG_SMBTO	SMBus模式下超时信号
I2C_FLAG_SMBALT	SMBus警报状态
I2C_FLAG_MASTER	表明I2C时钟在主机模式还是从机模式的标志位
I2C_FLAG_I2CBSY	忙标志
I2C_FLAG_TR	I2C作发送端还是接收端
I2C_FLAG_RXGC	是否接收到广播地址(00h)
I2C_FLAG_DEFSMB	从机模式下SMBus主机地址头
I2C_FLAG_HSTSMB	从机模式下监测到SMBus主机地址头
I2C_FLAG_DUMOD	从机模式下双标志位表明哪个地址和双地址模式匹配
I2C_FLAG_TFF	发送帧下降沿标志
I2C_FLAG_TFR	发送帧上升沿标志
I2C_FLAG_RFF	接收帧下降沿标志
I2C_FLAG_RFR	接收帧上升沿标志
I2C_FLAG_STLO	起始信号丢失
I2C_FLAG_STPSEND	主机模式下停止信号发送

### 枚举类型 i2c\_interrupt\_enum

表 3-574. 枚举类型 i2c\_interrupt\_enum

成员名称	功能描述
I2C_INT_ERR	错误中断使能
I2C_INT_EV	事件中断使能
I2C_INT_BUF	缓冲区中断使能
I2C_INT_TFF	发送帧下降沿中断使能
I2C_INT_TFR	发送帧上升沿中断使能
I2C_INT_RFF	接收帧下降沿中断使能
I2C_INT_RFR	接收帧上升沿中断使能
I2C_INT_STLO	起始信号丢失中断使能
I2C_INT_STPSEND	主机模式下停止信号发送中断使能

### 枚举类型 i2c\_interrupt\_flag\_enum

表 3-575. 枚举类型 i2c\_interrupt\_flag\_enum

成员名称	功能描述
I2C_INT_FLAG_SBSSEND	主机模式下发送START起始位
I2C_INT_FLAG_ADDSEND	主机模式下成功发送了地址 / 从机模式下接收到了地址并且和自

成员名称	功能描述
	身的地址匹配
I2C_INT_FLAG_BTC	字节发送结束
I2C_INT_FLAG_ADD10SEND	主机模式下10位地址地址头被发送
I2C_INT_FLAG_STPDET	从机模式下监测到STOP结束位
I2C_INT_FLAG_RBNE	接收期间I2C_DATA非空
I2C_INT_FLAG_TBE	发送期间I2C_DATA为空
I2C_INT_FLAG_BERR	总线错误
I2C_INT_FLAG_LOSTARB	主机模式下仲裁丢失
I2C_INT_FLAG_AERR	应答错误
I2C_INT_FLAG_OUERR	当禁用SCL拉低功能后，在从机模式下发生了过载或欠载事件
I2C_INT_FLAG_PECERR	接收数据时PEC错误
I2C_INT_FLAG_SMBTO	SMBus模式下超时信号
I2C_INT_FLAG_SMBALT	SMBus警报状态
I2C_INT_FLAG_TFF	发送帧下降沿中断标志位
I2C_INT_FLAG_TFR	发送帧上升沿中断标志位
I2C_INT_FLAG_RFF	接收帧下降沿中断标志位
I2C_INT_FLAG_RFR	接收帧上升沿中断标志位
I2C_INT_FLAG_STLO	起始信号丢失中断标志位
I2C_INT_FLAG_STPSEND	主机模式下停止信号发送中断标志位

### 枚举类型 i2c2\_interrupt\_flag\_enum

表 3-576. 枚举类型 i2c2\_interrupt\_flag\_enum

成员名称	功能描述
I2C2_INT_FLAG_TI	发送中断标志
I2C2_INT_FLAG_RBNE	接收期间I2C2_RDATA非空中断标志
I2C2_INT_FLAG_ADDSEND	从机模式下，接收到的地址与自身地址匹配中断标志
I2C2_INT_FLAG_NACK	NACK中断标志
I2C2_INT_FLAG_STPDET	从机模式下检测到STOP信号中断标志
I2C2_INT_FLAG_TC	主机模式下传输完成中断标志
I2C2_INT_FLAG_TCR	传输完成重载中断标志
I2C2_INT_FLAG_BERR	总线错误中断标志
I2C2_INT_FLAG_LOSTARB	仲裁丢失中断标志
I2C2_INT_FLAG_OUERR	从机模式下，过载/欠载错误中断标志
I2C2_INT_FLAG_PECERR	PEC错误中断标志
I2C2_INT_FLAG_TIMEOUT	超时中断标志
I2C2_INT_FLAG_SMBALT	SMBus报警中断标志

### 函数 i2c\_deinit

函数i2c\_deinit描述见下表：

表 3-577. 函数 i2c\_deinit

函数名称	i2c_deinit
函数原型	void i2c_deinit(uint32_t i2c_periph);
功能描述	复位外设I2C
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset I2C0 */
i2c_deinit (I2C0);
```

### 函数 i2c\_enable

函数i2c\_enable描述见下表:

表 3-578. 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模块
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
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 3-579. 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	关闭I2C模块
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
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 3-580. 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=0,1,2)
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 3-581. 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=0,1,2)
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\_slave\_response\_to\_gcall\_enable

函数i2c\_slave\_response\_to\_gcall\_enable描述见下表：

表 3-582. 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=0,1,2)
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 3-583. 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=0,1,2)
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 3-584. 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=0,1,2)
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 3-585. 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=0,1,2)
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\_data\_transmit

函数i2c\_data\_transmit描述见下表：

表 3-586. 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=0,1,2)
输入参数{in}	
data	待发送数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 transmit data */
```

```
i2c_data_transmit (I2C0, 0x55);
```

**函数 i2c\_data\_receive**

函数i2c\_data\_receive描述见下表：

**表 3-587. 函数 i2c\_data\_receive**

函数名称	i2c_data_receive
函数原型	uint8_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=0,1,2)
输出参数{out}	
-	-
返回值	
uint32_t	0x00..0xFF

例如：

```
/* I2C0 receive data */
```

```
uint32_t i2c_receiver;
```

```
i2c_receiver = i2c_data_receive(I2C0);
```

**函数 i2c\_pec\_transfer**

函数i2c\_pec\_transfer描述见下表：

**表 3-588. 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	传输 PEC 值使能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2)
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 3-589. 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	报文错误校验使能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2)
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 3-590. 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	报文错误校验使禁能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2)
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 3-591. 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=0,1,2)
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\_clock\_config

函数i2c\_clock\_config描述见下表:

表 3-592. 函数 i2c\_clock\_config

函数名称	i2c_clock_config
函数原型	void i2c_clock_config(uint32_t i2c_periph, uint32_t clk speed, uint32_t dutycyc);
功能描述	配置 I2C 时钟
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
clk speed	i2c 时钟速率
输入参数{in}	
dutycyc	快速模式下占空比
I2C_DTCY_2	T_low/T_high=2

<i>I2C_DTCY_16_9</i>	T_low/T_high=16/9
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure I2C0 clock speed as 100KHz*/
```

```
i2c_clock_config(I2C0, 100000, I2C_DTCY_2);
```

### 函数 i2c\_mode\_addr\_config

函数i2c\_mode\_addr\_config描述见下表：

表 3-593. 函数 i2c\_mode\_addr\_config

函数名称	i2c_mode_addr_config
函数原型	void i2c_mode_addr_config(uint32_t i2c_periph, uint32_t mode, uint32_t addformat, uint32_t addr);
功能描述	配置I2C地址
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>mode</b>	I2C模式选择
<i>I2C_I2CMODE_ENABLER</i>	I2C模式
<i>I2C_SMBUSMODE_ENABLED</i>	SMBus模式
输入参数{in}	
<b>addformat</b>	7位或10位
<i>I2C_ADDFORMAT_7BITS</i>	7位
<i>I2C_ADDFORMAT_10BITS</i>	10位
输入参数{in}	
<b>addr</b>	I2C地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure I2C0 address as 0x82, using 7 bits */
```

```
i2c_mode_addr_config(I2C0, I2C_I2CMODE_ENABLE, I2C_ADDFORMAT_7BITS, 0x82);
```

### 函数 i2c\_smbus\_type\_config

函数i2c\_smbus\_type\_config描述见下表:

表 3-594. 函数 i2c\_smbus\_type\_config

函数名称	i2c_smbus_type_config
函数原型	void i2c_smbus_type_config(uint32_t i2c_periph, uint32_t type);
功能描述	SMBus类型选择
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}	
type	主机或从机
I2C_SMBUS_DEVICE	从机
I2C_SMBUS_HOST	主机
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* config I2C0 as SMBUS host type*/
```

```
i2c_smbus_type_config (I2C0, I2C_SMBUS_HOST);
```

### 函数 i2c\_ack\_config

函数i2c\_ack\_config描述见下表:

表 3-595. 函数 i2c\_ack\_config

函数名称	i2c_ack_config
函数原型	void i2c_ack_config(uint32_t i2c_periph, uint32_t ack);
功能描述	是否发送ACK
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
ack	I2C外设

<i>I2C_ACK_ENABLE</i>	ACK会被发送
<i>I2C_ACK_DISABLE</i>	ACK不会被发送
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 will send ACK */
```

```
i2c_ack_config (I2C0, I2C_ACK_ENABLE);
```

### 函数 i2c\_ackpos\_config

函数i2c\_ackpos\_config描述见下表：

表 3-596. 函数 i2c\_ackpos\_config

函数名称	i2c_ackpos_config
函数原型	void i2c_ackpos_config(uint32_t i2c_periph, uint32_t pos);
功能描述	ACK位置配置
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>pos</b>	ACK位置
<i>I2C_ACKPOS_CURRENT</i>	当前正在接收的字节是否发送ACK
<i>I2C_ACKPOS_NEXT</i>	下一个接收的字节是否发送ACK
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* The ACK of I2C0 is send for the current frame*/
```

```
i2c_ackpos_config (I2C0, I2C_ACKPOS_CURRENT);
```

### 函数 i2c\_master\_addressing

函数i2c\_master\_addressing描述见下表：

表 3-597. 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 addr, uint32_t trandirection);
功能描述	主机发送从机地址
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}	
addr	从机地址
输入参数{in}	
trandirection	发送或接收
I2C_TRANSMITTER	发送
I2C_RECEIVER	接收
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* send slave address to I2C bus and I2C0 act as receiver */
```

```
i2c_master_addressing(I2C0, 0x82, I2C_RECEIVER);
```

### 函数 i2c\_dualaddr\_enable

函数i2c\_dualaddr\_enable描述见下表：

表 3-598. 函数 i2c\_dualaddr\_enable

函数名称	i2c_dualaddr_enable
函数原型	void i2c_dualaddr_enable(uint32_t i2c_periph, uint32_t dualaddr);
功能描述	双地址模式使能
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}	
addr	双地址模式下第二个地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 dual-address */
```

```
i2c_dualaddr_enable (I2C0, 0x82);
```

### 函数 i2c\_dualaddr\_disable

函数i2c\_dualaddr\_disable描述见下表:

**表 3-599. 函数 i2c\_dualaddr\_disable**

函数名称	i2c_dualaddr_disable
函数原型	void i2c_dualaddr_disable(uint32_t i2c_periph);
功能描述	双地址模式禁能
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 dual-address mode */
```

```
i2c_dualaddr_disable (I2C0);
```

### 函数 i2c\_dma\_config

函数i2c\_dma\_config描述见下表:

**表 3-600. 函数 i2c\_dma\_config**

函数名称	i2c_dma_config
函数原型	void i2c_dma_config(uint32_t i2c_periph, uint32_t dmastate);
功能描述	配置I2C DMA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)
输入参数{in}	
dmastate	开启或关闭
I2C_DMA_ON	开启DMA模式
I2C_DMA_OFF	关闭DMA模式
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* I2C0 DMA mode enable */
```

```
i2c_dma_config(I2C0, I2C_DMA_ON);
```

### 函数 i2c\_dma\_last\_transfer\_config

函数i2c\_dma\_last\_transfer\_config描述见下表：

**表 3-601. 函数 i2c\_dma\_last\_transfer\_config**

函数名称	i2c_dma_last_transfer_config
函数原型	void i2c_dma_last_transfer_config (uint32_t i2c_periph, uint32_t dmalast);
功能描述	配置下一个DMA EOT是最后传输
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}	
dmalast	是否使能下一个DMA EOT是最后传输
I2C_DMALST_ON	下一个DMA EOT是最后传输
I2C_DMALST_OFF	下一个DMA EOT不是最后传输
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* next DMA EOT is the last transfer */
```

```
i2c_dma_last_transfer_enable (I2C0, I2C_DMALST_ON);
```

### 函数 i2c\_software\_reset\_config

函数i2c\_software\_reset\_config描述见下表：

**表 3-602. 函数 i2c\_software\_reset\_config**

函数名称	i2c_software_reset_config
函数原型	void i2c_software_reset_config(uint32_t i2c_periph, uint32_t sreset);
功能描述	配置I2C软件复位
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>sreset</b>	是否复位
<i>I2C_SRESET_SET</i>	复位
<i>I2C_SRESET_RESET</i>	没有复位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* software reset I2C0 */
```

```
i2c_software_reset_config (I2C0, I2C_SRESET_SET);
```

### 函数 i2c\_smbus\_alert\_config

函数i2c\_smbus\_alert\_config描述见下表：

表 3-603. 函数 i2c\_smbus\_alert\_config

函数名称	i2c_smbus_alert_config
函数原型	void i2c_smbus_alert_config(uint32_t i2c_periph, uint32_t smbuspara);
功能描述	通过SMBA引脚发送警告
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>smbuspara</b>	是否通过SMBA引脚发送警告
<i>I2C_SALTSEND_ENABLE</i>	通过SMBA引脚发送警告
<i>I2C_SALTSEND_DISABLE</i>	不通过SMBA引脚发送警告
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 issue alert through SMBA pin enable*/
```

```
i2c_smbus_alert_config (I2C0, I2C_SALTSEND_ENABLE);
```

## 函数 i2c\_smbus\_arp\_config

函数i2c\_smbus\_arp\_config描述见下表:

表 3-604. 函数 i2c\_smbus\_arp\_config

函数名称	i2c_smbus_arp_config
函数原型	void i2c_smbus_arp_config(uint32_t i2c_periph, uint32_t arpstate);
功能描述	SMBus下ARP协议是否开启
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}	
arpstate	SMBus下ARP协议是否开启
I2C_ARP_ENABLE	使能ARP
I2C_ARP_DISABLE	关闭ARP
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable I2C0 ARP protocol in SMBus switch */
```

```
i2c_smbus_arp_config (I2C0, I2C_ARP_ENABLE);
```

## 函数 i2c\_sam\_enable

函数i2c\_sam\_enable描述见下表:

表 3-605. 函数 i2c\_sam\_enable

函数名称	i2c_sam_enable
函数原型	void i2c_sam_enable(uint32_t i2c_periph);
功能描述	使能SAM_V接口
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 SAM_V interface */
```

```
i2c_sam_enable (I2C0);
```

### 函数 i2c\_sam\_disable

函数i2c\_sam\_disable描述见下表：

表 3-606. 函数 i2c\_sam\_disable

函数名称	i2c_sam_disable
函数原型	void i2c_sam_disable(uint32_t i2c_periph);
功能描述	关闭SAM_V接口
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 SAM_V interface */
```

```
i2c_sam_disable (I2C0);
```

### 函数 i2c\_sam\_timeout\_enable

函数i2c\_sam\_timeout\_enable描述见下表：

表 3-607. 函数 i2c\_sam\_timeout\_enable

函数名称	i2c_sam_timeout_enable
函数原型	void i2c_sam_timeout_enable(uint32_t i2c_periph);
功能描述	使能SAM_V接口超时检测
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 SAM_V interface timeout detect */
```

```
i2c_sam_timeout_enable (I2C0);
```

### 函数 i2c\_sam\_timeout\_disable

函数i2c\_sam\_timeout\_disable描述见下表：

表 3-608. 函数 i2c\_sam\_timeout\_disable

函数名称	i2c_sam_timeout_disable
函数原型	void i2c_sam_timeout_disable(uint32_t i2c_periph);
功能描述	关闭SAM_V接口超时检测
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 SAM_V interface timeout detect */
```

```
i2c_sam_timeout_disable (I2C0);
```

### 函数 i2c\_start\_early\_termination\_mode\_config

函数i2c\_start\_early\_termination\_mode\_config描述见下表：

表 3-609. 函数 i2c\_start\_early\_termination\_mode\_config

函数名称	i2c_start_early_termination_mode_config
函数原型	void i2c_start_early_termination_mode_config(uint32_t i2c_periph, uint32_t mode);
功能描述	配置提前终止模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
mode	启动提前终止模式
STANDARD_I2C_PRO	与标准I2C协议一样操作

TOCOL_MODE	
ARBITRATION_LOST_MODE	与仲裁丢失一样操作
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure I2C start early termination mode */
```

```
i2c_start_early_termination_mode_config (I2C0, ARBITRATION_LOST_MODE);
```

### 函数 i2c\_timeout\_calculation\_enable

函数i2c\_timeout\_calculation\_enable描述见下表：

表 3-610. 函数 i2c\_timeout\_calculation\_enable

函数名称	i2c_timeout_calculation_enable
函数原型	void i2c_timeout_calculation_enable(uint32_t i2c_periph);
功能描述	使能I2C超时计数
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable i2c timeout calculation */
```

```
i2c_timeout_calculation_enable (I2C0);
```

### 函数 i2c\_timeout\_calculation\_disable

函数i2c\_timeout\_calculation\_disable描述见下表：

表 3-611. 函数 i2c\_timeout\_calculation\_disable

函数名称	i2c_timeout_calculation_disable
函数原型	void i2c_timeout_calculation_disable(uint32_t i2c_periph);
功能描述	禁能I2C超时计数
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：

```
/* disable i2c timeout calculation */
```

```
i2c_timeout_calculation_disable (I2C0);
```

### 函数 i2c\_record\_received\_slave\_address\_enable

函数i2c\_record\_received\_slave\_address\_enable描述见下表：

表 3-612. 函数 i2c\_record\_received\_slave\_address\_enable

函数名称	i2c_record_received_slave_address_enable
函数原型	void i2c_record_received_slave_address_enable(uint32_t i2c_periph);
功能描述	I2C接收到的从机地址记录在I2C_DATA寄存器中
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 i2c record the received slave address to the transfer buffer register */
```

```
i2c_record_received_slave_address_enable (I2C0);
```

### 函数 i2c\_record\_received\_slave\_address\_disable

函数i2c\_record\_received\_slave\_address\_disable描述见下表：

表 3-613. 函数 i2c\_record\_received\_slave\_address\_disable

函数名称	i2c_record_received_slave_address_disable
函数原型	void i2c_record_received_slave_address_disable(uint32_t i2c_periph);
功能描述	I2C接收到的从机地址不记录在I2C_DATA寄存器中
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：

```
/* disable i2c record the received slave address to the transfer buffer register */
```

```
i2c_record_received_slave_address_disable (I2C0);
```

### 函数 i2c\_address\_bit\_compare\_config

函数i2c\_address\_bit\_compare\_config描述见下表：

表 3-614. 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, uint16_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}	
<b>i2c_periph</b>	I2C外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
<b>mode</b>	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_BIT7_COMPARE);
```

### 函数 i2c\_status\_clear\_enable

函数i2c\_status\_clear\_enable描述见下表：

表 3-615. 函数 i2c\_status\_clear\_enable

函数名称	i2c_status_clear_enable
函数原型	void i2c_status_clear_enable(uint32_t i2c_periph);
功能描述	状态寄存器清除使能
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 status register clear */
```

```
i2c_status_clear_enable (I2C0);
```

### 函数 i2c\_status\_clear\_disable

函数i2c\_status\_clear\_disable描述见下表：

表 3-616. 函数 i2c\_status\_clear\_disable

函数名称	i2c_status_clear_disable
函数原型	void i2c_status_clear_disable(uint32_t i2c_periph);
功能描述	状态寄存器清除禁能
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 status register clear */
```

```
i2c_status_clear_disable (I2C0);
```

### 函数 i2c\_status\_bit\_clear

函数i2c\_status\_bit\_clear描述见下表:

表 3-617. 函数 i2c\_start\_early\_termination\_mode\_config

函数名称	i2c_status_bit_clear
函数原型	void i2c_status_bit_clear(uint32_t i2c_periph, uint32_t clear_bit);
功能描述	清除I2C状态寄存器位
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}	
clear_bit	选择清除哪一位
CLEAR_STPDET	清除I2C_STAT0寄存器中STPDET位
CLEAR_ADD10SEND	清除I2C_STAT0寄存器中ADD10SEND位
CLEAR_BTC	清除I2C_STAT0寄存器中BTC位
CLEAR_ADDSEND	清除I2C_STAT0寄存器中ADDSEND位
CLEAR_SBSSEND	清除I2C_STAT0寄存器中SBSSEND位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear i2c status register bit */
```

```
i2c_status_bit_clear (I2C0, CLEAR_ADDSEND);
```

### 函数 i2c\_flag\_get

函数i2c\_flag\_get描述见下表:

表 3-618. 函数 i2c\_flag\_get

函数名称	i2c_flag_get
函数原型	FlagStatus i2c_flag_get(uint32_t i2c_periph, i2c_flag_enum 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	需要获取的标志位，参考 <a href="#">表3-573. 枚举类型i2c_flag_enum</a> 。
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

```
/* check whether start condition send out */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_flag_get (I2C0, I2C_FLAG_SBSEND);
```

### 函数 i2c\_flag\_clear

函数i2c\_flag\_clear描述见下表：

表 3-619. 函数 i2c\_flag\_clear

函数名称	i2c_flag_clear
函数原型	void i2c_flag_clear(uint32_t i2c_periph, i2c_flag_enum 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}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
flag	标志类型，参考 <a href="#">表3-573. 枚举类型i2c_flag_enum</a> 。
I2C_FLAG_SMBTO	SMBus模式下超时信号
I2C_FLAG_SMBALT	SMBus警报状态
I2C_FLAG_PECERR	接收数据时PEC错误
I2C_FLAG_OUERR	当禁用SCL拉低功能后，在从机模式下发生了过载或欠载事件
I2C_FLAG_AERR	应答错误
I2C_FLAG_LOSTARB	主机模式下仲裁丢失
I2C_FLAG_BERR	总线错误
I2C_FLAG_ADDSEND	主机模式下地址是否发送/从机模式下地址是否匹配
I2C_FLAG_TFF	发送帧下降沿标志

<i>I2C_FLAG_TFR</i>	发送帧上升沿标志
<i>I2C_FLAG_RFF</i>	接收帧下降沿标志
<i>I2C_FLAG_RFR</i>	接收帧上升沿标志
<i>I2C_FLAG_STLO</i>	起始信号丢失
<i>I2C_FLAG_STPSEND</i>	主机模式下停止信号发送
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 3-620. 函数 i2c\_interrupt\_enable

函数名称	i2c_interrupt_enable
函数原型	void i2c_interrupt_enable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);
功能描述	使能I2C中断
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}	
interrupt	中断类型，参考 <a href="#">表3-574. 枚举类型i2c_interrupt_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 error interrupt */
```

```
i2c_interrupt_enable (I2C0, I2C_INT_EV);
```

### 函数 i2c\_interrupt\_disable

函数i2c\_interrupt\_disable描述见下表：

表 3-621. 函数 i2c\_interrupt\_disable

函数名称	i2c_interrupt_disable
------	-----------------------

函数原型	void i2c_interrupt_disable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);
功能描述	禁能I2C中断
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}	
interrupt	中断类型，参考 <a href="#">表3-574. 枚举类型i2c_interrupt_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C0 error interrupt */
```

```
i2c_interrupt_disable (I2C0, I2C_INT_EV);
```

### 函数 i2c\_interrupt\_flag\_get

函数i2c\_interrupt\_flag\_get描述见下表：

**表 3-622. 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, i2c_interrupt_flag_enum int_flag);
功能描述	获取I2C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	中断标志，参考 <a href="#">表3-575. 枚举类型i2c_interrupt_flag_enum</a> 。
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

```
/* check the byte transmission finishes interrupt flag is set or not*/
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_interrupt_flag_get (I2C0, I2C_INT_FLAG_BTC);
```

### 函数 i2c\_interrupt\_flag\_clear

函数i2c\_interrupt\_flag\_clear描述见下表：

表 3-623. 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I2C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	中断标志，参考 <a href="#">表3-575. 枚举类型i2c_interrupt_flag_enum</a> 。
I2C_INT_FLAG_ADDS END	主机模式下成功发送了地址 / 从机模式下接收到了地址并且和自身的地址匹配
I2C_INT_FLAG_BERR	总线错误
I2C_INT_FLAG_LOSTA RB	主机模式下仲裁丢失
I2C_INT_FLAG_AERR	应答错误
I2C_INT_FLAG_OUER R	当禁用SCL拉低功能后，在从机模式下发生了过载或欠载事件
I2C_INT_FLAG_PECERR	接收数据时PEC错误
I2C_INT_FLAG_SMBT O	SMBus模式下超时信号
I2C_INT_FLAG_SMBA LT	SMBus警报状态
I2C_INT_FLAG_TFF	发送帧下降沿中断标志位
I2C_INT_FLAG_TFR	发送帧上升沿中断标志位
I2C_INT_FLAG_RFF	接收帧下降沿中断标志位
I2C_INT_FLAG_RFR	接收帧上升沿中断标志位
I2C_INT_FLAG_STLO	起始信号丢失中断标志位
I2C_INT_FLAG_STPSE ND	主机模式下停止信号发送中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the acknowledge error interrupt flag */
```

```
i2c_interrupt_flag_clear (I2C0, I2C_INT_FLAG_AERR);
```

### 函数 i2c\_timing\_config

函数i2c\_timing\_config描述见下表：

表 3-624. 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=2)
输入参数{in}	
psc	0-0xf, 时序分频
输入参数{in}	
scl_dely	0-0xf, 数据建立时间
输入参数{in}	
sda_dely	0-0xf, 数据保持时间
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the timing parameters */
```

```
i2c_timing_config (I2C2, 0x1, 0x2, 0x1);
```

### 函数 i2c\_digital\_noise\_filter\_config

函数i2c\_digital\_noise\_filter\_config描述见下表：

表 3-625. 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	配置数字噪声滤波器
先决条件	-

被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=2)
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>I2CCLK</sub> 的尖峰
FILTER_LENGTH_2	数字噪声滤波使能并且可以滤除脉宽宽度不大于 2 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_3	数字噪声滤波使能并且可以滤除脉宽宽度不大于 3 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_4	数字噪声滤波使能并且可以滤除脉宽宽度不大于 4 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_5	数字噪声滤波使能并且可以滤除脉宽宽度不大于 5 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_6	数字噪声滤波使能并且可以滤除脉宽宽度不大于 6 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_7	数字噪声滤波使能并且可以滤除脉宽宽度不大于 7 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_8	数字噪声滤波使能并且可以滤除脉宽宽度不大于 8 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_9	数字噪声滤波使能并且可以滤除脉宽宽度不大于 9 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_10	数字噪声滤波使能并且可以滤除脉宽宽度不大于 10 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_11	数字噪声滤波使能并且可以滤除脉宽宽度不大于 11 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_12	数字噪声滤波使能并且可以滤除脉宽宽度不大于 12 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_13	数字噪声滤波使能并且可以滤除脉宽宽度不大于 13 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_14	数字噪声滤波使能并且可以滤除脉宽宽度不大于 14 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_15	数字噪声滤波使能并且可以滤除脉宽宽度不大于 15 t <sub>I2CCLK</sub> 的尖峰
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C2 digital filter filters spikes with a length of up to 1 tI2CCLK */
```

```
i2c_digital_noise_filter_config (I2C2, FILTER_LENGTH_1);
```

### 函数 i2c\_analog\_noise\_filter\_enable

函数 i2c\_analog\_noise\_filter\_enable 描述见下表：

表 3-626. 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}	
i2c_periph	I2C 外设

<i>I2Cx</i>	(x=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable analog noise filter */
```

```
i2c_analog_noise_filter_enable (I2C2);
```

### 函数 i2c\_analog\_noise\_filter\_disable

函数i2c\_analog\_noise\_filter\_disable描述见下表：

表 3-627. 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
<i>I2Cx</i>	(x=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable analog noise filter */
```

```
i2c_analog_noise_filter_disable (I2C2);
```

### 函数 i2c\_wakeup\_from\_deepsleep\_enable

函数i2c\_wakeup\_from\_deepsleep\_enable描述见下表：

表 3-628. 函数 i2c\_wakeup\_from\_deepsleep\_enable

函数名称	i2c_wakeup_from_deepsleep_enable
函数原型	void i2c_wakeup_from_deepsleep_enable(uint32_t i2c_periph);
功能描述	使能从Deep-sleep模式中唤醒
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设

<i>I2Cx</i>	(x=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable wakeup from Deep-sleep mode */
```

```
i2c_wakeup_from_deepsleep_enable (I2C2);
```

### 函数 i2c\_wakeup\_from\_deepsleep\_disable

函数i2c\_wakeup\_from\_deepsleep\_disable描述见下表：

**表 3-629. 函数 i2c\_wakeup\_from\_deepsleep\_disable**

函数名称	i2c_wakeup_from_deepsleep_disable
函数原型	void i2c_wakeup_from_deepsleep_disable(uint32_t i2c_periph);
功能描述	禁止从Deep-sleep模式中唤醒
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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable wakeup from Deep-sleep mode */
```

```
i2c_wakeup_from_deepsleep_disable (I2C2);
```

### 函数 i2c\_master\_clock\_config

函数i2c\_master\_clock\_config描述见下表：

**表 3-630. 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}	

<b>i2c_periph</b>	I2C 外设
<i>I2Cx</i>	(x=2)
输入参数{in}	
<b>sclh</b>	0-0xff, SCL 高电平周期
输入参数{in}	
<b>scll</b>	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 (I2C2, 0x0f, 0x0f);
```

### 函数 i2c2\_master\_addressing

函数i2c2\_master\_addressing描述见下表：

表 3-631. 函数 i2c2\_master\_transfer\_direction\_config

函数名称	i2c2_master_addressing
函数原型	void i2c2_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=2)
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>I2C2_MASTER_TRANSMIT</i>	主机发送
<i>I2C2_MASTER_RECEIVE</i>	主机接收
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* send slave address to I2C bus */
```

```
i2c2_master_addressing (I2C2, 0x82, I2C2_MASTER_TRANSMIT);
```

### 函数 i2c\_address10\_header\_enable

函数i2c\_address10\_header\_enable描述见下表：

**表 3-632. 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=2)
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 (I2C2);
```

### 函数 i2c\_address10\_header\_disable

函数i2c\_address10\_header\_disable描述见下表：

**表 3-633. 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=2)
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 (I2C2);
```

### 函数 i2c\_address10\_enable

函数i2c\_address10\_enable描述见下表：

**表 3-634. 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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable 10-bit addressing mode in master mode */
```

```
i2c_address10_enable (I2C2);
```

### 函数 i2c\_address10\_disable

函数i2c\_address10\_disable描述见下表：

**表 3-635. 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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable 10-bit addressing mode in master mode */
```

```
i2c_address10_disable (I2C2);
```

### 函数 i2c\_automatic\_end\_enable

函数i2c\_automatic\_end\_enable描述见下表：

表 3-636. 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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_enable (I2C2);
```

### 函数 i2c\_automatic\_end\_disable

函数i2c\_automatic\_end\_disable描述见下表：

表 3-637. 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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_disable (I2C2);
```

### 函数 i2c\_address\_config

函数i2c\_address\_config描述见下表：

**表 3-638. 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=2)
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 (I2C2, 0x82, I2C_ADDFORMAT_7BITS);
```

### 函数 i2c\_address\_disable

函数i2c\_address\_disable描述见下表：

**表 3-639. 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}	
<b>i2c_periph</b>	I2C 外设
<i>I2Cx</i>	(x=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable i2c address in slave mode */
```

```
i2c_address_disable (I2C2);
```

### 函数 i2c\_second\_address\_config

函数i2c\_second\_address\_config描述见下表：

表 3-640. 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}	
<b>i2c_periph</b>	I2C 外设
<i>I2Cx</i>	(x=2)
输入参数{in}	
<b>address</b>	I2C 地址
输入参数{in}	
<b>addr_mask</b>	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

T1_6	
ADDRESS2_MASK_AL L	ADDRESS2[7:1]屏蔽
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure i2c second slave address */
```

```
i2c_second_address_config (I2C2, 0x82, ADDRESS2_MASK_BIT1_2);
```

### 函数 i2c\_second\_address\_disable

函数i2c\_second\_address\_disable描述见下表：

表 3-641. 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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable i2c second address in slave mode */
```

```
i2c_second_address_disable (I2C2);
```

### 函数 i2c\_receved\_address\_get

函数i2c\_receved\_address\_get描述见下表：

表 3-642. 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}	
<b>i2c_periph</b>	I2C 外设
<i>I2Cx</i>	(x=2)
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	0x00..0x7F

例如:

```
/* get received match address in slave mode */
```

```
uint32_t address;
```

```
address = i2c_receivied_address_get (I2C2);
```

### 函数 i2c\_slave\_byte\_control\_enable

函数i2c\_slave\_byte\_control\_enable描述见下表:

表 3-643. 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}	
<b>i2c_periph</b>	I2C 外设
<i>I2Cx</i>	(x=2)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable slave byte control */
```

```
i2c_slave_byte_control_enable (I2C2);
```

### 函数 i2c\_slave\_byte\_control\_disable

函数i2c\_slave\_byte\_control\_disable描述见下表:

表 3-644. 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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable slave byte control */
```

```
i2c_slave_byte_control_disable (I2C2);
```

### 函数 i2c\_nack\_enable

函数i2c\_nack\_enable描述见下表：

表 3-645. 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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* generate a NACK in slave mode */
```

```
i2c_nack_enable (I2C2);
```

### 函数 i2c\_nack\_disable

函数i2c\_nack\_disable描述见下表：

表 3-646. 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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* generate a ACK in slave mode */
```

```
i2c_nack_disable (I2C2);
```

### 函数 i2c\_reload\_enable

函数i2c\_reload\_enable描述见下表：

表 3-647. 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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C reload mode */
```

```
i2c_reload_enable (I2C2);
```

### 函数 i2c\_reload\_disable

函数i2c\_reload\_disable描述见下表：

表 3-648. 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}	
<b>i2c_periph</b>	I2C 外设
<i>I2Cx</i>	(x=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C reload mode */
```

```
i2c_reload_disable (I2C2);
```

### 函数 i2c\_transfer\_byte\_number\_config

函数i2c\_transfer\_byte\_number\_config描述见下表：

表 3-649. 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}	
<b>i2c_periph</b>	I2C 外设
<i>I2Cx</i>	(x=2)
输入参数{in}	
<b>byte_number</b>	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 (I2C2, 0xFF);
```

### 函数 i2c2\_dma\_enable

函数i2c2\_dma\_enable描述见下表：

表 3-650. 函数 i2c2\_dma\_enable

函数名称	i2c2_dma_enable
函数原型	void i2c2_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}	
i2c_periph	I2C 外设
I2Cx	(x=2)
输入参数{in}	
dma	I2C DMA
I2C2_DMA_TRANSMIT	采用 DMA 方式发送数据
I2C2_DMA_RECEIVE	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 */
```

```
i2c2_dma_enable (I2C2, I2C2_DMA_RECEIVE);
```

### 函数 i2c2\_dma\_disable

函数i2c2\_dma\_disable描述见下表:

表 3-651. 函数 i2c2\_dma\_disable

函数名称	i2c2_dma_disable
函数原型	void i2c2_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}	
i2c_periph	I2C 外设
I2Cx	(x=2)
输入参数{in}	
dma	I2C DMA
I2C2_DMA_TRANSMIT	采用 DMA 方式发送数据
I2C2_DMA_RECEIVE	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 */
```

```
i2c2_dma_disable (I2C2, I2C2_DMA_RECEIVE);
```

### 函数 i2c\_smbus\_alert\_enable

函数i2c\_smbus\_alert\_enable描述见下表：

表 3-652. 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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SMBus Alert */
```

```
i2c_smbus_alert_enable (I2C2);
```

### 函数 i2c\_smbus\_alert\_disable

函数i2c\_smbus\_alert\_disable描述见下表：

表 3-653. 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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SMBus Alert */
```

```
i2c_smbus_alert_disable(I2C2);
```

### 函数 i2c\_smbus\_default\_addr\_enable

函数i2c\_smbus\_default\_addr\_enable描述见下表：

表 3-654. 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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SMBus device default address */
```

```
i2c_smbus_default_addr_enable(I2C2);
```

### 函数 i2c\_smbus\_default\_addr\_disable

函数i2c\_smbus\_default\_addr\_disable描述见下表：

表 3-655. 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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SMBus device default address */
```

```
i2c_smbus_default_addr_disable (I2C2);
```

### 函数 i2c\_smbus\_host\_addr\_enable

函数i2c\_smbus\_host\_addr\_enable描述见下表：

表 3-656. 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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SMBus Host address */
```

```
i2c_smbus_host_addr_enable (I2C2);
```

### 函数 i2c\_smbus\_host\_addr\_disable

函数i2c\_smbus\_host\_addr\_disable描述见下表：

表 3-657. 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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SMBus Host address */
```

```
i2c_smbus_host_addr_disable (I2C2);
```

### 函数 i2c\_extented\_clock\_timeout\_enable

函数i2c\_extented\_clock\_timeout\_enable描述见下表：

**表 3-658. 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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable extended clock timeout detection */
```

```
i2c_extented_clock_timeout_enable (I2C2);
```

### 函数 i2c\_extented\_clock\_timeout\_disable

函数i2c\_extented\_clock\_timeout\_disable描述见下表：

**表 3-659. 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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable extended clock timeout detection */
```

```
i2c_extented_clock_timeout_disable (I2C2);
```

### 函数 i2c\_clock\_timeout\_enable

函数i2c\_clock\_timeout\_enable描述见下表：

表 3-660. 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	使能时钟超时检测
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable clock timeout detection */
```

```
i2c_clock_timeout_enable (I2C2);
```

### 函数 i2c\_clock\_timeout\_disable

函数i2c\_clock\_timeout\_disable描述见下表：

表 3-661. 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=2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable clock timeout detection */
```

```
i2c_clock_timeout_disable (I2C2);
```

### 函数 i2c\_bus\_timeout\_b\_config

函数i2c\_bus\_timeout\_b\_config描述见下表：

表 3-662. 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=2)
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 (I2C2, 0xff);
```

### 函数 i2c\_bus\_timeout\_a\_config

函数i2c\_bus\_timeout\_a\_config描述见下表：

表 3-663. 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=2)
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 (I2C2, 0xff);
```

### 函数 i2c\_idle\_clock\_timeout\_config

函数i2c\_idle\_clock\_timeout\_config描述见下表：

表 3-664. 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=2)
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 (I2C2, BUSTOA_DETECT_SCL_LOW);
```

### 函数 i2c2\_flag\_get

函数i2c2\_flag\_get描述见下表：

表 3-665. 函数 i2c2\_flag\_get

函数名称	i2c2_flag_get
函数原型	FlagStatus i2c2_flag_get(uint32_t i2c_periph, uint32_t flag);

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=2)
输入参数{in}	
flag	I2C 标志位
I2C2_FLAG_TBE	发送期间 I2C2_TDATA 寄存器空标志
I2C2_FLAG_TI	发送中断标志
I2C2_FLAG_RBNE	接收期间 I2C2_RDATA 非空标志
I2C2_FLAG_ADDSEND	从机模式下，接收到的地址与自身地址匹配
I2C2_FLAG_NACK	NACK 标志
I2C2_FLAG_STPDET	从机模式下检测到 STOP 信号
I2C2_FLAG_TC	主机模式下传输完成标志
I2C2_FLAG_TCR	传输完成重载标志
I2C2_FLAG_BERR	总线错误标志
I2C2_FLAG_LOSTARB	仲裁丢失标志
I2C2_FLAG_OUERR	从机模式下，过载/欠载错误标志
I2C2_FLAG_PECERR	PEC 错误标志
I2C2_FLAG_TIMEOUT	超时标志
I2C2_FLAG_SMBALT	SMBus 报警标志
I2C2_FLAG_I2CBSY	忙标志
I2C2_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 = i2c2_flag_get (I2C2, I2C2_FLAG_TBE);
```

### 函数 i2c2\_flag\_clear

函数i2c2\_flag\_clear描述见下表：

表 3-666. 函数 i2c2\_flag\_clear

函数名称	i2c2_flag_clear
函数原型	void i2c2_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=2)
输入参数{in}	
<b>flag</b>	I2C 标志位
<i>I2C2_FLAG_ADDSEND</i>	从机模式下，接收到的地址与自身地址匹配
<i>I2C2_FLAG_NACK</i>	NACK 标志
<i>I2C2_FLAG_STPDET</i>	从机模式下检测到 STOP 信号
<i>I2C2_FLAG_BERR</i>	总线错误标志
<i>I2C2_FLAG_LOSTARB</i>	仲裁丢失标志
<i>I2C2_FLAG_OUERR</i>	从机模式下，过载/欠载错误标志
<i>I2C2_FLAG_PECERR</i>	PEC 错误标志
<i>I2C2_FLAG_TIMEOUT</i>	超时标志
<i>I2C2_FLAG_SMBALT</i>	SMBus 报警标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear a bus error flag*/
```

```
i2c2_flag_clear (I2C2, I2C2_FLAG_BERR);
```

### 函数 i2c2\_interrupt\_enable

函数i2c2\_interrupt\_enable描述见下表：

表 3-667. 函数 i2c2\_interrupt\_enable

函数名称	i2c2_interrupt_enable
函数原型	void i2c2_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=2)
输入参数{in}	
<b>interrupt</b>	I2C 中断
<i>I2C2_INT_ERR</i>	错误中断
<i>I2C2_INT_TC</i>	发送完成中断
<i>I2C2_INT_STPDET</i>	检测到 STOP 中断

<i>I2C2_INT_NACK</i>	接收到 NACK 中断
<i>I2C2_INT_ADDM</i>	地址匹配中断
<i>I2C2_INT_RBNE</i>	接收中断
<i>I2C2_INT_TI</i>	发送中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C2 transmit interrupt */
```

```
i2c2_interrupt_enable (I2C2, I2C2_INT_TI);
```

### 函数 i2c2\_interrupt\_disable

函数i2c2\_interrupt\_disable描述见下表：

表 3-668. 函数 i2c2\_interrupt\_disable

函数名称	i2c2_interrupt_disable
函数原型	void i2c2_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=2)
输入参数{in}	
<b>interrupt</b>	I2C 中断
<i>I2C2_INT_ERR</i>	错误中断
<i>I2C2_INT_TC</i>	发送完成中断
<i>I2C2_INT_STPDET</i>	检测到 STOP 中断
<i>I2C2_INT_NACK</i>	接收到 NACK 中断
<i>I2C2_INT_ADDM</i>	地址匹配中断
<i>I2C2_INT_RBNE</i>	接收中断
<i>I2C2_INT_TI</i>	发送中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C2 transmit interrupt */
```

```
i2c2_interrupt_disable (I2C2, I2C2_INT_TI);
```

## 函数 i2c2\_interrupt\_flag\_get

函数i2c2\_interrupt\_flag\_get描述见下表：

表 3-669. 函数 i2c2\_interrupt\_flag\_get

函数名称	i2c2_interrupt_flag_get
函数原型	FlagStatus i2c2_interrupt_flag_get(uint32_t i2c_periph, 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=2)
输入参数{in}	
int_flag	I2C 中断标志位，参考 <a href="#">表 3-576. 枚举类型 i2c2_interrupt_flag_enum</a> 。
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

```
/* get I2C interrupt flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c2_interrupt_flag_get (I2C2, I2C2_INT_FLAG_TI);
```

## 函数 i2c2\_interrupt\_flag\_clear

函数i2c2\_interrupt\_flag\_clear描述见下表：

表 3-670. 函数 i2c2\_interrupt\_flag\_clear

函数名称	i2c2_interrupt_flag_clear
函数原型	void i2c2_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=2)
输入参数{in}	
int_flag	I2C 中断标志位，参考 <a href="#">表 3-576. 枚举类型 i2c2_interrupt_flag_enum</a> 。
I2C2_INT_FLAG_ADD	从机模式下，接收到的地址与自身地址匹配中断标志

<i>SEND</i>	
<i>I2C2_INT_FLAG_NACK</i>	NACK 中断标志
<i>I2C2_INT_FLAG_STOPDET</i>	从机模式下检测到 STOP 信号中断标志
<i>I2C2_INT_FLAG_BERR</i>	总线错误中断标志
<i>I2C2_INT_FLAG_LOSTARB</i>	仲裁丢失中断标志
<i>I2C2_INT_FLAG_OVERR</i>	从机模式下，过载/欠载错误中断标志
<i>I2C2_INT_FLAG_PECERR</i>	PEC 错误中断标志
<i>I2C2_INT_FLAG_TIMEOUT</i>	超时中断标志
<i>I2C2_INT_FLAG_SMBALERT</i>	SMBus 报警中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear a bus error flag */
```

```
i2c2_interrupt_flag_clear(I2C2, I2C2_INT_FLAG_BERR);
```

## 3.19. 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 [3.19.1](#) 描述了 NVIC 和 SysTick 的寄存器列表，章节 [3.19.2](#) 对 MISC 库函数进行说明。

### 3.19.1. 外设寄存器说明

表 3-671. 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

寄存器名称	寄存器描述
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
SHP <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 3-672. SysTick 寄存器**

寄存器名称	寄存器描述
CTRL <sup>(1)</sup>	系统控制和状态寄存器
LOAD <sup>(1)</sup>	系统重载值寄存器
VAL <sup>(1)</sup>	系统当前值寄存器
CALIB <sup>(1)</sup>	系统校准寄存器

1. 参考 core\_cm33.h 文件中定义的结构体类型 SysTick\_Type

### 3.19.2. 外设库函数说明

#### 枚举类型 IRQn\_Type

**表 3-673. 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 中断
TAMPER_IRQn	侵入检测中断
RTC_IRQn	RTC 全局中断
FMC_IRQn	FMC 全局中断
RCU_CTC_IRQn	RCU 和 CTC 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

成员名称	功能描述
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 全局中断
USBD_HP_CAN0_TX_IRQn / CAN0_TX_IRQn	USBD 高优先级 CAN0 发送中断 / CAN0 发送中断
USBD_LP_CAN0_RX0_IRQn / CAN0_RX0_IRQn	USBD 低优先级 CAN0 接收 0 中断 / CAN0 接收 0 中断
CAN0_RX1_IRQn	CAN0 接收 1 中断
CAN0_EWMC_IRQn	CAN0 EWMC 中断
EXTI5_9_IRQn	EXTI 线[9:5]中断
TIMER0_BRK_TIMER8_TIMER14_IRQn	TIMER0 中止中断和 TIMER8 和 TIMER14 全局中断
TIMER0_UP_TIMER9_TIMER15_IRQn	TIMER0 更新中断和 TIMER9 和 TIMER15 全局中断
TIMER0_TRG_CMT_TIMER10_TIMER16_IRQn	TIMER0 触发与通道换相中断和 TIMER10 和 TIMER16 全局中断
TIMER0_Channel_IRQn	TIMER0 通道捕获比较中断
TIMER1_IRQn	TIMER1 全局中断
TIMER2_IRQn	TIMER2 全局中断
TIMER3_IRQn	TIMER3 全局中断
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_I2S1ADD_IRQn	SPI1 和 I2S1ADD 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
RTC_Alarm_IRQn	连接 EXTI 线的 RTC 闹钟中断
USBD_WKUP_IRQn / USBFS_WKUP_IRQn	连接 EXTI 线的 USBD / USBFS 唤醒中断
TIMER7_BRK_TIMER11_IRQn	TIMER7 中止中断和 TIMER11 全局中断
TIMER7_UP_TIMER12_IRQn	TIMER7 更新中断和 TIMER12 全局中断
TIMER7_TRG_CMT_TIMER13_IRQn	TIMER7 触发与通道换相中断和 TIMER13 全局中断
TIMER7_Channel_IRQn	TIMER7 通道捕获比较中断
ADC2_IRQn	ADC2 全局中断
EXMC_IRQn	EXMC 全局中断
SDIO_IRQn	SDIO 全局中断
TIMER4_IRQn	TIMER4 全局中断
SPI2_I2S2ADD_IRQn	SPI2 和 I2S2ADD 全局中断

成员名称	功能描述
UART3_IRQn	UART3 全局中断
UART4_IRQn	UART4 全局中断
TIMER5_DAC_IRQn	TIMER5 或 DAC 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_Channel4_IRQn / DMA1_Channel3_IRQn	DMA1 通道 3 全局中断和 DMA1 通道 4 全局中断 / DMA1 通道 3 全局中断
DMA1_Channel4_IRQn	DMA1 通道 4 全局中断
ENET_IRQn	ENET 中断
ENET_WKUP_IRQn	ENET 唤醒中断
CAN1_TX_IRQn	CAN1 发送中断
CAN1_RX0_IRQn	CAN1 接收 0 中断
CAN1_RX1_IRQn	CAN1 接收 1 中断
CAN1_EWMC_IRQn	CAN1 EWMC 中断
USBHS_IRQn	USBHS 全局中断
SHRTIMER_IRQ2_IRQn	SHRTIMER 全局中断 2
SHRTIMER_IRQ3_IRQn	SHRTIMER 全局中断 3
SHRTIMER_IRQ4_IRQn	SHRTIMER 全局中断 4
SHRTIMER_IRQ5_IRQn	SHRTIMER 全局中断 5
SHRTIMER_IRQ6_IRQn	SHRTIMER 全局中断 6
USBHS_EP1_OUT_IRQn	USBHS 端点 1 输出中断
USBHS_EP1_IN_IRQn	USBHS 端点 1 输入中断
SHRTIMER_IRQ0_IRQn	SHRTIMER 全局中断 0
SHRTIMER_IRQ1_IRQn	SHRTIMER 全局中断 1
CAN2_TX_IRQn	CAN2 发送中断
CAN2_RX0_IRQn	CAN2 接收 0 中断
CAN2_RX1_IRQn	CAN2 接收 1 中断
CAN2_EWMC_IRQn	CAN2 EWMC 中断
I2C2_EV_IRQn	I2C2 事件中断
I2C2_ER_IRQn	I2C2 错误中断
USART5_IRQn	USART5 全局中断
I2C2_WKUP_IRQn	I2C2 唤醒中断
USART5_WKUP_IRQn	USART5 唤醒中断
TMU_IRQn	TMU 全局中断

MISC库函数列表如下表所示:

**表 3-674. MISC 库函数**

库函数名称	库函数描述
nvic_priority_group_set	设置优先级组

库函数名称	库函数描述
<code>nvic_irq_enable</code>	使能NVIC的中断
<code>nvic_irq_disable</code>	禁能NVIC的中断
<code>nvic_system_reset</code>	复位MCU
<code>nvic_vector_table_set</code>	设置向量表地址
<code>system_lowpower_set</code>	设置系统低功耗模式状态
<code>system_lowpower_reset</code>	复位系统低功耗模式状态
<code>systick_clksource_set</code>	设置系统定时器时钟源

### 函数 `nvic_priority_group_set`

函数`nvic_priority_group_set`描述见下表:

**表 3-675. 函数 `nvic_priority_group_set`**

函数名称	<code>nvic_priority_group_set</code>
函数原形	<code>void nvic_priority_group_set(uint32_t nvic_prigroup);</code>
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}	
<b><code>nvic_prigroup</code></b>	优先级组
<code>NVIC_PRIGROUP_PRE0_SUB4</code>	0位用于抢占优先级, 4位用于响应优先级
<code>NVIC_PRIGROUP_PRE1_SUB3</code>	1位用于抢占优先级, 3位用于响应优先级
<code>NVIC_PRIGROUP_PRE2_SUB2</code>	2位用于抢占优先级, 2位用于响应优先级
<code>NVIC_PRIGROUP_PRE3_SUB1</code>	3位用于抢占优先级, 1位用于响应优先级
<code>NVIC_PRIGROUP_PRE4_SUB0</code>	4位用于抢占优先级, 0位用于响应优先级
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 3-676. 函数 nvic\_irq\_enable

函数名称	nvic_irq_enable
函数原形	void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority);
功能描述	使能中断，配置中断的优先级
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 <a href="#">表3-673. 枚举类型IRQn_Type</a>
输入参数{in}	
nvic_irq_pre_priority	抢占优先级（0~4）
输入参数{in}	
nvic_irq_sub_priority	响应优先级（0~4）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable window watchdog timer interrupt, pre-emption priority is 1, subpriority 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 3-677. 函数 nvic\_irq\_disable

函数名称	nvic_irq_disable
函数原形	void nvic_irq_disable (uint8_t nvic_irq);
功能描述	禁能中断
先决条件	-
被调用函数	-
输入参数{in}	
nvic_irq	NVIC中断，参考 <a href="#">表3-673. 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);
```

### 函数 nvic\_system\_reset

函数nvic\_system\_reset描述见下表：

表 3-678. 函数 nvic\_system\_reset

函数名称	nvic_system_reset
函数原形	void nvic_system_reset(void);
功能描述	initiates a system reset request to reset the MCU
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initiates a system reset request to reset the MCU */
nvic_system_reset();
```

### 函数 nvic\_vector\_table\_set

函数nvic\_vector\_table\_set描述见下表：

表 3-679. 函数 nvic\_vector\_table\_set

函数名称	nvic_vector_table_set
函数原形	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 3-680. 函数 system\_lowpower\_set**

函数名称	system_lowpower_set
函数原形	void system_lowpower_set(uint8_t lowpower_mode);
功能描述	系统低功耗模式状态的管理
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_DEEPSLEEP	该位为1时，系统处于deep sleep模式
SCB_LPM_WAKEUP_BY_ALL_INT	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 3-681. 函数 system\_lowpower\_reset**

函数名称	system_lowpower_reset
函数原形	void system_lowpower_reset(uint8_t lowpower_mode);
功能描述	系统低功耗模式状态的管理
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0时，退出ISR时退出低功耗模式
SCB_LPM_DEEPSLEEP	该位为0时，系统进入sleep模式
SCB_LPM_WAKE_BY_ALL_INT	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 3-682. 函数 systick\_clksource\_set

函数名称	systick_clksource_set
函数原形	void systick_clksource_set(uint32_t systick_clksource);
功能描述	设置SysTick时钟源
先决条件	-
被调用函数	-
输入参数{in}	
systick_clksource	SysTick时钟源
SYSTICK_CLKSOURCE_HCLK	SysTick时钟源为AHB时钟
SYSTICK_CLKSOURCE_HCLK_DIV8	SysTick时钟源为AHB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);
```

## 3.20. PMU

电源管理单元提供了五种省电模式，包括睡眠模式，深度睡眠模式，深度睡眠模式 1，深度睡眠模式 2 和待机模式。章节 [3.20.1](#) 描述了 PMU 的寄存器列表，章节 [3.20.2](#) 对 PMU 库函数进行说明。

### 3.20.1. 外设寄存器说明

PMU 寄存器列表如下表所示：

**表 3-683. PMU 寄存器**

寄存器名称	寄存器描述
PMU_CTL0	控制寄存器0
PMU_CS0	电源控制和状态寄存器0
PMU_CTL1	控制寄存器1
PMU_CS1	电源控制和状态寄存器1

### 3.20.2. 外设库函数说明

PMU 库函数列表如下表所示：

**表 3-684. PMU 库函数**

库函数名称	库函数描述
pmu_deinit	复位外设PMU
pmu_lvd_select	选择低压检测阈值
pmu_lvd_disable	关闭低压检测器
pmu_highdriver_switch_select	高驱动模式切换器选择
pmu_highdriver_mode_enable	使能高驱动模式
pmu_highdriver_mode_disable	失能高驱动模式
pmu_lowdriver_mode_enable	深度睡眠模式下使能低驱动模式
pmu_lowdriver_mode_disable	深度睡眠模式下失能低驱动模式
pmu_lowpower_driver_config	使用低功耗LDO时驱动模式选择
pmu_normalpower_driver_config	使用正常LDO时驱动模式选择
pmu_to_sleepmode	进入睡眠模式
pmu_to_deepsleepmode	进入深度睡眠模式
pmu_to_deepsleepmode_1	进入深度睡眠模式1
pmu_to_deepsleepmode_2	进入深度睡眠模式2
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

库函数名称	库函数描述
pmu_flag_clear	清除标志位

### 函数 pmu\_deinit

函数pmu\_deinit描述见下表：

**表 3-685. 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 3-686. 函数 pmu\_lvd\_select**

函数名称	pmu_lvd_select
函数原型	void pmu_lvd_select(uint32_t lvdt_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}	
lvdt_n	电压阈值
PMU_LVDT_0	电压阈值为2.1V
PMU_LVDT_1	电压阈值为2.3V
PMU_LVDT_2	电压阈值为2.4V
PMU_LVDT_3	电压阈值为2.6V
PMU_LVDT_4	电压阈值为2.7V
PMU_LVDT_5	电压阈值为2.9V
PMU_LVDT_6	电压阈值为3.0V
PMU_LVDT_7	电压阈值为3.1V

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select low voltage detector threshold as 3.1V */
```

```
pmu_lvd_select (PMU_LVDT_7);
```

### 函数 pmu\_lvd\_disable

函数pmu\_lvd\_disable描述见下表：

表 3-687. 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\_highdriver\_switch\_select

函数pmu\_highdriver\_switch\_select描述见下表：

表 3-688. 函数 pmu\_highdriver\_switch\_select

函数名称	pmu_highdriver_switch_select
函数原型	void pmu_highdriver_switch_select (uint32_t highdr_switch);
功能描述	高驱动模式切换器选择
先决条件	-
被调用函数	-
输入参数{in}	
highdr_switch	使能或失能高驱动模式切换器
PMU_HIGHDR_SW ITCH_NONE	失能高驱动模式切换器

PMU_HIGHDR_SW ITCH_EN	使能高驱动模式切换器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable high-driver mode switch */
```

```
pmu_highdriver_switch_select (PMU_HIGHDR_SWITCH_EN);
```

### 函数 pmu\_highdriver\_mode\_enable

函数pmu\_highdriver\_mode\_enable描述见下表：

表 3-689. 函数 pmu\_highdriver\_mode\_enable

函数名称	pmu_highdriver_mode_enable
函数原型	void pmu_highdriver_mode_enable(void);
功能描述	使能高驱动模式
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 high-driver mode */
```

```
pmu_highdriver_mode_enable ();
```

### 函数 pmu\_highdriver\_mode\_disable

函数pmu\_highdriver\_mode\_disable描述见下表：

表 3-690. 函数 pmu\_highdriver\_mode\_disable

函数名称	pmu_highdriver_mode_disable
函数原型	void pmu_highdriver_mode_disable(void);
功能描述	失能高驱动模式
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 high-driver mode */
pmu_highdriver_mode_disable ();
```

### 函数 pmu\_lowdriver\_mode\_enable

函数pmu\_lowdriver\_mode\_enable描述见下表：

表 3-691. 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	使能低驱动模式
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 */
pmu_lowdriver_mode_enable ();
```

### 函数 pmu\_lowdriver\_mode\_disable

函数pmu\_lowdriver\_mode\_disable描述见下表：

表 3-692. 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	失能低驱动模式
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 */
```

```
pmu_lowdriver_mode_disable ();
```

### 函数 pmu\_lowpower\_driver\_config

函数pmu\_lowpower\_driver\_config描述见下表：

**表 3-693. 函数 pmu\_lowpower\_driver\_config**

函数名称	pmu_lowpower_driver_config
函数原型	void pmu_lowpower_driver_config (void);
功能描述	低功耗LDO下驱动模式配置
先决条件	-
被调用函数	-
输入参数{in}	
mode	驱动模式
PMU_NORMALDR_LOW_PWR	正常驱动模式
PMU_LOWDR_LO_WPW	低驱动模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* low-driver mode when use low power LDO */
```

```
pmu_lowpower_driver_config (PMU_NORMALDR_LOW_PWR);
```

### 函数 pmu\_normalpower\_driver\_config

函数pmu\_normalpower\_driver\_config描述见下表：

**表 3-694. 函数 pmu\_normalpower\_driver\_config**

函数名称	pmu_normalpower_driver_config
函数原型	void pmu_normalpower_driver_config (void);
功能描述	正常功耗LDO下驱动模式配置
先决条件	-
被调用函数	-
输入参数{in}	
mode	驱动模式

<i>PMU_NORMALDR_</i> <i>LOWPWR</i>	正常驱动模式
<i>PMU_LOWDR_NO</i> <i>RMALPWR</i>	低驱动模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* normal driver when use low power LDO */
```

```
pmu_normalpower_driver_config (PMU_LOWDR_NORMALPWR);
```

### 函数 pmu\_to\_sleepmode

函数pmu\_to\_sleepmode描述见下表:

表 3-695. 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 in sleep mode */
```

```
pmu_to_sleepmode (WFI_CMD);
```

### 函数 pmu\_to\_deepsleepmode

函数pmu\_to\_deepsleepmode描述见下表:

表 3-696. 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}	
lowdrive	低驱动模式
PMU_LOWDRIVER_ENABLE	在深度睡眠模式下使能低驱动
PMU_LOWDRIVER_DISABLE	在深度睡眠模式下禁能低驱动
输入参数{in}	
deepsleepmodecmd	进入深度睡眠模式命令
WFI_CMD	WFI命令
WFE_CMD	WFE命令
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PMU work in deepsleep mode */
```

```
pmu_to_deepsleepmode (PMU_LDO_NORMAL, PMU_LOWDRIVER_ENABLE, WFI_CMD);
```

### 函数 pmu\_to\_deepsleepmode\_1

函数pmu\_to\_deepsleepmode\_1描述见下表：

表 3-697. 函数 pmu\_to\_deepsleepmode\_1

函数名称	pmu_to_deepsleepmode_1
函数原型	void pmu_to_deepsleepmode_1(uint32_t ldo, uint32_t lowdrive, uint8_t deepsleepmode1cmd);
功能描述	进入深度睡眠模式1
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1时，LDO仍正常工作

<i>L</i>	
<i>PMU_LDO_LOWPOWER</i>	当系统进入深度睡眠模式1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>	在深度睡眠模式1下使能低驱动
<i>PMU_LOWDRIVER_DISABLE</i>	在深度睡眠模式1下禁能低驱动
输入参数{in}	
<b>deepsleepmodecmd</b>	进入深度睡眠模式1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 in deepsleep mode 1 */
```

```
pmu_to_deepsleepmode_1(PMU_LDO_NORMAL, PMU_LOWDRIVER_ENABLE, WFI_CMD);
```

### 函数 pmu\_to\_deepsleepmode\_2

函数pmu\_to\_deepsleepmode\_2描述见下表：

表 3-698. 函数 pmu\_to\_deepsleepmode\_2

函数名称	pmu_to_deepsleepmode_2
函数原型	void pmu_to_deepsleepmode_2(uint32_t ldo, uint32_t lowdrive, uint8_t deepsleepmode2cmd);
功能描述	进入深度睡眠模式2
先决条件	-
被调用函数	-
输入参数{in}	
<b>ldo</b>	LDO工作模式
<i>PMU_LDO_NORMAL</i>	当系统进入深度睡眠模式2时，LDO仍正常工作
<i>PMU_LDO_LOWPOWER</i>	当系统进入深度睡眠模式2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>	在深度睡眠模式2下使能低驱动
<i>PMU_LOWDRIVER_DISABLE</i>	在深度睡眠模式2下禁能低驱动
输入参数{in}	
<b>deepsleepmodecmd</b>	进入深度睡眠模式2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 in deepsleep mode 2 */
```

```
pmu_to_deepsleepmode_2(PMU_LDO_NORMAL, PMU_LOWDRIVER_ENABLE, WFI_CMD);
```

### 函数 pmu\_to\_standbymode

函数pmu\_to\_standbymode描述见下表：

**表 3-699. 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 in standby mode */
```

```
pmu_to_standby ();
```

### 函数 pmu\_backup\_write\_enable

函数pmu\_backup\_write\_enable描述见下表：

表 3-700. 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 3-701. 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\_wakeup\_pin\_enable

函数pmu\_wakeup\_pin\_enable描述见下表:

表 3-702. 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	唤醒系统的管脚
PMU_WAKEUP_PIN0	唤醒脚0 (PA0)
PMU_WAKEUP_PIN1	唤醒脚1 (PC13)
PMU_WAKEUP_PIN2	唤醒脚2 (PE6)
PMU_WAKEUP_PIN3	唤醒脚3 (PA2)
PMU_WAKEUP_PIN4	唤醒脚4 (PC5)
PMU_WAKEUP_PIN5	唤醒脚5 (PB5)
PMU_WAKEUP_PIN6	唤醒脚6 (PB15)
PMU_WAKEUP_PIN7	唤醒脚7 (PF8)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable wakeup pin 0 */
```

```
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 3-703. 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	唤醒系统的管脚

PMU_WAKEUP_PIN0	唤醒脚0 (PA0)
PMU_WAKEUP_PIN1	唤醒脚1 (PC13)
PMU_WAKEUP_PIN2	唤醒脚2 (PE6)
PMU_WAKEUP_PIN3	唤醒脚3 (PA2)
PMU_WAKEUP_PIN4	唤醒脚4 (PC5)
PMU_WAKEUP_PIN5	唤醒脚5 (PB5)
PMU_WAKEUP_PIN6	唤醒脚6 (PB15)
PMU_WAKEUP_PIN7	唤醒脚7 (PF8)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable wakeup pin 0 */
```

```
pmu_wakeup_pin_disable (PMU_WAKEUP_PIN0);
```

### 函数 pmu\_flag\_get

函数pmu\_flag\_get描述见下表:

表 3-704. 函数 pmu\_flag\_get

函数名称	pmu_flag_get
函数原型	FlagStatus pmu_flag_get(uint32_t flag_reset);
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_RESET_WAKEUP	唤醒标志
PMU_FLAG_RESET_STANDBY	待机标志
PMU_FLAG_LVD	低电压检测标志
PMU_FLAG_HDRF	高驱动就绪标志

<i>PMU_FLAG_HDSR</i> <i>F</i>	高驱动切换器就绪标志
<i>PMU_FLAG_LDRF</i>	低驱动就绪标志
<i>PMU_FLAG_DEEP</i> <i>SLEEP_1</i>	深度睡眠模式1标志
<i>PMU_FLAG_DEEP</i> <i>SLEEP_2</i>	深度睡眠模式2标志
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如:

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

```
FlagStatus status;
```

```
status = pmu_flag_get (PMU_FLAG_RESET_STANDBY);
```

### 函数 pmu\_flag\_clear

函数pmu\_flag\_clear描述见下表:

表 3-705. 函数 pmu\_flag\_clear

函数名称	pmu_flag_clear
函数原型	void pmu_flag_clear(uint32_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}	
<b>flag</b>	标志位
<i>PMU_FLAG_RESE</i> <i>T_WAKEUP</i>	清除唤醒标志
<i>PMU_FLAG_RESE</i> <i>T_STANDBY</i>	清除待机标志
<i>PMU_FLAG_RESE</i> <i>T_DEEPSLEEP_1</i>	清除深度睡眠模式1标志
<i>PMU_FLAG_RESE</i> <i>T_DEEPSLEEP_2</i>	清除深度睡眠模式2标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear flag bit */
```

```
pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

## 3.21. 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 [3.21.1](#) 描述了 RCU 的寄存器列表，章节 [3.21.2](#) 对 RCU 库函数进行说明。

### 3.21.1. 外设寄存器说明

RCU寄存器列表如下表所示：

**表 3-706. RCU 寄存器**

寄存器名称	寄存器描述
RCU_CTL	控制寄存器
RCU_CFG0	时钟配置寄存器0
RCU_INT	时钟中断寄存器
RCU_APB2RST	APB2复位寄存器
RCU_APB1RST	APB1复位寄存器
RCU_AHBMEN	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（仅适用于CL、E518系列）
RCU_CFG1	时钟配置寄存器1
RCU_DSV	深度睡眠模式电压寄存器
RCU_ADDCTL	附加时钟控制寄存器
RCU_ADDCFG	附加时钟配置寄存器（仅适用于CL、E518系列）
RCU_ADDINT	附加时钟中断寄存器
RCU_PLLSSCTL	PLL 时钟扩频控制寄存器（仅适用于CL、E518系列）
RCU_CFG2	时钟配置寄存器2
RCU_ADDAPB1RST	APB1附加复位寄存器
RCU_ADDAPB1EN	APB1附加使能寄存器

### 3.21.2. 外设库函数说明

RCU库函数列表如下表所示：

表 3-707. RCU 库函数

库函数名称	库函数描述
rcu_deinit	复位RCU
rcu_periph_clock_enable	使能外设时钟
rcu_periph_clock_disable	禁能外设时钟
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_periph_reset_enable	使能外设复位
rcu_periph_reset_disable	禁能外设复位
rcu_bkp_reset_enable	使能BKP复位
rcu_bkp_reset_disable	禁能BKP复位
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_ckout0_config	配置CKOUT0时钟源选择
rcu_pll_config	配置主PLL时钟
rcu_pllpreselect_config	配置PLL时钟源选择
rcu_predv0_config	配置PREDV0分频因子
rcu_predv1_config	配置PREDV1分频因子
rcu_pll1_config	配置PLL1时钟
rcu_pll2_config	配置PLL2时钟
rcu_pllusbprediv_config	配置PLLUSBPREV分频系数（仅适用于CL、E518系列）
rcu_pllusb_config	配置PLLUSB倍频因子（仅适用于CL、E518系列）
rcu_adc_clock_config	配置ADC的时钟分频系数
rcu_usb_clock_config	配置USB的时钟分频系数
rcu_rtc_clock_config	配置RTC的时钟源选择
rcu_shrtimer_clock_config	配置SHRTIMER时钟源选择
rcu_usart5_clock_config	配置USART5时钟源选择
rcu_i2s1_clock_config	配置I2S1的时钟源选择（仅适用于CL、E518系列）
rcu_i2s2_clock_config	配置I2S2的时钟源选择（仅适用于CL、E518系列）
rcu_ck48m_clock_config	配置CK48M时钟选择
rcu_usbssel_config	配置USBSEL时钟源选择
rcu_usbdv_config	配置USBDV分频系数（仅适用于CL、E518系列）
rcu_flag_get	获取时钟稳定和复位标志
rcu_all_reset_flag_clear	清除所有复位标志位
rcu_interrupt_flag_get	获取时钟稳定中断和时钟阻塞中断标志
rcu_interrupt_flag_clear	清除中断标志
rcu_interrupt_enable	使能时钟稳定中断
rcu_interrupt_disable	禁能时钟稳定中断
rcu_lxtal_drive_capability_config	配置LXTAL驱动能力

库函数名称	库函数描述
rcu_osc_stab_wait	等待振荡器稳定标志位置位或振荡器起振超时
rcu_osc_on	打开振荡器
rcu_osc_off	关闭振荡器
rcu_osc_bypass_mode_enable	使能振荡器时钟旁路模式
rcu_osc_bypass_mode_disable	禁能振荡器时钟旁路模式
rcu_hxtal_clock_monitor_enable	使能HXTAL时钟监视器
rcu_hxtal_clock_monitor_disable	禁能HXTAL时钟监视器
rcu_irc8m_adjust_value_set	设置内部8MHz RC振荡器时钟调整值
rcu_deepsleep_voltage_set	设置深度睡眠模式电压值
rcu_clock_freq_get	获取系统时钟、总线频率

### 枚举类型 rcu\_periph\_enum

表 3-708. 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_CRC	CRC时钟
RCU_EXMC	EXMC时钟
RCU_SDIO	SDIO时钟（仅适用于HD系列）
RCU_USBHS	USBHS时钟（仅适用于CL、E518系列）
RCU_ULPI	ULPI时钟（仅适用于CL、E518系列）
RCU_ENET	ENET时钟（仅适用于CL、E518系列）
RCU_ENETTX	ENETTX时钟（仅适用于CL、E518系列）
RCU_ENETRX	ENETRX时钟（仅适用于CL、E518系列）
RCU_TMU	TMU时钟
RCU_SQPI	SQPI时钟
RCU_TIMER1	TIMER1时钟
RCU_TIMER2	TIMER2时钟
RCU_TIMER3	TIMER3时钟
RCU_TIMER4	TIMER4时钟
RCU_TIMER5	TIMER5时钟
RCU_TIMER6	TIMER6时钟
RCU_TIMER11	TIMER11时钟
RCU_TIMER12	TIMER12时钟
RCU_TIMER13	TIMER13时钟
RCU_TIMER14	TIMER14时钟
RCU_TIMER15	TIMER15时钟
RCU_TIMER16	TIMER16时钟
RCU_WWDGT	WWDGT时钟
RCU_SPI1	SPI1时钟
RCU_SPI2	SPI2时钟

成员名称	功能描述
RCU_USART1	USART1时钟
RCU_USART2	USART2时钟
RCU_UART3	UART3时钟
RCU_UART4	UART4时钟
RCU_I2C0	I2C0时钟
RCU_I2C1	I2C1时钟
RCU_USBD	USBD 时钟（仅适用于 HD 系列）
RCU_I2C2	I2C2时钟
RCU_CAN0	CAN0时钟
RCU_CAN1	CAN1时钟
RCU_BKPI	BKPI时钟
RCU_PMU	PMU时钟
RCU_DAC0	DAC0时钟
RCU_DAC1	DAC1时钟
RCU_RTC	RTC时钟
RCU_CTC	CTC时钟
RCU_CAN2	CAN2时钟
RCU_AF	复用功能时钟
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_GPIOG	GPIOG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_ADC2	ADC2时钟（适用于除CL其他系列）
RCU_TIMER8	TIMER8时钟
RCU_TIMER9	TIMER9时钟
RCU_TIMER10	TIMER10时钟
RCU_SHRTIMER	SHRTIMER时钟
RCU_USART5	USART5时钟
RCU_CMP	CMP时钟（仅适用于CL、E518系列）

## 枚举类型 `rcu_periph_sleep_enum`

表 3-709. 枚举类型 `rcu_periph_sleep_enum`

成员名称	功能描述
RCU_SRAM_SLP	SRAM接口时钟
RCU_FMC_SLP	FMC时钟

## 枚举类型 `rcu_periph_reset_enum`

表 3-710. 枚举类型 `rcu_periph_reset_enum`

成员名称	功能描述
RCU_USBHSRST	复位USBHS时钟（仅适用于CL、E518系列）
RCU_ENETRST	复位ENET时钟（仅适用于CL、E518系列）
RCU_TMURST	复位TMU时钟
RCU_SQPIRST	复位SQPI时钟
RCU_TIMER1RST	复位TIMER1时钟
RCU_TIMER2RST	复位TIMER2时钟
RCU_TIMER3RST	复位TIMER3时钟
RCU_TIMER4RST	复位TIMER4时钟
RCU_TIMER5RST	复位TIMER5时钟
RCU_TIMER6RST	复位TIMER6时钟
RCU_TIMER11RST	复位TIMER11时钟
RCU_TIMER12RST	复位TIMER12时钟
RCU_TIMER13RST	复位TIMER13时钟
RCU_TIMER14RST	复位TIMER14时钟
RCU_TIMER15RST	复位TIMER15时钟
RCU_TIMER16RST	复位TIMER16时钟
RCU_WWDGTRST	复位WWDGT时钟
RCU_SPI1RST	复位SPI1时钟
RCU_SPI2RST	复位SPI2时钟
RCU_USART1RST	复位USART1时钟
RCU_USART2RST	复位USART2时钟
RCU_UART3RST	复位UART3时钟
RCU_UART4RST	复位UART4时钟
RCU_I2C0RST	复位I2C0时钟
RCU_I2C1RST	复位I2C1时钟
RCU_USBD RST	复位 USB D 时钟（仅适用于 HD 系列）
RCU_I2C2RST	复位I2C2时钟
RCU_CAN0RST	复位CAN0时钟
RCU_CAN1RST	复位CAN1时钟
RCU_BKPIRST	复位BKPI时钟
RCU_PMURST	复位PMU时钟
RCU_DAC0RST	复位DAC0时钟

成员名称	功能描述
RCU_DAC1RST	复位DAC1时钟
RCU_CTCRST	复位CTC时钟
RCU_CAN2RST	复位CAN2时钟
RCU_AFRST	复位功能时钟
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_GPIOGRST	复位GPIOG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
RCU_USART0RST	复位USART0时钟
RCU_ADC2RST	复位ADC2时钟（适用于除CL其他系列）
RCU_TIMER8RST	复位TIMER8时钟
RCU_TIMER9RST	复位TIMER9时钟
RCU_TIMER10RST	复位TIMER10时钟
RCU_USART5RST	复位USART5时钟
RCU_SHRTIMERRST	复位HPTIEMR时钟
RCU_CMPRST	复位CMP时钟（仅适用于CL、E518系列）

### 枚举类型 `rcu_flag_enum`

表 3-711. 枚举类型 `rcu_flag_enum`

成员名称	功能描述
RCU_FLAG_IRC8MSTB	IRC8M振荡器稳定标志
RCU_FLAG_HXTALSTB	外部高速晶振稳定标志
RCU_FLAG_PLLSTB	PLL稳定标志
RCU_FLAG_PLL1STB	PLL1 稳定标志（仅适用于CL、E518系列）
RCU_FLAG_PLL2STB	PLL2 稳定标志（仅适用于CL、E518系列）
RCU_FLAG_PLLUSBSTB	PLLUSB 稳定标志（仅适用于CL、E518系列）
RCU_FLAG_LXTALSTB	LXTAL稳定标志
RCU_FLAG_IRC40KSTB	IRC40K稳定标志
RCU_FLAG_IRC48MSTB	IRC48M稳定标志
RCU_FLAG_BORRST	BOR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

成员名称	功能描述
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 3-712. 枚举类型 `rcu_int_flag_enum`

成员名称	功能描述
RCU_INT_FLAG_IRC40KSTB	IRC40K时钟稳定中断标志
RCU_INT_FLAG_LXTALSTB	外部低速晶振时钟稳定中断标志
RCU_INT_FLAG_IRC8MSTB	IRC8M时钟稳定中断标志
RCU_INT_FLAG_HXTALSTB	外部高速晶振时钟稳定中断标志
RCU_INT_FLAG_PLLSTB	PLL时钟稳定中断标志
RCU_INT_FLAG_PLL1STB	PLL1时钟稳定中断标志（仅适用于CL、E518系列）
RCU_INT_FLAG_PLL2STB	PLL2时钟稳定中断标志（仅适用于CL、E518系列）
RCU_INT_FLAG_PLLUSBSTB	PLLUSB时钟稳定中断标志（仅适用于CL、E518系列）
RCU_INT_FLAG_CKM	外部高速晶振时钟阻塞中断标志
RCU_INT_FLAG_IRC48MSTB	IRC48M时钟稳定中断标志

### 枚举类型 `rcu_int_flag_clear_enum`

表 3-713. 枚举类型 `rcu_int_flag_clear_enum`

成员名称	功能描述
RCU_INT_FLAG_IRC40KSTB_CLR	IRC40K时钟稳定中断清除标志
RCU_INT_FLAG_LXTALSTB_CLR	外部低速晶振时钟稳定中断清除标志
RCU_INT_FLAG_IRC8MSTB_CLR	IRC8M时钟稳定中断清除标志
RCU_INT_FLAG_HXTALSTB_CLR	外部高速晶振时钟稳定中断清除标志
RCU_INT_FLAG_PLLSTB_CLR	PLL时钟稳定中断清除标志
RCU_INT_FLAG_PLL1STB_CLR	PLL1时钟稳定中断清除标志（仅适用于CL、E518系列）
RCU_INT_FLAG_PLL2STB_CLR	PLL2时钟稳定中断清除标志（仅适用于CL、E518系列）
RCU_INT_FLAG_PLLUSBSTB_CLR	PLLUSB时钟稳定中断清除标志（仅适用于CL、E518系列）
RCU_INT_FLAG_CKM_CLR	外部高速晶振时钟阻塞中断清除标志
RCU_INT_FLAG_IRC48MSTB_CLR	IRC48M时钟稳定中断清除标志

## 枚举类型 `rcu_int_enum`

表 3-714. 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>	外部低速晶振时钟稳定中断
<code>RCU_INT_IRC8MSTB</code>	IRC8M时钟稳定中断
<code>RCU_INT_HXTALSTB</code>	外部高速晶振时钟稳定中断
<code>RCU_INT_PLLSTB</code>	PLL时钟稳定中断
<code>RCU_INT_PLL1STB</code>	PLL1时钟稳定中断（仅适用于CL、E518系列）
<code>RCU_INT_PLL2STB</code>	PLL2时钟稳定中断（仅适用于CL、E518系列）
<code>RCU_INT_PLLUSBSTB</code>	PLLUSB时钟稳定中断（仅适用于CL、E518系列）
<code>RCU_INT_IRC48MSTB</code>	IRC48M时钟稳定中断

## 枚举类型 `rcu_osci_type_enum`

表 3-715. 枚举类型 `rcu_osci_type_enum`

成员名称	功能描述
<code>RCU_HXTAL</code>	外部高速振荡器
<code>RCU_LXTAL</code>	外部低速振荡器
<code>RCU_IRC8M</code>	IRC8M振荡器
<code>RCU_IRC48M</code>	IRC48M振荡器
<code>RCU_IRC40K</code>	IRC40K振荡器
<code>RCU_PLL_CK</code>	锁相环时钟
<code>RCU_PLL1_CK</code>	锁相环1时钟（仅适用于CL、E518系列）
<code>RCU_PLL2_CK</code>	锁相环2时钟（仅适用于CL、E518系列）
<code>RCU_PLLUSB_CK</code>	锁相环USB时钟（仅适用于CL、E518系列）

## 枚举类型 `rcu_clock_freq_enum`

表 3-716. 枚举类型 `rcu_clock_freq_enum`

成员名称	功能描述
<code>CK_SYS</code>	系统时钟
<code>CK_AHB</code>	AHB时钟
<code>CK_APB1</code>	APB1时钟
<code>CK_APB2</code>	APB2时钟
<code>CK_USART</code>	USART5时钟

## 函数 `rcu_deinit`

函数`rcu_deinit`描述见下表：

表 3-717. 函数 `rcu_deinit`

函数名称	<code>rcu_deinit</code>
------	-------------------------

函数原形	void rcu_deinit(void);
功能描述	复位RCU
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 3-718. 函数 rcu\_periph\_clock\_enable

函数名称	rcu_periph_clock_enable
函数原形	void rcu_periph_clock_enable(rcu_periph_enum periph);
功能描述	使能外设时钟
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3-708. 枚举类型rcu_periph_enum</a>
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 3-719. 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3-708. 枚举类型rcu_periph_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the USART0 clock */
```

```
rcu_periph_clock_disable(RCU_USART0);
```

### 函数 rcu\_periph\_clock\_sleep\_enable

函数rcu\_periph\_clock\_sleep\_enable描述见下表：

**表 3-720. 函数 rcu\_periph\_clock\_sleep\_enable**

函数名称	rcu_periph_clock_sleep_enable
函数原形	void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);
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}	
periph	RCU外设，参考 <a href="#">表3-709. 枚举类型rcu_periph_sleep_enum</a>
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 3-721. 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3-709. 枚举类型rcu_periph_sleep_enum</a>
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\_periph\_reset\_enable

函数rcu\_periph\_reset\_enable描述见下表：

表 3-722. 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}	
periph_reset	RCU外设复位，参考 <a href="#">表3-710. 枚举类型rcu_periph_reset_enum</a>
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 3-723. 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}	
periph_reset	RCU外设复位，参考 <a href="#">表3-710. 枚举类型rcu_periph_reset_enum</a>

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 reset */
rcu_periph_reset_disable(RCU_SPI0RST);
```

### 函数 rcu\_bkp\_reset\_enable

函数rcu\_bkp\_reset\_enable描述见下表：

表 3-724. 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 3-725. 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 3-726. 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 RC8M	选择CK_IRC8M时钟作为CK_SYS时钟源
RCU_CKSYSSRC_ HXTAL	选择CK_HXTAL时钟作为CK_SYS时钟源
RCU_CKSYSSRC_ PLL	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 3-727. 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 3-728. 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_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 3-729. 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}	
<b>ck_apb1</b>	APB1预分频选择
<i>RCU_APB1_CKAHB_DIV1</i>	选择CK_AHB为CK_APB1
<i>RCU_APB1_CKAHB_DIV2</i>	选择CK_AHB / 2为CK_APB1
<i>RCU_APB1_CKAHB_DIV4</i>	选择CK_AHB / 4为CK_APB1
<i>RCU_APB1_CKAHB_DIV8</i>	选择CK_AHB / 8为CK_APB1
<i>RCU_APB1_CKAHB_DIV16</i>	选择CK_AHB / 16为CK_APB1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CK_AHB/16 as CK_APB1 */
```

```
rcu_apb1_clock_config(RCU_APB1_CKAHB_DIV16);
```

### 函数 rcu\_apb2\_clock\_config

函数rcu\_apb2\_clock\_config描述见下表：

表 3-730. 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}	
<b>ck_apb2</b>	APB2预分频选择
<i>RCU_APB2_CKAHB_DIV1</i>	选择CK_AHB为CK_APB2
<i>RCU_APB2_CKAHB_DIV2</i>	选择CK_AHB / 2为CK_APB2
<i>RCU_APB2_CKAHB_DIV4</i>	选择CK_AHB / 4为CK_APB2
<i>RCU_APB2_CKAHB_DIV8</i>	选择CK_AHB / 8为CK_APB2

<i>RCU_APB2_CK_AHB_DIV16</i>	选择CK_AHB / 16为CK_APB2
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\_ckout0\_config

函数rcu\_ckout0\_config描述见下表：

表 3-731. 函数 rcu\_ckout0\_config

函数名称	rcu_ckout0_config
函数原形	void rcu_ckout0_config(uint32_t ckout0_src);
功能描述	配置CKOUT0时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
ckout0_src	CKOUT0时钟源选择
<i>RCU_CKOUT0SRC_NONE</i>	无时钟输出
<i>RCU_CKOUT0SRC_CKSYS</i>	选择系统时钟CK_SYS
<i>RCU_CKOUT0SRC_IRC8M</i>	选择内部8M RC振荡器时钟
<i>RCU_CKOUT0SRC_HXTAL</i>	选择高速晶体振荡器时钟（HXTAL）
<i>RCU_CKOUT0SRC_CKPLL_DIV2</i>	选择（CK_PLL / 2）时钟
<i>RCU_CKOUT0SRC_CKPLL1</i>	选择CK_PLL1时钟
<i>RCU_CKOUT0SRC_CKPLL2_DIV2</i>	选择（CK_PLL2 / 2）时钟
<i>RCU_CKOUT0SRC_CKPLL2</i>	选择CK_PLL2时钟
<i>RCU_CKOUT0SRC_EXT1</i>	选择EXT1时钟
<i>RCU_CKOUT0SRC_CKIRC48M</i>	选择CK_IRC48M时钟

<i>RCU_CKOUT0SRC</i> <i>_CKIRC48M_DIV8</i>	选择CK_IRC48M / 8时钟
<i>RCU_CKOUT0SRC</i> <i>_CKPLLUSB_DIV32</i>	选择CK_PLLUSB / 32时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the HXTAL as CK_OUT0 clock source */
```

```
rcu_ckout0_config(RCU_CKOUT0SRC_HXTAL);
```

### 函数 rcu\_pll\_config

函数rcu\_pll\_config描述见下表：

表 3-732. 函数 rcu\_pll\_config

函数名称	rcu_pll_config
函数原形	void rcu_pll_config(uint32_t pll_src, uint32_t pll_mul);
功能描述	配置主PLL时钟
先决条件	-
被调用函数	-
输入参数{in}	
pll_src	PLL时钟源选择
<i>RCU_PLLSRC_IRC</i> <i>8M_DIV2</i>	(IRC8M / 2)被选择为PLL时钟的时钟源
<i>RCU_PLLSRC_HXT</i> <i>AL_IRC48M</i>	HXTAL时钟或者IRC48M时钟被选择为PLL时钟的时钟源
输入参数{in}	
pll_mul	PLL时钟倍频因子
<i>RCU_PLL_MULx</i>	PLL源时钟 * x (HD 型 x = 2..63, CL 型 x = 2..14, 16..64, 6.5)
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_pllpresel_config`

函数`rcu_pllpresel_config`描述见下表：

表 3-733. 函数 `rcu_pllpresel_config`

函数名称	<code>rcu_pllpresel_config</code>
函数原形	<code>void rcu_pllpresel_config(uint32_t pll_presel);</code>
功能描述	配置PLL时钟源预选
先决条件	-
被调用函数	-
输入参数{in}	
<code>pll_presel</code>	PLL时钟源
<code>RCU_PLLPRESRC_HXTAL</code>	PLL输入为HXTAL
<code>RCU_PLLPRESRC_IRC48M</code>	PLL输入为IRC48M
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PLL preselection */
```

```
rcu_pllpresel_config (RCU_PLLPRESRC_HXTAL);
```

函数 `rcu_predv0_config`（HD 产品）

函数`rcu_predv0_config`描述见下表：

表 3-734. 函数 `rcu_predv0_config`

函数名称	<code>rcu_predv0_config</code>
函数原形	<code>void rcu_predv0_config(uint32_t predv0_div);</code>
功能描述	配置PREDV0分频因子
先决条件	-
被调用函数	-
输入参数{in}	
<code>predv0_div</code>	PREDV0分频因子
<code>RCU_PREDV0_DIV_x</code>	PREDV0输入源时钟x分频（x = 1,2）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PREDV0 division factor */
```

```
rcu_predv0_config(RCU_PREDV0_DIV1);
```

### 函数 rcu\_predv0\_config (CL、E518 型产品)

函数rcu\_predv0\_config描述见下表：

表 3-735. 函数 rcu\_predv0\_config

函数名称	rcu_predv0_config
函数原形	void rcu_predv0_config(uint32_t predv0_source, uint32_t predv0_div);
功能描述	配置PREDV0分频因子
先决条件	-
被调用函数	-
输入参数{in}	
predv0_source	PREDV0输入时钟源
RCU_PREDV0SRC_HXTAL_IRC48M	HXTAL或者IRC48M被选择为PREDV0的时钟源
RCU_PREDV0SRC_CKPLL1	CK_PLL1被选择为PREDV0的时钟源
输入参数{in}	
predv0_div	PREDV0分频因子
RCU_PREDV0_DIV_x	PREDV0输入源时钟x分频 (x = 1..16)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PREDV0 division factor */
```

```
rcu_predv0_config(RCU_PREDV0SRC_HXTAL_IRC48M, RCU_PREDV0_DIV4);
```

### 函数 rcu\_predv1\_config

函数rcu\_predv1\_config描述见下表：

表 3-736. 函数 rcu\_predv1\_config

函数名称	rcu_predv1_config
函数原形	void rcu_predv1_config(uint32_t predv1_div);
功能描述	配置PREDV1分频因子
先决条件	-
被调用函数	-

输入参数{in}	
<b>predv1_div</b>	PREDV1分频因子
<i>RCU_PREDV1_DIV</i> x	PREDV1输入源时钟x分频 (x = 1..16)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the PREDV1 division factor */
```

```
rcu_predv1_config(RCU_PREDV1_DIV8);
```

### 函数 rcu\_pll1\_config

函数rcu\_pll1\_config描述见下表:

表 3-737. 函数 rcu\_pll1\_config

函数名称	rcu_pll1_config
函数原形	void rcu_pll1_config(uint32_t pll_mul);
功能描述	配置PLL1时钟
先决条件	-
被调用函数	-
输入参数{in}	
<b>pll_mul</b>	PLL1时钟倍频因子
<i>RCU_PLL1_MULx</i>	PLL1源时钟*x, (x = 8..14,16,20)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the PLL1 clock */
```

```
rcu_pll1_config(RCU_PLL1_MUL8);
```

### 函数 rcu\_pll2\_config

函数rcu\_pll2\_config描述见下表:

表 3-738. 函数 rcu\_pll2\_config

函数名称	rcu_pll2_config
函数原形	void rcu_pll2_config(uint32_t pll_mul);
功能描述	配置PLL2时钟
先决条件	-

被调用函数	-
输入参数{in}	
pll_mul	PLL2时钟倍频因子
RCU_PLL2_MULx	PLL2源时钟*x, (x = 8..14,16,20,18..32,40,34..64,80)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the PLL2 clock */
```

```
rcu_pll2_config(RCU_PLL2_MUL8);
```

### 函数 rcu\_pllusbpreisel\_config

函数rcu\_pllusbpreisel\_config描述见下表:

表 3-739. 函数 rcu\_pllusbpreisel\_config

函数名称	rcu_pllusbpreisel_config
函数原形	void rcu_pllusbpreisel_config(uint32_t pllusb_preisel);
功能描述	配置PLLUSB时钟源预选择
先决条件	-
被调用函数	-
输入参数{in}	
pllusb_preisel	PLLUSB时钟源预选择
RCU_PLLUSBPRE SRC_HXTAL	选择HXTAL为PLLUSB时钟源
RCU_PLLUSBPRE SRC_IRC48M	选择IRC48M为PLLUSB时钟源
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the PLLUSB clock source preselection */
```

```
rcu_pllusbpreisel_config(RCU_PLLUSBPRESEL_HXTAL);
```

### 函数 rcu\_pllusbpredv\_config

函数rcu\_pllusbpredv\_config描述见下表:

表 3-740. 函数 rcu\_pllusbpredv\_config

函数名称	rcu_pllusbpredv_config
------	------------------------

函数原形	void rcu_pllusbpredv_config(uint32_t pllusbpredv_source, uint32_t pllusbpredv_div);
功能描述	配置PLLUSBPREVDV分频系数和时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
pllusbpredv_source	PLLUSBPREDV输入时钟源选择
RCU_PLLUSBPRE DVSRC_HXTAL_IR C48M	选择HXTAL或IRC48M为PLLUSBPREDV输入时钟源
RCU_PLLUSBPRE DVSRC_CKPLL1	选择CK_PLL1为PLLUSBPREDV输入时钟源
输入参数{in}	
pllusbpredv_div	PLLUSBPREDV分频系数
RCU_PLLUSBPRE DV_DIVx	PLLUSBPREDV输入时钟源分频系数为x, (x = 1..15)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the PLLUSBPREDV division factor and clock source */
```

```
rcu_pllusbpredv_config(RCU_PLLUSBPREDVSRH_HXTAL_IRC48M, RCU_PLLUSBPREDV_DIV15);
```

### 函数 rcu\_pllusb\_config

函数rcu\_pllusb\_config描述见下表:

表 3-741. 函数 rcu\_pllusb\_config

函数名称	rcu_pllusb_config
函数原形	void rcu_pllusb_config(uint32_t pllusb_mul);
功能描述	配置PLLUSB时钟
先决条件	-
被调用函数	-
输入参数{in}	
pllusb_mul	PLLUSB时钟的倍频因子
RCU_PLLUSB_MU Lx	PLLUSB时钟源倍乘x, (x = 16,17..127)
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* configure the PLLUSB clock */
```

```
rcu_pllusb_config(RCU_PLLUSB_MUL16);
```

### 函数 rcu\_adc\_clock\_config

函数rcu\_adc\_clock\_config描述见下表：

表 3-742. 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的时钟分频系数
先决条件	-
被调用函数	-
输入参数{in}	
adc_psc	ADC分频因子
RCU_CKADC_CKA PB2_DIV2	CK_ADC = CK_APB2 / 2
RCU_CKADC_CKA PB2_DIV4	CK_ADC = CK_APB2 / 4
RCU_CKADC_CKA PB2_DIV6	CK_ADC = CK_APB2 / 6
RCU_CKADC_CKA PB2_DIV8	CK_ADC = CK_APB2 / 8
RCU_CKADC_CKA PB2_DIV12	CK_ADC = CK_APB2 / 12
RCU_CKADC_CKA PB2_DIV16	CK_ADC = CK_APB2 / 16
RCU_CKADC_CKA HB_DIV5	CK_ADC = CK_AHB / 5
RCU_CKADC_CKA HB_DIV6	CK_ADC = CK_AHB / 6
RCU_CKADC_CKA HB_DIV10	CK_ADC = CK_AHB / 10
RCU_CKADC_CKA HB_DIV20	CK_ADC = CK_AHB / 20
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the ADC prescaler factor */
```

```
rcu_adc_clock_config(RCU_CKADC_CKAPB2_DIV8);
```

### 函数 rcu\_usb\_clock\_config

函数rcu\_usb\_clock\_config描述见下表:

表 3-743. 函数 rcu\_usb\_clock\_config

函数名称	rcu_usb_clock_config
函数原形	void rcu_usb_clock_config(uint32_t usb_psc);
功能描述	配置USB的时钟分频系数
先决条件	-
被调用函数	-
输入参数{in}	
usb_psc	USB时钟分频系数
RCU_CKUSB_CKP LL_DIV1_5	$CK\_USBD/USBHS = CK\_PLL / 1.5$
RCU_CKUSB_CKP LL_DIV1	$CK\_USBD/USBHS = CK\_PLL / 1$
RCU_CKUSB_CKP LL_DIV2_5	$CK\_USBD/USBHS = CK\_PLL / 2.5$
RCU_CKUSB_CKP LL_DIV2	$CK\_USBD/USBHS = CK\_PLL / 2$
RCU_CKUSB_CKP LL_DIV3	$CK\_USBD/USBHS = CK\_PLL / 3$
RCU_CKUSB_CKP LL_DIV3_5	$CK\_USBD/USBHS = CK\_PLL / 3.5$
RCU_CKUSB_CKP LL_DIV4	$CK\_USBD/USBHS = CK\_PLL / 4$
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the USB prescaler factor */
```

```
rcu_usb_clock_config(RCU_CKUSB_CKPLL_DIV2_5);
```

### 函数 rcu\_rtc\_clock\_config

函数rcu\_rtc\_clock\_config描述见下表:

表 3-744. 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的时钟源选择
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_NO NE	没有时钟
RCU_RTCSRC_LX TAL	选择CK_LXTAL时钟作为RTC的时钟源
RCU_RTCSRC_IRC 40K	选择CK_IRC40K时钟作为RTC的时钟源
RCU_RTCSRC_HX TAL_DIV_128	选择CK_HXTAL / 128时钟作为RTC的时钟源
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\_shrtimer\_clock\_config

函数rcu\_shrtimer\_clock\_config描述见下表:

表 3-745. 函数 rcu\_shrtimer\_clock\_config

函数名称	rcu_shrtimer_clock_config
函数原形	void rcu_shrtimer_clock_config (uint32_t shrtimer_clock_source);
功能描述	配置HPTIEMR的时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
shrtimer_clock_source	SHRTIMER时钟源选择
RCU_SHRTIMERS RC_CKAPB2	选择APB2时钟作为SHRTIMER时钟
RCU_SHRTIMERS RC_CKSYS	选择系统时钟作为SHRTIMER的时钟源
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* configure the SHRTIMER clock source selection */
```

```
rcu_shrtimer_clock_config(RCU_SHRTIMERSRC_CKAPB2);
```

### 函数 rcu\_usart5\_clock\_config

函数rcu\_usart5\_clock\_config描述见下表：

表 3-746. 函数 rcu\_usart5\_clock\_config

函数名称	rcu_usart5_clock_config
函数原形	void rcu_usart5_clock_config(uint32_t usart5_clock_source);
功能描述	配置USART5的时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
usart5_clock_source	USART5时钟源选择
RCU_USART5SRC_CKAPB2	选择APB2时钟作为USART5时钟
RCU_USART5SRC_CKSYS	选择系统时钟作为USART5的时钟源
RCU_USART5SRC_LXTAL	选择LXTAL时钟作为USART5的时钟源
RCU_USART5SRC_IRC8M	选择IRC8M时钟作为USART5的时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the USART5 clock source selection */
```

```
rcu_usart5_clock_config(RCU_USART5SRC_CKAPB2);
```

### 函数 rcu\_i2c2\_clock\_config

函数rcu\_i2c2\_clock\_config描述见下表：

表 3-747. 函数 rcu\_i2c2\_clock\_config

函数名称	rcu_i2c2_clock_config
函数原形	void rcu_i2c2_clock_config(uint32_t i2c2_clock_source);
功能描述	配置I2C2的时钟源选择

先决条件	-
被调用函数	-
输入参数{in}	
<b>i2c2_clock_source</b>	I2C2时钟源选择
<i>RCU_I2C2SRC_CK</i> <i>APB1</i>	选择APB1时钟作为I2C2时钟
<i>RCU_I2C2SRC_CK</i> <i>SYS</i>	选择系统时钟作为I2C2的时钟源
<i>RCU_I2C2SRCSRC</i> <i>_CKIRC8M</i>	选择CK_IRC8M时钟作为I2C2的时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the I2C2 clock source selection */
```

```
rcu_i2c2_clock_config(RCU_I2C2SRC_CKAPB1);
```

### 函数 **rcu\_i2s1\_clock\_config**

函数rcu\_i2s1\_clock\_config描述见下表：

**表 3-748. 函数 rcu\_i2s1\_clock\_config**

函数名称	rcu_i2s1_clock_config
函数原形	void rcu_i2s1_clock_config(uint32_t i2s_clock_source);
功能描述	配置I2S1的时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
<b>i2s_clock_source</b>	I2S1时钟源选择
<i>RCU_I2S1SRC_CK</i> <i>SYS</i>	系统时钟被选择为I2S1时钟的时钟源
<i>RCU_I2S1SRC_CK</i> <i>PLL2_MUL2</i>	(CK_PLL2 * 2) 被选择为I2S1时钟的时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the I2S1 clock source selection */
```

```
rcu_i2s1_clock_config(RCU_I2S1SRC_CKPLL2_MUL2);
```

函数 `rcu_i2s2_clock_config`

函数 `rcu_i2s2_clock_config` 描述见下表：

表 3-749. 函数 `rcu_i2s2_clock_config`

函数名称	<code>rcu_i2s2_clock_config</code>
函数原形	<code>void rcu_i2s2_clock_config(uint32_t i2s_clock_source);</code>
功能描述	配置I2S2的时钟源选择
先决条件	-
被调用函数	
输入参数{in}	
<code>i2s_clock_source</code>	I2S2时钟源选择
<code>RCU_I2S2SRC_CKSYS</code>	系统时钟被选择为I2S2时钟的时钟源
<code>RCU_I2S2SRC_CKPLL2_MUL2</code>	( $CK\_PLL2 * 2$ ) 被选择为I2S2时钟的时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the I2S2 clock source selection */
```

```
rcu_i2s2_clock_config(RCU_I2S2SRC_CKPLL2_MUL2);
```

函数 `rcu_ck48m_clock_config`

函数 `rcu_ck48m_clock_config` 描述见下表：

表 3-750. 函数 `rcu_ck48m_clock_config`

函数名称	<code>rcu_ck48m_clock_config</code>
函数原形	<code>void rcu_ck48m_clock_config(uint32_t ck48m_clock_source);</code>
功能描述	配置CK48M时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
<code>ck48m_clock_source</code>	CK48M时钟源
<code>RCU_CK48MSRC_CKPLL</code>	选择CK_PLL为CK_48M时钟源
<code>RCU_CK48MSRC_IRC48M</code>	选择CK_IRC48M为CK48M时钟源
<code>RCU_CK48MSRC_CKPLLUSB</code>	选择PLLUSB时钟作为CK48M时钟源

<i>RCU_CK48MSRC_</i> <i>CKPLL2</i>	选择CKPLL2作为CK48M时钟源
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* configure the CK48M clock source selection */
```

```
rcu_ck48m_clock_config (RCU_CK48MSRC_IRC48M);
```

### 函数 rcu\_usbhsel\_config

函数rcu\_usbhsel\_config描述见下表：

表 3-751. 函数 rcu\_usbhsel\_config

函数名称	rcu_usbhsel_config
函数原形	void rcu_usbhsel_config(uint32_t usbhsel_clock_source);
功能描述	配置USBHSSEL的时钟源选择
先决条件	-
被调用函数	
输入参数{in}	
usbhsel_clock_source	USBHSSEL时钟源选择
<i>RCU_USBHSSRC_</i> <i>48M</i>	选择48M时钟作为USBHS时钟的时钟源
<i>RCU_CK48MSRC_</i> <i>60M</i>	选择60M时钟作为USBHS时钟的时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the USBHSSEL source clock selection */
```

```
rcu_usbhsel_config(RCU_USBHSSRC_48M);
```

### 函数 rcu\_usbdv\_config

函数rcu\_usbdv\_config描述见下表：

表 3-752. 函数 rcu\_usbdv\_config

函数名称	rcu_usbdv_config
函数原形	void rcu_usbdv_config(uint32_t usbhs_dv);

功能描述	配置USBHSDV分频系数
先决条件	-
被调用函数	-
输入参数{in}	
usbhs_dv	USBHSDV分频系数
RCU_USBHSDV_DIV16 Vx	USBHSDV输入时钟源分频系数为x，（x=2,4,...,16）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the USBHSDV division factor */
```

```
rcu_usbdv_config(RCU_USBHSDV_DIV16);
```

### 函数 rcu\_lxtal\_drive\_capability\_config

函数rcu\_lxtal\_drive\_capability\_config描述见下表：

表 3-753. 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	低驱动能力
RCU_LXTAL_MED_LOW_DRI	中低驱动能力
RCU_LXTAL_MED_HIGH_DRI	中高驱动能力
RCU_LXTAL_HIGH_DRI	高驱动能力
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the LXTAL drive capability */
```

rcu\_lxtal\_drive\_capability\_config (RCU\_LXTAL\_LOWDR1);

### 函数 rcu\_osci\_stab\_wait

函数rcu\_osci\_stab\_wait描述见下表：

表 3-754. 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3-715. 枚举类型rcu_osci_type_enum</a>
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 3-755. 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3-715. 枚举类型rcu_osci_type_enum</a>
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 3-756. 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3-715. 枚举类型rcu_osc_i_type_enum</a>
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\_irc8m\_adjust\_value\_set

函数rcu\_irc8m\_adjust\_value\_set描述见下表:

**表 3-757. 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\_osc\_bypass\_mode\_enable

函数rcu\_osc\_bypass\_mode\_enable描述见下表:

表 3-758. 函数 rcu\_osc\_bypass\_mode\_enable

函数名称	rcu_osc_bypass_mode_enable
函数原形	void rcu_osc_bypass_mode_enable(rcu_osc_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}	
osci	振荡器类型, 参考 <a href="#">表3-715. 枚举类型rcu_osc_type_enum</a>
RCU_HXTAL	高速晶体振荡器
RCU_LXTAL	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_osc_bypass_mode_enable(RCU_HXTAL);
```

## 函数 rcu\_osc\_bypass\_mode\_disable

函数rcu\_osc\_bypass\_mode\_disable描述见下表:

表 3-759. 函数 rcu\_osc\_bypass\_mode\_disable

函数名称	rcu_osc_bypass_mode_disable
函数原形	void rcu_osc_bypass_mode_disable(rcu_osc_type_enum osci);
功能描述	禁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}	
osci	振荡器类型, 参考 <a href="#">表3-715. 枚举类型rcu_osc_type_enum</a>
RCU_HXTAL	高速晶体振荡器
RCU_LXTAL	低速晶体振荡器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the high speed crystal oscillator bypass mode */
```

rcu\_osc\_bypass\_mode\_disable(RCU\_HXTAL);

### 函数 rcu\_hxtal\_clock\_monitor\_enable

函数rcu\_hxtal\_clock\_monitor\_enable描述见下表：

表 3-760. 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 3-761. 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\_deepsleep\_voltage\_set

函数rcu\_deepsleep\_voltage\_set描述见下表:

表 3-762. 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值
先决条件	-
被调用函数	-
输入参数{in}	
dsvol	深度睡眠模式电压值
RCU_DEEPSLEEP_V_1_0	在深度睡眠模式下内核电压为1.0V
RCU_DEEPSLEEP_V_0_9	在深度睡眠模式下内核电压为0.9V
RCU_DEEPSLEEP_V_0_8	在深度睡眠模式下内核电压为0.8V
RCU_DEEPSLEEP_V_0_7	在深度睡眠模式下内核电压为0.7V
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set the 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 3-763. 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	获取系统时钟、总线频率
先决条件	-
被调用函数	-
输入参数{in}	
clock	要获取的时钟频率, 参考 <a href="#">表3-716. 枚举类型rcu_clock_freq_enum</a>
CK_SYS	系统时钟
CK_AHB	AHB时钟
CK_APB1	APB1时钟

<i>CK_APB2</i>	APB2时钟
<i>CK_USART</i>	USART5时钟
输出参数{out}	
-	-
返回值	
<b>ck_freq</b>	系统时钟/AHB时钟/APB1时钟/APB2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 3-764. 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}	
<b>flag</b>	时钟稳定和外设复位标志，参考 <a href="#">表3-711. 枚举类型rcu_flag_enum</a>
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	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 3-765. 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 3-766. 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3-712. 枚举类型rcu_int_flag_enum</a>
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 3-767. 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);
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	时钟稳定和阻塞中断标志清除, 参考 <a href="#">表3-713. 枚举类型 rcu_int_flag_clear_enum</a>
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 3-768. 函数 rcu\_interrupt\_enable

函数名称	rcu_interrupt_enable
函数原形	void rcu_interrupt_enable(rcu_int_enum interrupt);
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}	
interrupt	时钟稳定中断,参考 <a href="#">表3-714. 枚举类型rcu_int_enum</a>
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 3-769. 函数 rcu\_interrupt\_disable

函数名称	rcu_interrupt_disable
函数原形	void rcu_interrupt_disable(rcu_int_enum interrupt);
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}	
interrupt	时钟稳定中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the HXTAL stabilization interrupt */
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

## 3.22. 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[3.22.1](#)描述了RTC的寄存器列表，章节[3.22.2](#)对RTC库函数进行说明。

### 3.22.1. 外设寄存器描述

RTC寄存器列表如下表所示：

表 3-770. 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

### 3.22.2. 外设库函数描述

RTC库函数列表如下表所示：

表 3-771. 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

库函数名称	库函数描述
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

### 函数 rtc\_configuration\_mode\_enter

函数rtc\_configuration\_mode\_enter描述见下表：

表 3-772. 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 3-773. 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 3-774. 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 3-775. 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 3-776. 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}	
RTC counter value	RTC计时器的值
返回值	
-	-

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 3-777. 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 3-778. 函数 rtc\_prescaler\_set

函数名称	rtc_interrupt_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 3-779. 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 3-780. 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	
RTC divider value	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 3-781. 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 3-782. 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 3-783. 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 D	秒中断标志位
RTC_FLAG_ALARM	闹钟中断标志位
RTC_FLAG_OVERF 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 3-784. 函数 `rtc_flag_clear`

函数名称	<code>rtc_flag_clear</code>
函数原型	<code>void rtc_flag_clear(uint32_t flag);</code>
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}	
<b>flag</b>	待清除的RTC标志位
<code>RTC_FLAG_SECON</code> <code>D</code>	秒中断标志位
<code>RTC_FLAG_ALARM</code>	闹钟中断标志位
<code>RTC_FLAG_OVERF</code> <code>LOW</code>	溢出中断标志位
<code>RTC_FLAG_RSYN</code>	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);
```

## 3.23. SDIO

安全的数字输入/输出接口（SDIO）定义了SD卡、SD I/O卡、多媒体卡（MMC）和CE-ATA卡主机接口，提供AHB系统总线与SD存储卡、SD I/O卡、MMC和CE-ATA设备之间的数据传输。章节[3.23.1](#)描述了SDIO的寄存器列表，章节[3.23.2](#)对SDIO库函数进行说明。

### 3.23.1. 外设寄存器说明

SDIO寄存器列表如下表所示：

表 3-785. SDIO 寄存器

寄存器名称	寄存器描述
<code>SDIO_PWRCTL</code>	电源控制寄存器
<code>SDIO_CLKCTL</code>	时钟控制寄存器
<code>SDIO_CMDAGMT</code>	命令参数寄存器
<code>SDIO_CMDCTL</code>	命令控制寄存器
<code>SDIO_RSPCMDIDX</code>	命令索引响应寄存器
<code>SDIO_RESPx</code> <code>x=0..3</code>	响应寄存器

寄存器名称	寄存器描述
SDIO_DATATO	数据超时寄存器
SDIO_DATALEN	数据长度寄存器
SDIO_DATACTL	数据控制寄存器
SDIO_DATACNT	数据计数寄存器
SDIO_STAT	状态寄存器
SDIO_INTC	中断清除寄存器
SDIO_INTEN	中断使能寄存器
SDIO_FIFOCNT	FIFO计数寄存器
SDIO_FIFO	FIFO数据寄存器

### 3.23.2. 外设库函数说明

SDIO库函数列表如下表所示：

**表 3-786. SDIO 库函数**

库函数名称	库函数描述
sdio_deinit	复位SDIO
sdio_clock_config	配置SDIO时钟
sdio_hardware_clock_enable	使能硬件时钟控制
sdio_hardware_clock_disable	禁能硬件时钟控制
sdio_bus_mode_set	设置多种SDIO卡总线模式
sdio_power_state_set	设置SDIO电源状态
sdio_power_state_get	获取SDIO电源状态
sdio_clock_enable	使能SDIO_CLK时钟
sdio_clock_disable	禁能SDIO_CLK时钟
sdio_command_response_config	配置命令和响应
sdio_wait_type_set	设置命令状态机等待类型
sdio_csm_enable	使能命令状态机
sdio_csm_disable	禁能命令状态机
sdio_command_index_get	获取上一次响应的命令索引
sdio_response_get	获取上一次响应的接收命令
sdio_data_config	配置数据超时、数据长度和数据块大小
sdio_data_transfer_config	配置数据传输模式和方向
sdio_dsm_enable	使能数据传输的数据状态机
sdio_dsm_disable	禁能数据传输的数据状态机
sdio_data_write	在发送FIFO里写入数据（一个字）
sdio_data_read	在接收FIFO里读取数据（一个字）
sdio_data_counter_get	获取要传输到卡的剩余数据字节的数目
sdio_fifo_counter_get	从FIFO中获取要写入或读取的字数
sdio_dma_enable	使能SDIO的DMA请求
sdio_dma_disable	禁能SDIO的DMA请求
sdio_flag_get	获取SDIO的标志位状态

库函数名称	库函数描述
sdio_flag_clear	清除SDIO的标志位状态
sdio_interrupt_enable	使能SDIO中断
sdio_interrupt_disable	禁能SDIO中断
sdio_interrupt_flag_get	获取SDIO的中断标志位状态
sdio_interrupt_flag_clear	清除SDIO的中断标志位状态
sdio_readwait_enable	使能读等待模式（仅限SD I/O模式）
sdio_readwait_disable	禁能读等待模式（仅限SD I/O模式）
sdio_stop_readwait_enable	使能停止读等待过程的功能（仅限SD I/O模式）
sdio_stop_readwait_disable	禁能停止读等待过程的功能（仅限SD I/O模式）
sdio_readwait_type_set	设置读等待类型（仅限SD I/O模式）
sdio_operation_enable	使能SD I/O模式特定操作（仅限SD I/O模式）
sdio_operation_disable	禁能SD I/O模式特定操作（仅限SD I/O模式）
sdio_suspend_enable	使能SD I/O暂停模式（仅限SD I/O模式）
sdio_suspend_disable	禁能SD I/O暂停模式（仅限SD I/O模式）
sdio_ceata_command_enable	使能CE-ATA命令(仅限CE-ATA模式)
sdio_ceata_command_disable	禁能CE-ATA命令(仅限CE-ATA模式)
sdio_ceata_interrupt_enable	使能CE-ATA中断(仅限CE-ATA模式)
sdio_ceata_interrupt_disable	禁能CE-ATA中断(仅限CE-ATA模式)
sdio_ceata_command_completion_enable	使能CE-ATA命令完成信号(仅限CE-ATA模式)
sdio_ceata_command_completion_disable	禁能CE-ATA命令完成信号(仅限CE-ATA模式)

## 函数 sdio\_deinit

函数sdio\_deinit描述见下表：

**表 3-787. 函数 sdio\_deinit**

函数名称	sdio_deinit
函数原形	void sdio_deinit(void);
功能描述	复位SDIO
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 SDIO */
```

```
sdio_deinit();
```

## 函数 sdio\_clock\_config

函数sdio\_clock\_config描述见下表：

**表 3-788. 函数 sdio\_clock\_config**

函数名称	sdio_clock_config
函数原形	void sdio_clock_config(uint32_t clock_edge, uint32_t clock_bypass, uint32_t clock_powersave, uint16_t clock_division);
功能描述	配置SDIO时钟
先决条件	-
被调用函数	-
输入参数{in}	
<b>clock_edge</b>	SDIO_CLK时钟边沿选择
SDIO_SDIOLCKED GE_RISING	选择SDIOCLK的上升沿产生SDIO_CLK
SDIO_SDIOLCKED GE_FALLING	选择SDIOCLK的下降沿产生SDIO_CLK
输入参数{in}	
<b>clock_bypass</b>	旁路时钟使能
SDIO_CLOCKBYPASS_ENABLE	使能旁路时钟
SDIO_CLOCKBYPASS_DISABLE	失能旁路时钟
输入参数{in}	
<b>clock_powersave</b>	SDIO_CLK时钟动态开启/关闭以节省功耗
SDIO_CLOCKPWRSAVE_ENABLE	SDIO_CLK时钟在总线空闲时关闭
SDIO_CLOCKPWRSAVE_DISABLE	SDIO_CLK时钟总是开启
输入参数{in}	
<b>clock_division</b>	时钟分频，小于512
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the SDIO clock */
```

```
sdio_clock_config(SDIO_SDIOLCKEDGE_RISING, SDIO_CLOCKBYPASS_DISABLE, SDIO_CLOCKPWRSAVE_DISABLE, SD_CLK_DIV_TRANS);
```

## 函数 sdio hardware clock enable

函数sdio hardware clock enable描述见下表：

**表 3-789. 函数 sdio hardware clock enable**

函数名称	sdio hardware clock enable
函数原形	void sdio hardware clock enable(void);
功能描述	使能硬件时钟控制
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 hardware clock control */
sdio hardware clock enable();
```

## 函数 sdio hardware clock disable

函数sdio hardware clock disable描述见下表：

**表 3-790. 函数 sdio hardware clock disable**

函数名称	sdio hardware clock disable
函数原形	void sdio hardware clock disable(void);
功能描述	禁能硬件时钟控制
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 hardware clock control */
sdio hardware clock disable();
```

**函数 sdio\_bus\_mode\_set**

函数sdio\_bus\_mode\_set描述见下表：

**表 3-791. 函数 sdio\_bus\_mode\_set**

函数名称	sdio_bus_mode_set
函数原形	void sdio_bus_mode_set(uint32_t bus_mode);
功能描述	设置多种SDIO卡总线模式
先决条件	-
被调用函数	-
输入参数{in}	
bus_mode	SDIO卡总线模式
SDIO_BUSMODE_1 BIT	1位SDIO卡总线模式
SDIO_BUSMODE_4 BIT	4位SDIO卡总线模式
SDIO_BUSMODE_8 BIT	8位SDIO卡总线模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SDIO bus mode */
```

```
sdio_bus_mode_set(SDIO_BUSMODE_1BIT);
```

**函数 sdio\_power\_state\_set**

函数sdio\_power\_state\_set描述见下表：

**表 3-792. 函数 sdio\_power\_state\_set**

函数名称	sdio_power_state_set
函数原形	void sdio_power_state_set(uint32_t power_state);
功能描述	设置SDIO电源状态
先决条件	-
被调用函数	-
输入参数{in}	
power_state	SDIO电源状态
SDIO_POWER_ON	SDIO上电
SDIO_POWER_OFF	SDIO断电
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* set SDIO power state */
```

```
sdio_power_state_set(SDIO_POWER_ON);
```

### 函数 **sdio\_power\_state\_get**

函数sdio\_power\_state\_get描述见下表：

**表 3-793. 函数 sdio\_power\_state\_get**

函数名称	sdio_power_state_get
函数原形	uint32_t sdio_power_state_get(void);
功能描述	获取SDIO电源状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	SDIO_POWER_ON / SDIO_POWER_OFF

例如：

```
/* get the SDIO power state */
```

```
uint32_t sdio_power_value;
```

```
sdio_power_value = sdio_power_state_get();
```

### 函数 **sdio\_clock\_enable**

函数sdio\_clock\_enable描述见下表：

**表 3-794. 函数 sdio\_clock\_enable**

函数名称	sdio_clock_enable
函数原形	void sdio_clock_enable(void);
功能描述	使能SDIO_CLK时钟
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 SDIO_CLK clock output */
```

```
sdio_clock_enable();
```

### 函数 **sdio\_clock\_disable**

函数sdio\_clock\_disable描述见下表：

**表 3-795. 函数 sdio\_clock\_disable**

函数名称	sdio_clock_disable
函数原形	void sdio_clock_disable(void);
功能描述	禁能SDIO_CLK时钟
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 SDIO_CLK clock output */
```

```
sdio_clock_disable();
```

### 函数 **sdio\_command\_response\_config**

函数sdio\_command\_response\_config描述见下表：

**表 3-796. 函数 sdio\_command\_response\_config**

函数名称	sdio_command_response_config
函数原形	void sdio_command_response_config(uint32_t cmd_index, uint32_t cmd_argument, uint32_t response_type);
功能描述	配置命令和响应
先决条件	-
被调用函数	-
输入参数{in}	
cmd_index	命令索引，请参阅相关规范
输入参数{in}	
cmd_argument	命令参数，请参阅相关规范
输入参数{in}	

<b>response_type</b>	命令响应类型
SDIO_RESPONSETYPE_NO	无响应
SDIO_RESPONSETYPE_SHORT	短响应
SDIO_RESPONSETYPE_LONG	长响应
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CMD2(SD_CMD_ALL_SEND_CID) command response config*/
```

```
sdio_command_response_config(SD_CMD_ALL_SEND_CID, (uint32_t)0x0, SDIO_RESPONSETYPE_LONG);
```

### 函数 sdio\_wait\_type\_set

函数sdio\_wait\_type\_set描述见下表：

表 3-797. 函数 sdio\_wait\_type\_set

<b>函数名称</b>	sdio_wait_type_set
<b>函数原形</b>	void sdio_wait_type_set(uint32_t wait_type);
<b>功能描述</b>	设置命令状态机等待类型
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>wait_type</b>	等待类型
SDIO_WAITTYPE_NO	不等待中断
SDIO_WAITTYPE_INTERRUPT	等待中断
SDIO_WAITTYPE_DATAEND	等待数据传输结束
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the command state machine wait type */
```

```
sdio_wait_type_set(SDIO_WAITTYPE_NO);
```

## 函数 sdio\_csm\_enable

函数sdio\_csm\_enable描述见下表：

**表 3-798. 函数 sdio\_csm\_enable**

函数名称	sdio_csm_enable
函数原形	void sdio_csm_enable(void);
功能描述	使能命令状态机
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 CSM(command state machine) */
```

```
sdio_csm_enable();
```

## 函数 sdio\_csm\_disable

函数sdio\_csm\_disable描述见下表：

**表 3-799. 函数 sdio\_csm\_disable**

函数名称	sdio_csm_disable
函数原形	void sdio_csm_disable(void);
功能描述	禁能命令状态机
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 CSM(command state machine) */
```

```
sdio_csm_disable();
```

**函数 sdio\_command\_index\_get**

函数sdio\_command\_index\_get描述见下表：

**表 3-800. 函数 sdio\_command\_index\_get**

函数名称	sdio_command_index_get
函数原形	uint8_t sdio_command_index_get(void);
功能描述	获取上一次响应的命令索引
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint8_t	上一次响应的命令索引

例如：

```
/* get SDIO command index */
uint8_t sdio_commond_value;

sdio_commond_value = sdio_command_index_get();
```

**函数 sdio\_response\_get**

函数sdio\_response\_get描述见下表：

**表 3-801. 函数 sdio\_response\_get**

函数名称	sdio_response_get
函数原形	uint32_t sdio_response_get(uint32_t responsex);
功能描述	获取上一次响应的接收命令
先决条件	-
被调用函数	-
输入参数{in}	
responsex	SDIO响应
SDIO_RESPONSE0	卡响应 [31:0]/卡响应 [127:96]
SDIO_RESPONSE1	卡响应 [95:64]
SDIO_RESPONSE2	卡响应 [63:32]
SDIO_RESPONSE3	卡响应 [31:1]，加上位0
输出参数{out}	
-	-
返回值	
uint32_t	上一次响应的接收命令

例如：

```
/* store the CID0 numbers */
```

```
uint32_t sdio_cid[0];
```

```
sdio_cid[0] = sdio_response_get(SDIO_RESPONSE0);
```

### 函数 **sdio\_data\_config**

函数 **sdio\_data\_config** 描述见下表：

**表 3-802. 函数 **sdio\_data\_config****

函数名称	sdio_data_config
函数原形	void sdio_data_config(uint32_t data_timeout, uint32_t data_length, uint32_t data_blocksize);
功能描述	配置数据超时、数据长度和数据块大小
先决条件	-
被调用函数	-
输入参数{in}	
data_timeout	卡总线时钟周期中的数据超时周期
输入参数{in}	
data_length	要传输的数据字节数
输入参数{in}	
data_blocksize	块传输中数据块的大小
SDIO_DATABLOCK SIZE_1BYTE	块大小 = 1字节
SDIO_DATABLOCK SIZE_2BYTES	块大小 = 2字节
SDIO_DATABLOCK SIZE_4BYTES	块大小 = 4字节
SDIO_DATABLOCK SIZE_8BYTES	块大小 = 8字节
SDIO_DATABLOCK SIZE_16BYTES	块大小 = 16字节
SDIO_DATABLOCK SIZE_32BYTES	块大小 = 32字节
SDIO_DATABLOCK SIZE_64BYTES	块大小 = 64字节
SDIO_DATABLOCK SIZE_128BYTES	块大小 = 128字节
SDIO_DATABLOCK SIZE_256BYTES	块大小 = 256字节
SDIO_DATABLOCK SIZE_512BYTES	块大小 = 512字节
SDIO_DATABLOCK SIZE_1024BYTES	块大小 = 1024字节

SDIO_DATABLOCK SIZE_2048BYTES	块大小 = 2048字节
SDIO_DATABLOCK SIZE_4096BYTES	块大小 = 4096字节
SDIO_DATABLOCK SIZE_8192BYTES	块大小 = 8192字节
SDIO_DATABLOCK SIZE_16384BYTES	块大小 = 16384字节
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure SDIO data */
```

```
sdio_data_config(0, 0, SDIO_DATABLOCKSIZE_1BYTE);
```

### 函数 sdio\_data\_transfer\_config

函数sdio\_data\_transfer\_config描述见下表:

表 3-803. 函数 sdio\_data\_transfer\_config

函数名称	sdio_data_transfer_config
函数原形	void sdio_data_transfer_config(uint32_t transfer_mode, uint32_t transfer_direction);
功能描述	配置数据传输模式和方向
先决条件	-
被调用函数	-
输入参数{in}	
transfer_mode	数据传输模式
SDIO_TRANSMOD E_BLOCK	块传输模式
SDIO_TRANSMOD E_STREAM	流传输或SDIO多字节传输模式
输入参数{in}	
transfer_direction	数据传输方向
SDIO_TRANSDIRE CTION_TOCARD	写数据到卡上
SDIO_TRANSDIRE CTION_TOSDIO	从卡中读取数据
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* configure SDIO data transmisson */

sdio_data_transfer_config(SDIO_TRANSDIRECTION_TOSDIO,
SDIO_TRANSMODE_BLOCK);
```

### 函数 **sdio\_dsm\_enable**

函数sdio\_dsm\_enable描述见下表：

**表 3-804. 函数 sdio\_dsm\_enable**

函数名称	sdio_dsm_enable
函数原形	void sdio_dsm_enable(void);
功能描述	使能数据传输的数据状态机
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 DSM(data state machine) */

sdio_dsm_enable();
```

### 函数 **sdio\_dsm\_disable**

函数sdio\_dsm\_disable描述见下表：

**表 3-805. 函数 sdio\_dsm\_disable**

函数名称	sdio_dsm_disable
函数原形	void sdio_dsm_disable(void);
功能描述	禁能数据传输的数据状态机
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 DSM(data state machine) */
```

```
sdio_dsm_disable();
```

### 函数 **sdio\_data\_write**

函数sdio\_data\_write描述见下表：

**表 3-806. 函数 sdio\_data\_write**

函数名称	sdio_data_write
函数原形	void sdio_data_write(uint32_t data);
功能描述	在发送FIFO里写入数据（一个字）
先决条件	-
被调用函数	-
输入参数{in}	
<b>data</b>	往卡里写入32位数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write data(one word) to the transmit FIFO */
```

```
sdio_data_write(0x0000 0001);
```

### 函数 **sdio\_data\_read**

函数sdio\_data\_read描述见下表：

**表 3-807. 函数 sdio\_data\_read**

函数名称	sdio_data_read
函数原形	uint32_t sdio_data_read(void);
功能描述	在接收FIFO里读取数据（一个字）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	接收的数据

例如：

```
/* read data(one word) from the receive FIFO */
```

```
sdio_data_read();
```

### 函数 sdio\_data\_counter\_get

函数sdio\_data\_counter\_get描述见下表：

表 3-808. 函数 sdio\_data\_counter\_get

函数名称	sdio_data_counter_get
函数原形	uint32_t sdio_data_counter_get(void);
功能描述	获取要传输到卡的剩余数据字节的数目
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	要传输的剩余数据字节数

例如：

```
/* get the number of remaining data bytes to be transferred to card */
```

```
uint32_t sdio_data_value;
```

```
sdio_data_value = sdio_data_counter_get();
```

### 函数 sdio\_fifo\_counter\_get

函数sdio\_fifo\_counter\_get描述见下表：

表 3-809. 函数 sdio\_data\_counter\_get

函数名称	sdio_fifo_counter_get
函数原形	uint32_t sdio_fifo_counter_get(void);
功能描述	从FIFO中获取要写入或读取的字数
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	剩余字数

例如：

```
/* get the number of words remaining to be written or read from FIFO */
```

```
uint32_t sdio_fifo_value;
```

```
sdio_fifo_value = sdio_fifo_counter_get();
```

## 函数 sdio\_dma\_enable

函数sdio\_dma\_enable描述见下表：

**表 3-810. 函数 sdio\_dma\_enable**

函数名称	sdio_dma_enable
函数原形	void sdio_dma_enable(void);
功能描述	使能SDIO的DMA请求
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 SDIO DMA */
```

```
sdio_dma_enable();
```

## 函数 sdio\_dma\_disable

函数sdio\_dma\_disable描述见下表：

**表 3-811. 函数 sdio\_dma\_disable**

函数名称	sdio_dma_disable
函数原形	void sdio_dma_disable(void);
功能描述	禁能SDIO的DMA请求
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 SDIO DMA */
```

```
sdio_dma_disable();
```

### 函数 **sdio\_flag\_get**

函数sdio\_flag\_get描述见下表:

**表 3-812. 函数 sdio\_flag\_get**

函数名称	sdio_flag_get
函数原形	FlagStatus sdio_flag_get(uint32_t flag);
功能描述	获取SDIO的标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	SDIO标志位状态
SDIO_FLAG_CCRCERR	命令响应已接收（CRC检测失败）
SDIO_FLAG_DTCCRERR	数据块已发送/已接收（CRC检测失败）
SDIO_FLAG_CMDTMOU	命令响应超时
SDIO_FLAG_DTTMOU	数据超时
SDIO_FLAG_TXUR	发送FIFO下溢错误发生
SDIO_FLAG_RXOR	接收FIFO上溢错误发生
SDIO_FLAG_CMDRECV	命令响应已接收（CRC检测通过）
SDIO_FLAG_CMDS	命令已发送（不需响应）
SDIO_FLAG_DTEND	数据结束（数据计数器，SDIO_DATACNT为零）
SDIO_FLAG_STBIT	总线上起始位错误
SDIO_FLAG_DTBLKEND	数据块已发送/已接收（CRC检测通过）
SDIO_FLAG_CMDRUN	正在传输命令
SDIO_FLAG_TXRUN	正在传输数据
SDIO_FLAG_RXRUN	正在接收数据

<i>SDIO_FLAG_TFH</i>	发送FIFO半空：至少还有8个字可被写入到FIFO中
<i>SDIO_FLAG_RFH</i>	接收FIFO半满：FIFO中至少还有8个字可被读取
<i>SDIO_FLAG_TFF</i>	发送FIFO为满
<i>SDIO_FLAG_RFF</i>	接收FIFO为满
<i>SDIO_FLAG_TFE</i>	发送FIFO为空
<i>SDIO_FLAG_RFE</i>	接收FIFO为空
<i>SDIO_FLAG_TXDTVAL</i>	发送FIFO中的数据有效
<i>SDIO_FLAG_RXDTVAL</i>	接收FIFO中的数据有效
<i>SDIO_FLAG_SDIOINT</i>	SD I/O中断已接收
<i>SDIO_FLAG_ATAEND</i>	CE-ATA命令完成信号已接收（仅用于CMD61）
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the flags state of SDIO */
```

```
FlagStatus flag_value;
```

```
flag_value = sdio_flag_get(SDIO_FLAG_RFH);
```

### 函数 **sdio\_flag\_clear**

函数sdio\_flag\_clear描述见下表：

**表 3-813. 函数 sdio\_flag\_clear**

函数名称	sdio_flag_clear
函数原形	void sdio_flag_clear(uint32_t flag);
功能描述	清除SDIO的标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	SDIO标志位状态
<i>SDIO_FLAG_CCRCERR</i>	命令响应已接收（CRC检测失败）
<i>SDIO_FLAG_DTCCRERR</i>	数据块已发送/已接收（CRC检测失败）
<i>SDIO_FLAG_CMDTMOU</i>	命令响应超时
<i>SDIO_FLAG_DTTM</i>	数据超时

OUT	
SDIO_FLAG_TXUR E	发送FIFO下溢错误发生
SDIO_FLAG_RXOR E	接收FIFO上溢错误发生
SDIO_FLAG_CMDR ECV	命令响应已接收（CRC检测通过）
SDIO_FLAG_CMDS END	命令已发送（不需响应）
SDIO_FLAG_DTEN D	数据结束（数据计数器，SDIO_DATACNT为零）
SDIO_FLAG_STBIT E	总线上起始位错误
SDIO_FLAG_DTBL KEND	数据块已发送/已接收（CRC检测通过）
SDIO_FLAG_SDIOI NT	SD I/O中断已接收
SDIO_FLAG_ATAE ND	CE-ATA命令完成信号已接收（仅用于CMD61）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the pending flags of SDIO */
sdio_flag_clear(SDIO_FLAG_DTCRCERR);
```

### 函数 sdio\_interrupt\_enable

函数sdio\_interrupt\_enable描述见下表：

表 3-814. 函数 sdio\_interrupt\_enable

函数名称	sdio_interrupt_enable
函数原形	void sdio_interrupt_enable(uint32_t int_flag);
功能描述	使能SDIO中断
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	SDIO中断标志位状态
SDIO_INT_CCRCE RR	命令响应CRC错误中断
SDIO_INT_DTCRC	数据CRC错误中断

<i>ERR</i>	
<i>SDIO_INT_CMDTMO OUT</i>	命令响应超时中断
<i>SDIO_INT_DTTMO UT</i>	数据超时中断
<i>SDIO_INT_TXURE</i>	发送FIFO下溢错误中断
<i>SDIO_INT_RXORE</i>	接收FIFO上溢错误中断
<i>SDIO_INT_CMDRE CV</i>	命令响应已接收中断
<i>SDIO_INT_CMDSE ND</i>	命令已发送中断
<i>SDIO_INT_DTEND</i>	数据结束中断
<i>SDIO_INT_STBITE</i>	起始位错误中断
<i>SDIO_INT_DTBLE ND</i>	数据块已发送/已接收中断
<i>SDIO_INT_CMDRU N</i>	正在传输命令中断
<i>SDIO_INT_TXRUN</i>	正在传输数据中断
<i>SDIO_INT_RXRUN</i>	正在接收数据中断
<i>SDIO_INT_TFH</i>	发送FIFO半满中断
<i>SDIO_INT_RFH</i>	接收FIFO半满中断
<i>SDIO_INT_TFF</i>	发送FIFO满中断
<i>SDIO_INT_RFF</i>	接收FIFO满中断
<i>SDIO_INT_TFE</i>	发送FIFO空中断
<i>SDIO_INT_RFE</i>	接收FIFO空中断
<i>SDIO_INT_TXDTVA L</i>	发送FIFO中的数据有效中断
<i>SDIO_INT_RXDTV AL</i>	接收FIFO中的数据有效中断
<i>SDIO_INT_SDIOIN T</i>	SD I/O中断已接收中断
<i>SDIO_INT_ATAEN D</i>	CE-ATA命令完成信号已接收中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the SDIO corresponding interrupts */
```

```
sdio_interrupt_enable(SDIO_INT_CCRERR | SDIO_INT_DTTMOUT | SDIO_INT_RXORE  
| SDIO_INT_DTEND | SDIO_INT_STBITE);
```

## 函数 sdio\_interrupt\_disable

函数sdio\_interrupt\_disable描述见下表：

**表 3-815. 函数 sdio\_interrupt\_disable**

函数名称	sdio_interrupt_disable
函数原形	void sdio_interrupt_disable(uint32_t int_flag);
功能描述	禁能SDIO中断
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	SDIO中断标志位状态
SDIO_INT_CCRCE RR	命令响应CRC错误中断
SDIO_INT_DTCRC ERR	数据CRC错误中断
SDIO_INT_CMDTM OUT	命令响应超时中断
SDIO_INT_DTTMO UT	数据超时中断
SDIO_INT_TXURE	发送FIFO下溢错误中断
SDIO_INT_RXORE	接收FIFO上溢错误中断
SDIO_INT_CMDRE CV	命令响应已接收中断
SDIO_INT_CMDSE ND	命令已发送中断
SDIO_INT_DTEND	数据结束中断
SDIO_INT_STBITE	起始位错误中断
SDIO_INT_DTBKE ND	数据块已发送/已接收中断
SDIO_INT_CMDRU N	正在传输命令中断
SDIO_INT_TXRUN	正在传输数据中断
SDIO_INT_RXRUN	正在接收数据中断
SDIO_INT_TFH	发送FIFO半满中断
SDIO_INT_RFH	接收FIFO半满中断
SDIO_INT_TFF	发送FIFO满中断
SDIO_INT_RFF	接收FIFO满中断
SDIO_INT_TFE	发送FIFO空中断
SDIO_INT_RFE	接收FIFO空中断
SDIO_INT_TXDTVA L	发送FIFO中的数据有效中断
SDIO_INT_RXDTV	接收FIFO中的数据有效中断

AL	
SDIO_INT_SDIOINT	SD I/O中断已接收中断
SDIO_INT_ATAEND	CE-ATA命令完成信号已接收中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the SDIO interrupt */
```

```
sdio_interrupt_disable(SDIO_INT_DTCRCERR);
```

### 函数 sdio\_interrupt\_flag\_get

函数sdio\_interrupt\_flag\_get描述见下表:

表 3-816. 函数 sdio\_interrupt\_flag\_get

函数名称	sdio_interrupt_flag_get
函数原形	FlagStatus sdio_interrupt_flag_get(uint32_t int_flag);
功能描述	获取SDIO的中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	SDIO中断标志位状态
SDIO_INT_FLAG_CRCERR	命令响应CRC错误中断
SDIO_INT_FLAG_DTCRCERR	数据CRC错误中断
SDIO_INT_FLAG_CMDTMOUT	命令响应超时中断
SDIO_INT_FLAG_DTTMOUT	数据超时中断
SDIO_INT_FLAG_TXURE	发送FIFO下溢错误中断
SDIO_INT_FLAG_RXORE	接收FIFO上溢错误中断
SDIO_INT_FLAG_CMDRECV	命令响应已接收中断
SDIO_INT_FLAG_CMDSEND	命令已发送中断
SDIO_INT_FLAG_DATAEND	数据结束中断

<i>TEND</i>	
<i>SDIO_INT_FLAG_S</i> <i>TBITE</i>	起始位错误中断
<i>SDIO_INT_FLAG_D</i> <i>TBLKEND</i>	数据块已发送/已接收中断
<i>SDIO_INT_FLAG_C</i> <i>MDRUN</i>	正在传输命令中断
<i>SDIO_INT_FLAG_T</i> <i>XRUN</i>	正在传输数据中断
<i>SDIO_INT_FLAG_R</i> <i>XRUN</i>	正在接收数据中断
<i>SDIO_INT_FLAG_T</i> <i>FH</i>	发送FIFO半满中断
<i>SDIO_INT_FLAG_R</i> <i>FH</i>	接收FIFO半满中断
<i>SDIO_INT_FLAG_T</i> <i>FF</i>	发送FIFO满中断
<i>SDIO_INT_FLAG_R</i> <i>FF</i>	接收FIFO满中断
<i>SDIO_INT_FLAG_T</i> <i>FE</i>	发送FIFO空中断
<i>SDIO_INT_FLAG_R</i> <i>FE</i>	接收FIFO空中断
<i>SDIO_INT_FLAG_T</i> <i>XDTVAL</i>	发送FIFO中的数据有效中断
<i>SDIO_INT_FLAG_R</i> <i>XDTVAL</i>	接收FIFO中的数据有效中断
<i>SDIO_INT_FLAG_S</i> <i>DIOINT</i>	SD I/O中断已接收中断
<i>SDIO_INT_FLAG_A</i> <i>TAEND</i>	CE-ATA命令完成信号已接收中断
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或 RESET

例如：

```
/* get the interrupt flags state of SDIO */
```

```
FlagStatus flag_value;
```

```
flag_value = sdio_interrupt_flag_get(SDIO_INT_FLAG_DTEND);
```

函数 `sdio_interrupt_flag_clear`

函数 `sdio_interrupt_flag_clear` 描述见下表：

表 3-817. 函数 `sdio_interrupt_flag_clear`

函数名称	<code>sdio_interrupt_flag_clear</code>
函数原形	<code>void sdio_interrupt_flag_clear(uint32_t int_flag);</code>
功能描述	清除SDIO的中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>int_flag</b>	SDIO中断标志位状态
<code>SDIO_INT_FLAG_C_CRCERR</code>	命令响应CRC错误中断
<code>SDIO_INT_FLAG_D_TCRCERR</code>	数据CRC错误中断
<code>SDIO_INT_FLAG_C_MDTMOUT</code>	命令响应超时中断
<code>SDIO_INT_FLAG_D_TTMOUT</code>	数据超时中断
<code>SDIO_INT_FLAG_TXURE</code>	发送FIFO下溢错误中断
<code>SDIO_INT_FLAG_RXORE</code>	接收FIFO上溢错误中断
<code>SDIO_INT_FLAG_C_MDRECV</code>	命令响应已接收中断
<code>SDIO_INT_FLAG_C_MDSEND</code>	命令已发送中断
<code>SDIO_INT_FLAG_D_TEND</code>	数据结束中断
<code>SDIO_INT_FLAG_S_TBITE</code>	起始位错误中断
<code>SDIO_INT_FLAG_D_TBLKEND</code>	数据块已发送/已接收中断
<code>SDIO_INT_FLAG_S_DIOINT</code>	SD I/O中断已接收中断
<code>SDIO_INT_FLAG_A_TAEND</code>	CE-ATA命令完成信号已接收中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the interrupt pending flags of SDIO */
```

```
sdio_interrupt_flag_clear(SDIO_INT_FLAG_DTEND);
```

### 函数 **sdio\_readwait\_enable**

函数sdio\_readwait\_enable描述见下表：

**表 3-818. 函数 sdio\_readwait\_enable**

函数名称	sdio_readwait_enable
函数原形	void sdio_readwait_enable(void);
功能描述	使能读等待模式（仅限SD I/O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 the read wait mode(SD I/O only) */
```

```
sdio_readwait_enable();
```

### 函数 **sdio\_readwait\_disable**

函数sdio\_readwait\_disable描述见下表：

**表 3-819. 函数 sdio\_readwait\_disable**

函数名称	sdio_readwait_disable
函数原形	void sdio_readwait_disable(void);
功能描述	禁能读等待模式（仅限SD I/O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 the read wait mode(SD I/O only) */
```

```
sdio_readwait_disable();
```

### 函数 `sdio_stop_readwait_enable`

函数`sdio_stop_readwait_enable`描述见下表：

表 3-820. 函数 `sdio_stop_readwait_enable`

函数名称	sdio_stop_readwait_enable
函数原形	void sdio_stop_readwait_enable(void);
功能描述	使能停止读等待过程的功能（仅限SD I/O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 the function that stop the read wait process(SD I/O only) */
```

```
sdio_stop_readwait_enable();
```

### 函数 `sdio_stop_readwait_disable`

函数`sdio_stop_readwait_disable`描述见下表：

表 3-821. 函数 `sdio_stop_readwait_disable`

函数名称	sdio_stop_readwait_disable
函数原形	void sdio_stop_readwait_disable(void);
功能描述	禁能停止读等待过程的功能（仅限SD I/O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 the function that stop the read wait process(SD I/O only) */
```

```
sdio_stop_readwait_disable();
```

## 函数 sdio\_readwait\_type\_set

函数sdio\_readwait\_type\_set描述见下表：

表 3-822. 函数 sdio\_readwait\_type\_set

函数名称	sdio_readwait_type_set
函数原形	void sdio_readwait_type_set(uint32_t readwait_type);
功能描述	设置读等待类型（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
readwait_type	SD I/O 读等待模式
SDIO_READWAITTYPE_CLK	通过停止SDIO_CLK控制读等待
SDIO_READWAITTYPE_DAT2	使用SDIO_DAT[2] 控制读等待
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the read wait type(SD I/O only) */
sdio_readwait_type_set(uint32_t readwait_type);
```

## 函数 sdio\_operation\_enable

函数sdio\_operation\_enable描述见下表：

表 3-823. 函数 sdio\_operation\_enable

函数名称	sdio_operation_enable
函数原形	void sdio_operation_enable(void);
功能描述	使能SD I/O模式特定操作（仅限SD I/O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 the SD I/O mode specific operation(SD I/O only) */
```

```
sdio_operation_enable();
```

### 函数 `sdio_operation_disable`

函数`sdio_operation_disable`描述见下表：

表 3-824. 函数 `sdio_operation_disable`

函数名称	sdio_operation_disable
函数原形	void sdio_operation_disable(void);
功能描述	禁能SD I/O模式特定操作（仅限SD I/O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 the SD I/O mode specific operation(SD I/O only) */
```

```
void sdio_operation_disable();
```

### 函数 `sdio_suspend_enable`

函数`sdio_suspend_enable`描述见下表：

表 3-825. 函数 `sdio_suspend_enable`

函数名称	sdio_suspend_enable
函数原形	void sdio_suspend_enable(void);
功能描述	使能SD I/O暂停模式（仅限SD I/O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 the SD I/O suspend operation(SD I/O only) */
```

```
sdio_suspend_enable();
```

## 函数 sdio\_suspend\_disable

函数sdio\_suspend\_disable描述见下表：

**表 3-826. 函数 sdio\_suspend\_disable**

函数名称	sdio_suspend_disable
函数原形	void sdio_suspend_disable(void);
功能描述	禁能SD I/O暂停模式（仅限SD I/O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 the SD I/O suspend operation(SD I/O only) */
```

```
sdio_suspend_disable();
```

## 函数 sdio\_ceata\_command\_enable

函数sdio\_ceata\_command\_enable描述见下表：

**表 3-827. 函数 sdio\_ceata\_command\_enable**

函数名称	sdio_ceata_command_enable
函数原形	void sdio_ceata_command_enable(void);
功能描述	使能CE-ATA命令(仅限CE-ATA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 the CE-ATA command(CE-ATA only) */
```

```
sdio_ceata_command_enable();
```

## 函数 sdio\_ceata\_command\_disable

函数sdio\_ceata\_command\_disable描述见下表:

**表 3-828. 函数 sdio\_ceata\_command\_disable**

函数名称	sdio_ceata_command_disable
函数原形	void sdio_ceata_command_disable(void);
功能描述	禁能CE-ATA命令(仅限CE-ATA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 the CE-ATA command(CE-ATA only) */
sdio_ceata_command_disable();
```

## 函数 sdio\_ceata\_interrupt\_enable

函数sdio\_ceata\_interrupt\_enable描述见下表:

**表 3-829. 函数 sdio\_ceata\_interrupt\_enable**

函数名称	sdio_ceata_interrupt_enable
函数原形	void sdio_ceata_interrupt_enable(void);
功能描述	使能CE-ATA中断(仅限CE-ATA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 the CE-ATA interrupt(CE-ATA only) */
sdio_ceata_interrupt_enable();
```

**函数 sdio\_ceata\_interrupt\_disable**

函数sdio\_ceata\_interrupt\_disable描述见下表：

**表 3-830. 函数 sdio\_ceata\_interrupt\_disable**

函数名称	sdio_ceata_interrupt_disable
函数原形	void sdio_ceata_interrupt_disable(void);
功能描述	禁能CE-ATA中断(仅限CE-ATA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 the CE-ATA interrupt(CE-ATA only) */
```

```
sdio_ceata_interrupt_disable();
```

**函数 sdio\_ceata\_command\_completion\_enable**

函数sdio\_ceata\_command\_completion\_enable描述见下表：

**表 3-831. 函数 sdio\_ceata\_command\_completion\_enable**

函数名称	sdio_ceata_command_completion_enable
函数原形	void sdio_ceata_command_completion_enable(void);
功能描述	使能CE-ATA命令完成信号(仅限CE-ATA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 the CE-ATA command completion signal(CE-ATA only) */
```

```
sdio_ceata_command_completion_enable();
```

**函数 sdio\_ceata\_command\_completion\_disable**

函数sdio\_ceata\_command\_completion\_disable描述见下表：

**表 3-832. 函数 sdio\_ceata\_command\_completion\_disable**

函数名称	sdio_ceata_command_completion_disable
函数原形	void sdio_ceata_command_completion_disable(void);
功能描述	禁能CE-ATA命令完成信号(仅限CE-ATA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 the CE-ATA command completion signal(CE-ATA only) */
sdio_ceata_command_completion_disable();
```

**3.24. 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[3.24.1](#)描述了SPI/I2S的寄存器列表，章节[3.24.2](#)对SPI/I2S库函数进行说明。

**3.24.1. 外设寄存器说明**

SPI/I2S寄存器列表如下表所示：

**表 3-833. 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时钟分频寄存器
SPI_QCTL	SPI四线模式控制寄存器

## 3.24.2. 外设库函数说明

SPI/I2S库函数列表如下表所示：

表 3-834. SPI/I2S 库函数

库函数名称	库函数描述
spi_i2s_deinit	复位外设SPIx/I2Sx
spi_struct_para_init	将SPI结构体中所有参数初始化为默认值
spi_init	初始化外设SPIx
spi_enable	使能外设SPIx
spi_disable	失能外设SPIx
i2s_init	初始化外设I2Sx
i2s_psc_config	配置I2Sx预分频器
i2s_enable	使能外设I2Sx
i2s_disable	失能外设I2Sx
spi_nss_output_enable	使能外设SPIx NSS输出
spi_nss_output_disable	失能外设SPIx NSS输出
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SPIx的DMA功能
spi_dma_disable	失能外设SPIx的DMA功能
spi_i2s_data_frame_format_config	配置外设SPIx/I2Sx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SPIx的数据传输方向
spi_i2s_format_error_clear	清除帧错误标志状态
spi_crc_polynomial_set	设置外设SPIx的CRC多项式值
spi_crc_polynomial_get	获取外设SPIx的CRC多项式值
spi_crc_on	打开外设SPIx的CRC功能
spi_crc_off	关闭外设SPIx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SPIx获取CRC值
spi_crc_error_clear	清除SPIx 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
i2s_full_duplex_mode_config	配置I2S全双工模式
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

库函数名称	库函数描述
spi_quad_io23_output_disable	禁能SPI_IO2和SPI_IO3输出
spi_i2s_interrupt_enable	使能外设SPIx/I2Sx中断
spi_i2s_interrupt_disable	失能外设SPIx/I2Sx中断
spi_i2s_interrupt_flag_get	获取外设SPIx/I2Sx中断状态
spi_i2s_flag_get	获取外设SPIx/I2Sx标志状态

## 结构体 spi\_parameter\_struct

表 3-835. 结构体 spi\_parameter\_struct

成员名称	功能描述
device_mode	配置SPI为主机或从机模式 (SPI_MASTER, SPI_SLAVE)
trans_mode	传输模式 (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	数据帧格式配置 (SPI_FRAME_SIZE_16BIT, SPI_FRAME_SIZE_8BIT)
nss	配置NSS由软件或硬件控制 (SPI_NSS_SOFT, SPI_NSS_HARD)
endian	大端或小端模式配置 (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)
clock_polarity_phase	相位和极性配置 (SPI_CK_PL_LOW_PH_1EDGE, SPI_CK_PL_HIGH_PH_1EDGE, SPI_CK_PL_LOW_PH_2EDGE, SPI_CK_PL_HIGH_PH_2EDGE)
prescale	预分频器配置 (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 3-836. 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SPIx/I2Sx
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=0,1,2
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 3-837. 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	将SPI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}	
*spi_struct	一个已经定义的spi_parameter_struct结构体变量地址
输出参数{out}	
-	-
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 3-838. 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SPIx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
spi_struct	初始化结构体, 结构体成员参考 <a href="#">表3-835. 结构体spi_parameter_struct</a>

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize SPI0 */

spi_parameter_struct spi_init_struct;

spi_init_struct.trans_mode      = SPI_TRANSMODE_BDTRANSMIT;
spi_init_struct.device_mode     = SPI_MASTER;
spi_init_struct.frame_size      = SPI_FRAME_SIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_CK_PL_HIGH_PH_2EDGE;
spi_init_struct.nss             = SPI_NSS_SOFT;
spi_init_struct.prescale        = SPI_PSC_8;
spi_init_struct.endian          = SPI_ENDIAN_MSB;

spi_init(SPI0, &spi_init_struct);
```

### 函数 spi\_enable

函数spi\_enable描述见下表：

表 3-839. 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SPIx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
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 3-840. 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	禁能外设SPIx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
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 3-841. 函数 **i2s\_init**

函数名称	i2s_init
函数原形	void i2s_init(uint32_t spi_periph, uint32_t i2s_mode, uint32_t i2s_standard, uint32_t i2s_ckpl);
功能描述	初始化外设I2Sx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设I2Sx
SPIx	x=1,2
输入参数{in}	
i2s_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

<i>I2S_MODE_MASTE RRX</i>	I2S主机接收模式
输入参数{in}	
<b>i2s_standard</b>	I2S标准选择
<i>I2S_STD_PHILLIPS</i>	I2S飞利浦标准
<i>I2S_STD_MSB</i>	I2S MSB对齐标准
<i>I2S_STD_LSB</i>	I2S LSB对齐标准
<i>I2S_STD_PCMSHO RT</i>	I2S PCM短帧标准
<i>I2S_STD_PCMLON G</i>	I2S PCM长帧标准
输入参数{in}	
<b>i2s_ckpl</b>	I2S空闲状态时钟极性
<i>I2S_CKPL_LOW</i>	I2S_CK空闲状态为低电平
<i>I2S_CKPL_HIGH</i>	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 3-842. 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 i2s_audiosample, uint32_t i2s_frameformat, uint32_t i2s_mckout);
功能描述	配置I2Sx预分频器
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
<b>spi_periph</b>	外设I2Sx
<i>SPIx</i>	x=1,2
输入参数{in}	
<b>i2s_audiosample</b>	I2S音频采样频率
<i>I2S_AUDIOSAMPL E_8K</i>	音频采样频率为8KHz
<i>I2S_AUDIOSAMPL</i>	音频采样频率为11KHz

<i>E_11K</i>	
<i>I2S_AUDIOSAMPL E_16K</i>	音频采样频率为16KHz
<i>I2S_AUDIOSAMPL E_22K</i>	音频采样频率为22KHz
<i>I2S_AUDIOSAMPL E_32K</i>	音频采样频率为32KHz
<i>I2S_AUDIOSAMPL E_44K</i>	音频采样频率为44KHz
<i>I2S_AUDIOSAMPL E_48K</i>	音频采样频率为48KHz
<i>I2S_AUDIOSAMPL E_96K</i>	音频采样频率为96KHz
<i>I2S_AUDIOSAMPL E_192K</i>	音频采样频率为192KHz
<b>输入参数{in}</b>	
<b>i2s_frameformat</b>	I2S数据长度和通道长度
<i>I2S_FRAMEFORMA T_DT16B_CH16B</i>	I2S数据长度为16位，通道长度为16位
<i>I2S_FRAMEFORMA T_DT16B_CH32B</i>	I2S数据长度为16位，通道长度为32位
<i>I2S_FRAMEFORMA T_DT24B_CH32B</i>	I2S数据长度为24位，通道长度为32位
<i>I2S_FRAMEFORMA T_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>i2s_mckout</b>	2S_MCK输出使能
<i>I2S_MCKOUT_ENA BLE</i>	I2S_MCK输出使能
<i>I2S_MCKOUT_DIS ABLE</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 3-843. 函数 i2s\_enable**

函数名称	i2s_enable
函数原形	void i2s_enable(uint32_t spi_periph);
功能描述	使能外设I2Sx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI/I2Sx
SPIx	x=1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2S1 */
i2s_enable(SPI1);
```

**函数 i2s\_disable**

函数i2s\_disable描述见下表：

**表 3-844. 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	禁能外设I2Sx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI/I2Sx
SPIx	x=1,2
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 3-845. 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SPIx NSS输出
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
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 3-846. 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	禁能外设SPIx NSS输出
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
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 3-847. 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=0,1,2
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 3-848. 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=0,1,2
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 3-849. 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SPIx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=0,1,2
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
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 3-850. 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	禁能外设SPIx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=0,1,2
输入参数{in}	
dma	SPI DMA模式
SPI_DMA_TRANSMIT	SPI发送缓冲区DMA使能

<i>IT</i>	
<i>SPI_DMA_RECEIVE</i>	SPI接收缓冲区DMA使能
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 3-851. 函数 spi\_i2s\_data\_frame\_format\_config

函数名称	spi_i2s_data_frame_format_config
函数原形	void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
功能描述	配置外设SPIx/I2Sx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=0,1,2
输入参数{in}	
frame_format	SPI帧大小
SPI_FRAME_SIZE_16BIT	SPI 16位数据帧格式
SPI_FRAME_SIZE_8BIT	SPI 8位数据帧格式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SPI1/I2S1 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 3-852. 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=0,1,2
输入参数{in}	
data	16位数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 transmit data */
```

```
uint16_t spi0_send_array[] = {0x5050,0xA0A0};
```

```
spi_i2s_data_transmit(SPI0, spi0_send_array[0]);
```

**函数 spi\_i2s\_data\_receive**

函数spi\_i2s\_data\_receive描述见下表：

**表 3-853. 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=0,1,2
输出参数{out}	
-	-
返回值	
-	16位数据

例如：

```
/* SPI0 receive data */
```

```
uint16_t spi0_receive_data;
```

```
spi0_receive_data = spi_i2s_data_receive(SPI0);
```

### 函数 **spi\_bidirectional\_transfer\_config**

函数 `spi_bidirectional_transfer_config` 描述见下表：

**表 3-854. 函数 `spi_bidirectional_transfer_config`**

函数名称	<code>spi_bidirectional_transfer_config</code>
函数原形	<code>void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);</code>
功能描述	配置外设SPIx的数据传输方向
先决条件	-
被调用函数	-
输入参数{in}	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=0,1,2
输入参数{in}	
<b>transfer_direction</b>	SPI双向传输输出使能
<code>SPI_BIDIRECTIONAL_TRANSMIT</code>	SPI工作在只发送模式
<code>SPI_BIDIRECTIONAL_RECEIVE</code>	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 3-855. 函数 `spi_i2s_format_error_clear`**

函数名称	<code>spi_i2s_format_error_clear</code>
函数原形	<code>void spi_i2s_format_error_clear(uint32_t spi_periph, uint32_t flag);</code>
功能描述	清除帧错误标志状态
先决条件	-
被调用函数	-

输入参数{in}	
<b>spi_periph</b>	外设SPIx
<i>SPIx</i>	x=0,1,2
输入参数{in}	
<b>flag</b>	SPI/I2S帧错误状态
<i>SPI_FLAG_FERR</i>	SPI TI模式帧错误
<i>I2S_FLAG_FERR</i>	I2S帧错误
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear SPI0 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 3-856. 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SPIx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=0,1,2
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 */
```

```
uint16_t CRC_VALUE = 0x5050;
```

```
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 3-857. 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SPIx的CRC多项式值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
uint16_t	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 3-858. 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SPIx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=0,1,2
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 3-859. 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SPIx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=0,1,2
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 3-860. 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SPIx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=0,1,2
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 3-861. 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SPIx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=0,1,2
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 3-862. 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SPIx 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=0,1,2
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 3-863. 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=0,1,2
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 3-864. 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	禁能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=0,1,2

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 3-865. 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=0,1,2
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 3-866. 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	禁能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=0,1,2

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 NSS pulse mode */
```

```
spi_nssp_mode_disable(SPI0);
```

### 函数 i2s\_full\_duplex\_mode\_config

函数i2s\_full\_duplex\_mode\_config描述见下表：

表 3-867. 函数 i2s\_full\_duplex\_mode\_config

函数名称	i2s_full_duplex_mode_config
函数原形	void i2s_full_duplex_mode_config(uint32_t i2s_add_periph,uint32_t i2s_mode,uint32_t i2s_standard,uint32_t i2s_ckpl,uint32_t i2s_frameformat);
功能描述	配置I2S全双工模式
先决条件	-
被调用函数	-
输入参数{in}	
i2s_add_periph	外设I2Sx_ADD
I2Sx_ADD	x=1,2
输入参数{in}	
i2s_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}	
i2s_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_PCMLON G	I2S PCM长帧标准
输入参数{in}	

<b>i2s_ckpl</b>	I2S空闲状态时钟极性
<i>I2S_CKPL_LOW</i>	I2S_CK空闲状态为低电平
<i>I2S_CKPL_HIGH</i>	I2S_CK空闲状态为高电平
<b>输入参数{in}</b>	
<b>i2s_frameformat</b>	I2S数据长度和通道长度
<i>I2S_FRAMEFORMA T_DT16B_CH16B</i>	I2S数据长度为16位，通道长度为16位
<i>I2S_FRAMEFORMA T_DT16B_CH32B</i>	I2S数据长度为16位，通道长度为32位
<i>I2S_FRAMEFORMA T_DT24B_CH32B</i>	I2S数据长度为24位，通道长度为32位
<i>I2S_FRAMEFORMA T_DT32B_CH32B</i>	I2S数据长度为32位，通道长度为32位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure I2S1_ADD */
```

```
i2s_full_duplex_mode_config(I2S1_ADD,I2S_MODE_MASTERTX,I2S_STD_PHILLIPS,  
I2S_CKPL_HIGH, I2S_FRAMEFORMAT_DT16B_CH16B);
```

### 函数 spi\_quad\_enable

函数spi\_quad\_enable描述见下表：

表 3-868. 函数 spi\_quad\_enable

<b>函数名称</b>	spi_quad_enable
<b>函数原形</b>	void spi_quad_enable (uint32_t spi_periph);
<b>功能描述</b>	使能四线SPI模式
<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SPIx
<i>SPIx</i>	x=0
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-869. 函数 spi\_quad\_disable

函数名称	spi_quad_disable
函数原形	spi_quad_disable(uint32_t spi_periph);
功能描述	禁能四线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 3-870. 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 3-871. 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 3-872. 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 3-873. 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	禁能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 3-874. 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SPIx/I2Sx中断
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
interrupt	SPI/I2S中断
SPI_I2S_INT_TBE	发送缓冲区空中断使能
SPI_I2S_INT_RBNE	接收缓冲区非空中断使能
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 3-875. 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	禁能外设SPIx/I2Sx中断
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
interrupt	SPI/I2S中断
SPI_I2S_INT_TBE	发送缓冲区空中断使能
SPI_I2S_INT_RBNE	接收缓冲区非空中断使能
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 3-876. 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SPIx/I2Sx中断状态
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=0,1,2
输入参数{in}	
<b>interrupt</b>	SPI/I2S中断状态
<i>SPI_I2S_INT_FLAG_TBE</i>	发送缓冲区空中断
<i>SPI_I2S_INT_FLAG_RBNE</i>	接收缓冲区非空中断
<i>SPI_I2S_INT_FLAG_RXOVERR</i>	接收过载错误中断
<i>SPI_INT_FLAG_COFERR</i>	配置错误中断
<i>SPI_INT_FLAG_CRCERR</i>	CRC错误中断
<i>I2S_INT_FLAG_TXURERR</i>	发送欠载错误中断
<i>SPI_I2S_INT_FLAG_FERR</i>	帧错误中断
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	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 3-877. 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SPIx/I2Sx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
<b>SPIx</b>	x=0,1,2
<b>输入参数{in}</b>	
<b>flag</b>	SPI/I2S标志状态
<b>SPI_FLAG_TBE</b>	发送缓冲区空标志
<b>SPI_FLAG_RBNE</b>	接收缓冲区非空标志
<b>SPI_FLAG_TRANS</b>	通信进行中标志
<b>SPI_I2S_INT_FLAG_RXOERR</b>	接收过载错误标志
<b>SPI_FLAG_CONFERR</b>	配置错误标志
<b>SPI_FLAG_CRCERR</b>	CRC错误标志
<b>SPI_FLAG_FERR</b>	帧错误标志
<b>I2S_FLAG_TBE</b>	发送缓冲区空标志
<b>I2S_FLAG_RBNE</b>	接收缓冲区非空标志
<b>I2S_FLAG_TRANS</b>	通信进行中标志
<b>I2S_FLAG_RXOERR</b>	接收过载错误标志
<b>I2S_FLAG_TXURERR</b>	发送欠载错误标志
<b>I2S_FLAG_CH</b>	通道标志
<b>I2S_FLAG_FERR</b>	帧错误标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	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++]);

```

## 3.25. SQPI

SQPI接口是一个用于串行、双线、四线接口存储设备的控制器。章节[3.25.1](#)描述了SQPI的寄存器列表，章节[3.25.2](#)对SQPI库函数进行说明。

### 3.25.1. 外设寄存器说明

SQPI寄存器列表如下表所示：

表 3-878. SQPI 寄存器

寄存器名称	寄存器描述
SQPI_INIT	SQPI初始化寄存器
SQPI_RCMD	SQPI读命令寄存器
SQPI_WCMD	SQPI写命令寄存器
SQPI_IDL	SQPI ID低位寄存器
SQPI_IDH	SQPI ID 高位寄存器

### 3.25.2. 外设库函数说明

SQPI库函数列表如下表所示：

表 3-879. SQPI 库函数

库函数名称	库函数描述
sqpi_deinit	复位外设SQPI
sqpi_struct_para_init	将SQPI结构体中所有参数初始化为默认值
sqpi_init	初始化外设SQPI
sqpi_read_id_command	SQPI发送读ID命令
sqpi_special_command	SQPI发送特殊命令
sqpi_read_command_config	SQPI读命令配置
sqpi_write_command_config	SQPI写命令配置
sqpi_low_id_receive	SQPI获取ID低位数据
sqpi_high_id_receive	SQPI获取ID高位数据

### 结构体 sqpi\_parameter\_struct

表 3-880. 结构体 sqpi\_parameter\_struct

成员名称	功能描述
polarity	SQPI采样极性 (SQPI_SAMPLE_POLARITY_RISING, SQPI_SAMPLE_POLARITY_FALLING)
id_length	外部存储器ID长度 (QSPI_ID_LENGTH_n_BITS (n=8,16,32,64))
addr_bit	地址阶段的地址位数 (0x00 - 0x1F)
clk_div	SQPI时钟分频 (0x01 - 0x3F)
cmd_bit	命令阶段的命令位数 (QSPI_CMDBIT_n_BITS (n=4,8,16))

### 函数 sqpi\_deinit

函数sqpi\_deinit描述见下表：

表 3-881. 函数 sqpi\_deinit

函数名称	sqpi_deinit
------	-------------

函数原形	void sqpi_deinit(void);
功能描述	复位外设SQPI
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 SQPI */
sqpi_deinit();
```

### 函数 sqpi\_struct\_para\_init

函数sqpi\_struct\_para\_init描述见下表：

表 3-882. 函数 sqpi\_struct\_para\_init

函数名称	sqpi_struct_para_init
函数原形	void sqpi_struct_para_init(sqpi_parameter_struct* sqpi_struct);
功能描述	将SQPI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}	
*sqpi_struct	一个已经定义的sqpi_parameter_struct结构体变量地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the parameters of SQPI */
sqpi_parameter_struct sqpi_init_struct;
sqpi_struct_para_init(&sqpi_init_struct);
```

### 函数 sqpi\_init

函数sqpi\_init描述见下表：

表 3-883. 函数 sqpi\_init

函数名称	sqpi_init
------	-----------

函数原形	void sqpi_init(sqpi_parameter_struct* sqpi_struct);
功能描述	初始化外设SQPI
先决条件	-
被调用函数	-
输入参数{in}	
sqpi_struct	初始化结构体，结构体成员参考 <a href="#">表3-880. 结构体sqpi_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize SQPI */

sqpi_parameter_struct sqpi_init_struct;

sqpi_struct->polarity = SQPI_SAMPLE_POLARITY_RISING;

sqpi_struct->id_length = QSPI_ID_LENGTH_32_BITS;

sqpi_struct->addr_bit = 24U;

sqpi_struct->clk_div = 2U;

sqpi_struct->cmd_bit = QSPI_CMDBIT_8_BITS;

sqpi_init(&sqpi_init_struct);

```

### 函数 sqpi\_read\_id\_command

函数sqpi\_read\_id\_command描述见下表：

**表 3-884. 函数 sqpi\_read\_id\_command**

函数名称	sqpi_read_id_command
函数原形	void sqpi_read_id_command (void);
功能描述	发送读ID命令
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* send SQPI read ID command */

```

```
sqpi_read_id_command ();
```

### 函数 sqpi\_special\_command

函数sqpi\_special\_command描述见下表：

表 3-885. 函数 sqpi\_special\_command

函数名称	sqpi_special_command
函数原形	void sqpi_special_command(void);
功能描述	SQPI发送特殊命令
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* send SQPI special command */
```

```
sqpi_special_command ();
```

### 函数 sqpi\_read\_command\_config

函数sqpi\_read\_command\_config描述见下表：

表 3-886. 函数 sqpi\_read\_command\_config

函数名称	sqpi_read_command_config
函数原形	void sqpi_read_command_config(uint32_t rmode, uint32_t rwaitcycle, uint32_t rcmd);
功能描述	配置SQPI读命令
先决条件	-
被调用函数	-
输入参数{in}	
rmode	SQPI读命令模式
QSPI_MODE_SSQ	SSQ模式
QSPI_MODE_SSS	SSS模式
QSPI_MODE_SQQ	SQQ模式
QSPI_MODE_QQQ	QQQ模式
QSPI_MODE_SSD	SSD模式
QSPI_MODE_SDD	SDD模式
输入参数{in}	
rwaitcycle	SQPI read wait cycle (0x00 – 0x1F)

输入参数{in}	
<b>rcmd</b>	SQPI read command(0x00 – 0xFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* send SQPI special command */
```

```
sqpi_special_command ();
```

### 函数 sqpi\_write\_command\_config

函数sqpi\_write\_command\_config描述见下表:

表 3-887. 函数 sqpi\_write\_command\_config

函数名称	sqpi_write_command_config
函数原形	void sqpi_write_command_config(uint32_t wmode, uint32_t wwaitcycle, uint32_t wcmd);
功能描述	配置SQPI写命令
先决条件	-
被调用函数	-
输入参数{in}	
<b>wmode</b>	SQPI写命令模式
QSPI_MODE_SSQ	SSQ模式
QSPI_MODE_SSS	SSS模式
QSPI_MODE_SQQ	SQQ模式
QSPI_MODE_QQQ	QQQ模式
QSPI_MODE_SSD	SSD模式
QSPI_MODE_SDD	SDD模式
输入参数{in}	
<b>wwaitcycle</b>	SQPI write wait cycle (0x00 – 0x1F)
输入参数{in}	
<b>wcmd</b>	SQPI write command(0x00 – 0xFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure SQPI write command */
```

```
sqpi_write_command_config(QSPI_MODE_SSS,0x00,0x9f);
```

## 函数 sqpi\_low\_id\_receive

函数sqpi\_low\_id\_receive描述见下表：

**表 3-888. 函数 sqpi\_low\_id\_receive**

函数名称	sqpi_low_id_receive
函数原形	uint32_t sqpi_low_id_receive(void);
功能描述	获取低位ID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	32位ID值(0-0xFFFF)

例如：

```
/* get SQPI low ID value */
uint32_t val;
val = sqpi_low_id_receive ();
```

## 函数 sqpi\_high\_id\_receive

函数sqpi\_high\_id\_receive描述见下表：

**表 3-889. 函数 sqpi\_high\_id\_receive**

函数名称	sqpi_high_id_receive
函数原形	uint32_t sqpi_high_id_receive(void);
功能描述	获取高位ID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	32位ID值(0-0xFFFF)

例如：

```
/* get SQPI high ID value */
uint32_t val;
val = sqpi_high_id_receive ();
```

## 3.26. 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，分为五种类型：高级定时器(TIMERx, x=0, 7)，通用定时器L0(TIMERx, x=1, 2, 3, 4)，通用定时器L1(TIMERx, x=8, 11)，通用定时器L2(TIMERx, x=9, 10, 12, 13)，通用定时器L3(TIMERx, x=14)，通用定时器L4(TIMERx, x=15, 16)，基本定时器(TIMERx, x=5, 6)，不同类型的定时器具体功能有所差别。章节[3.26.1](#)描述了TIMER的寄存器列表，章节[3.26.2](#)对TIMER库函数进行说明。

### 3.26.1. 外设寄存器说明

TIMER寄存器列表如下表所示：

**表 3-890. 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
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_DMACFG	DMA配置寄存器
TIMER_DMATB	DMA发送缓冲区寄存器
TIMER_IRMP	通道输入重映射寄存器
TIMER_CFG	配置寄存器

### 3.26.2. 外设库函数说明

TIMER库函数列表如下表所示：

表 3-891. TIMER 库函数

库函数名称	库函数描述
timer_deinit	复位外设TIMERx
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TIMERx
timer_enable	使能外设TIMERx
timer_disable	禁能外设TIMERx
timer_auto_reload_shadow_enable	TIMERx自动重载影子使能
timer_auto_reload_shadow_disable	TIMERx自动重载影子禁能
timer_update_event_enable	TIMERx更新使能
timer_update_event_disable	TIMERx更新禁能
timer_counter_alignment	设置外设TIMERx的对齐模式
timer_counter_up_direction	设置外设TIMERx向上计数
timer_counter_down_direction	设置外设TIMERx向下计数
timer_prescaler_config	配置外设TIMERx预分频器
timer_repetition_value_config	配置外设TIMERx的重复计数器
timer_autoreload_value_config	配置外设TIMERx的自动重载寄存器
timer_counter_value_config	配置外设TIMERx的计数器值
timer_counter_read	读取外设TIMERx的计数器值
timer_prescaler_read	读取外设TIMERx的预分频器值
timer_single_pulse_mode_config	配置外设TIMERx的单脉冲模式
timer_update_source_config	配置外设TIMERx的更新源
timer_dma_enable	外设TIMERx的DMA使能
timer_dma_disable	外设TIMERx的DMA禁能
timer_channel_dma_request_source_select	外设TIMERx的通道DMA请求源选择
timer_dma_transfer_config	配置外设TIMERx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TIMERx的中止功能
timer_break_disable	禁能TIMERx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_control_shadow_config	通道换相控制影子寄存器配置
timer_channel_control_shadow_update_config	通道换相控制影子寄存器更新控制
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TIMERx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TIMERx通道输出比较模式
timer_channel_output_pulse_value_config	配置外设TIMERx的通道输出比较值
timer_channel_output_shadow_config	配置TIMERx通道输出比较影子寄存器功能
timer_channel_output_fast_config	配置TIMERx通道输出比较快速功能
timer_channel_output_clear_config	配置TIMERx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TIMERx输入捕获参数
timer_channel_input_capture_prescaler_config	配置TIMERx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TIMERx捕获PWM输入参数
timer_hall_mode_config	配置TIMERx的HALL接口功能
timer_input_trigger_source_select	TIMERx的输入触发源选择
timer_master_output_trigger_source_select	选择TIMERx主模式输出触发
timer_slave_mode_select	TIMERx从模式配置
timer_master_slave_mode_config	TIMERx主从模式配置
timer_external_trigger_config	配置TIMERx外部触发输入
timer_quadrature_decoder_mode_config	TIMERx配置为编码器模式
timer_internal_clock_config	TIMERx配置为内部时钟模式
timer_internal_trigger_as_external_clock_config	配置TIMERx的内部触发为时钟源
timer_external_trigger_as_external_clock_config	配置TIMERx的外部触发作为时钟源
timer_external_clock_mode0_config	配置TIMERx外部时钟模式0，ETI作为时钟源
timer_external_clock_mode1_config	配置TIMERx外部时钟模式1
timer_external_clock_mode1_disable	TIMERx外部时钟模式1禁能
timer_channel_remap_config	配置TIMERx通道重映射功能
timer_write_chxval_register_config	配置TIMERx写CHxVAL选择位

库函数名称	库函数描述
timer_output_value_selection_config	配置TIMERx输出值选择位
timer_flag_get	获取外设TIMERx的状态标志
timer_flag_clear	清除外设TIMERx状态标志
timer_interrupt_enable	外设TIMERx中断使能
timer_interrupt_disable	外设TIMERx中断禁能
timer_interrupt_flag_get	获取外设TIMERx中断标志
timer_interrupt_flag_clear	清除外设TIMERx的中断标志

### 结构体 timer\_parameter\_struct

表 3-892. 结构体 timer\_parameter\_struct

成员名称	功能描述
prescaler	预分频值（0~65535）
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 3-893. 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 3-894. 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\_ic\_parameter\_struct

表 3-895. 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)
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)

## 函数 timer\_deinit

函数timer\_deinit描述见下表:

表 3-896. 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TIMERx
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..16)	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 3-897. 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 3-898. 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TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..16)	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 3-899. 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TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..16)	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 3-900. 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TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..16)	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 3-901. 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	TIMERx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..16)	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 3-902. 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	TIMERx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..16)	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 3-903. 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	TIMERx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..16)	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 3-904. 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	TIMERx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..16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER0 the update event */
timer_update_event_disable(TIMER0);
```

### 函数 timer\_counter\_alignment

函数timer\_counter\_alignment描述见下表：

表 3-905. 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TIMERx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..4,7)	TIMER外设选择
输入参数{in}	
aligned	对齐模式
TIMER_COUNTER_EDGE	无中央对齐计数模式(边沿对齐模式)，DIR位指定了计数方向
TIMER_COUNTER_CENTER_DOWN	中央对齐向下计数置1模式。计数器在中央计数模式计数，通道被配置在输出模式（TIMERx_CHCTL0寄存器中CHxMS=00），只有在向下计数时，通道的比较中断标志置1
TIMER_COUNTER_CENTER_UP	中央对齐向上计数置1模式。计数器在中央计数模式计数，通道被配置在输出模式（TIMERx_CHCTL0寄存器中CHxMS=00），只有在向上计数时，通道的比较中断标志置1
TIMER_COUNTER_CENTER_BOTH	中央对齐上下计数置1模式。计数器在中央计数模式计数，通道被配置在输出模式（TIMERx_CHCTL0寄存器中CHxMS=00），在向上和向下计数时，通道的比

	较中断标志都会置1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set TIMER0 counter center-aligned and counting up assert mode */
```

```
timer_counter_alignment(TIMER0, TIMER_COUNTER_CENTER_UP);
```

### 函数 timer\_counter\_up\_direction

函数timer\_counter\_up\_direction描述见下表：

表 3-906. 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TIMERx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..4,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set TIMER0 counter up direction */
```

```
timer_counter_up_direction(TIMER0);
```

### 函数 timer\_counter\_down\_direction

函数timer\_counter\_down\_direction描述见下表：

表 3-907. 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TIMERx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

<i>TIMERx(x=0..4,7)</i>	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set TIMER0 counter down direction */
timer_counter_down_direction(TIMER0);
```

### 函数 timer\_prescaler\_config

函数timer\_prescaler\_config描述见下表:

表 3-908. 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TIMERx预分频器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0..16)</i>	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
<i>TIMER_PSC_RELOAD_NOW</i>	预分频值立即加载
<i>TIMER_PSC_RELOAD_UPDATE</i>	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 3-909. 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TIMERx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=0,7,14..16)	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 3-910. 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, uint32_t autoreload);
功能描述	配置外设TIMERx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..16)	TIMER外设选择
输入参数{in}	
autoreload	计数器自动重载值（0-0xFFFFFFFF）
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 3-911. 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, uint32_t counter);
功能描述	配置外设TIMERx的计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..16)	TIMER外设选择
输入参数{in}	
counter	计数器值（0-0xFFFFFFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 counter register value */
timer_counter_value_config(TIMER0);
```

### 函数 timer\_counter\_read

函数timer\_counter\_read描述见下表：

表 3-912. 函数 timer\_counter\_read

函数名称	timer_counter_read
函数原型	uint32_t timer_counter_read(uint32_t timer_periph);
功能描述	读取外设TIMERx的计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..16)	TIMER外设选择
输出参数{out}	
-	-

返回值	
uint32_t	外设TIMERx的计数器值（0~0xFFFFFFFF）

例如：

```
/* read TIMERO counter value */

uint32_t i = 0;

i = timer_counter_read(TIMERO);
```

### 函数 timer\_prescaler\_read

函数timer\_prescaler\_read描述见下表：

表 3-913. 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TIMERx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..16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint16_t	外设TIMERx的预分频器值（0x0000~0xFFFF）

例如：

```
/* read TIMERO prescaler value */

uint16_t i = 0;

i = timer_prescaler_read(TIMERO);
```

### 函数 timer\_single\_pulse\_mode\_config

函数timer\_single\_pulse\_mode\_config描述见下表：

表 3-914. 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, uint8_t spmode);
功能描述	配置外设TIMERx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

<i>TIMERx</i> ( <i>x</i> =0..4,7,11,14..16)	TIMER外设选择
输入参数{in}	
<b>spmode</b>	脉冲模式
<i>TIMER_SP_MODE_SINGLE</i>	单脉冲模式计数
<i>TIMER_SP_MODE_REPETITIVE</i>	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 3-915. 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TIMERx的更新源
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..16)	TIMER外设选择
输入参数{in}	
<b>update</b>	更新源
<i>TIMER_UPDATE_SRC_GLOBAL</i>	下述任一事件产生更新中断或DMA请求: - UPG位被置1 - 计数器溢出/下溢 - 从模式控制器产生的更新
<i>TIMER_UPDATE_SRC_REGULAR</i>	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\_dma\_enable

函数timer\_dma\_enable描述见下表:

表 3-916. 函数 timer\_dma\_enable

函数名称	timer_dma_enable
函数原型	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
功能描述	外设TIMERx的DMA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}	
dma	DMA源
TIMER_DMA_UPD	更新DMA请求, TIMERx(x=0..16)
TIMER_DMA_CH0 D	通道0比较/捕获 DMA请求, TIMERx(x=0..4,7,8,11,14..16)
TIMER_DMA_CH1 D	通道1比较/捕获 DMA请求, TIMERx(x=0..4,7,8,11,14)
TIMER_DMA_CH2 D	通道2比较/捕获 DMA请求, TIMERx(x=0..4,7)
TIMER_DMA_CH3 D	通道3比较/捕获 DMA请求, TIMERx(x=0..4,7)
TIMER_DMA_CMT D	换相DMA更新请求, TIMERx(x=0,7)
TIMER_DMA_TRG D	触发DMA请求使能, TIMERx(x=0..4,7,8,11)
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 3-917. 函数 timer\_dma\_disable

函数名称	timer_dma_disable
函数原型	void timer_dma_disable (uint32_t timer_periph, uint16_t dma);
功能描述	外设TIMERx的DMA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}	
dma	DMA源
TIMER_DMA_UPD	更新DMA请求, TIMERx(x=0..16)
TIMER_DMA_CH0 D	通道0比较/捕获 DMA请求, TIMERx(x=0..4,7,8,11,14..16)
TIMER_DMA_CH1 D	通道1比较/捕获 DMA请求, TIMERx(x=0..4,7,8,11,14)
TIMER_DMA_CH2 D	通道2比较/捕获 DMA请求, TIMERx(x=0..4,7)
TIMER_DMA_CH3 D	通道3比较/捕获 DMA请求, TIMERx(x=0..4,7)
TIMER_DMA_CMT D	换相DMA更新请求, TIMERx(x=0,7)
TIMER_DMA_TRG D	触发DMA请求使能, TIMERx(x=0..4,7,8,11)
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 3-918. 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TIMERx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..4,7,14..16)	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	当更新事件发生时，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 y 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 3-919. 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TIMERx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_DMACFG_DMATA_CTL0, TIMERx(x=0..4,7,14..16)
TIMER_DMACFG_DMATA_CTL1	DMA传输起始地址：TIMER_DMACFG_DMATA_CTL1, TIMERx(x=0..4,7,14..16)

<code>TIMER_DMACFG_DMATA_SMCFG</code>	DMA传输起始地址: <code>TIMER_DMACFG_DMATA_SMCFG</code> , <code>TIMERx(x=0..4,7,14..16)</code>
<code>TIMER_DMACFG_DMATA_DMAINTEN</code>	DMA传输起始地址: <code>TIMER_DMACFG_DMATA_DMAINTEN</code> , <code>TIMERx(x=0..4,7,14..16)</code>
<code>TIMER_DMACFG_DMATA_INTF</code>	DMA传输起始地址: <code>TIMER_DMACFG_DMATA_INTF</code> , <code>TIMERx(x=0..4,7,14..16)</code>
<code>TIMER_DMACFG_DMATA_SWEVG</code>	DMA传输起始地址: <code>TIMER_DMACFG_DMATA_SWEVG</code> , <code>TIMERx(x=0..4,7,14..16)</code>
<code>TIMER_DMACFG_DMATA_CHCTL0</code>	DMA传输起始地址: <code>TIMER_DMACFG_DMATA_CHCTL0</code> , <code>TIMERx(x=0..4,7,14..16)</code>
<code>TIMER_DMACFG_DMATA_CHCTL1</code>	DMA传输起始地址: <code>TIMER_DMACFG_DMATA_CHCTL1</code> , <code>TIMERx(x=0..4,7,14..16)</code>
<code>TIMER_DMACFG_DMATA_CHCTL2</code>	DMA传输起始地址: <code>TIMER_DMACFG_DMATA_CHCTL2</code> , <code>TIMERx(x=0..4,7,14..16)</code>
<code>TIMER_DMACFG_DMATA_CNT</code>	DMA传输起始地址: <code>TIMER_DMACFG_DMATA_CNT</code> , <code>TIMERx(x=0..4,7,14..16)</code>
<code>TIMER_DMACFG_DMATA_PSC</code>	DMA传输起始地址: <code>TIMER_DMACFG_DMATA_PSC</code> , <code>TIMERx(x=0..4,7,14..16)</code>
<code>TIMER_DMACFG_DMATA_CAR</code>	DMA传输起始地址: <code>TIMER_DMACFG_DMATA_CAR</code> , <code>TIMERx(x=0..4,7,14..16)</code>
<code>TIMER_DMACFG_DMATA_CREP</code>	DMA传输起始地址: <code>TIMER_DMACFG_DMATA_CREP</code> , <code>TIMERx(x=0..4,7,14..16)</code>
<code>TIMER_DMACFG_DMATA_CH0CV</code>	DMA传输起始地址: <code>TIMER_DMACFG_DMATA_CH0CV</code> , <code>TIMERx(x=0..4,7,14..16)</code>
<code>TIMER_DMACFG_DMATA_CH1CV</code>	DMA传输起始地址: <code>TIMER_DMACFG_DMATA_CH1CV</code> , <code>TIMERx(x=0..4,7,14..16)</code>
<code>TIMER_DMACFG_DMATA_CH2CV</code>	DMA传输起始地址: <code>TIMER_DMACFG_DMATA_CH2CV</code> , <code>TIMERx(x=0..4,7,14..16)</code>
<code>TIMER_DMACFG_DMATA_CH3CV</code>	DMA传输起始地址: <code>TIMER_DMACFG_DMATA_CH3CV</code> , <code>TIMERx(x=0..4,7,14..16)</code>
<code>TIMER_DMACFG_DMATA_CCHP</code>	DMA传输起始地址: <code>TIMER_DMACFG_DMATA_CCHP</code> , <code>TIMERx(x=0..4,7,14..16)</code>
<code>TIMER_DMACFG_DMATA_DMACFG</code>	DMA传输起始地址: <code>TIMER_DMACFG_DMATA_DMACFG</code> , <code>TIMERx(x=0..4,7,14..16)</code>
<code>TIMER_DMACFG_DMATA_DMATB</code>	DMA传输起始地址: <code>TIMER_DMACFG_DMATA_DMATB</code> , <code>TIMERx(x=0..4,7,14..16)</code>
输入参数{in}	
<code>dma_lenth</code>	DMA传输长度
<code>TIMER_DMACFG_DMATC_xTRANSFER</code>	<code>x=1..18</code> , DMA传输 <code>x</code> 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 3-920. 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, uint16_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}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
event	事件源
TIMER_EVENT_SR_C_UPG	更新事件产生, TIMERx(x=0..16)
TIMER_EVENT_SR_C_CH0G	通道0捕获或比较事件发生, TIMERx(x=0..4,7,8..16)
TIMER_EVENT_SR_C_CH1G	通道1捕获或比较事件发生, TIMERx(x=0..4,7,8,11,14)
TIMER_EVENT_SR_C_CH2G	通道2捕获或比较事件发生, TIMERx(x=0..4,7)
TIMER_EVENT_SR_C_CH3G	通道3捕获或比较事件发生, TIMERx(x=0..4,7)
TIMER_EVENT_SR_C_CMTG	通道换相更新事件发生, TIMERx(x=0,7,14..16)
TIMER_EVENT_SR_C_TRGG	触发事件产生, TIMERx(x=0,7,14)
TIMER_EVENT_SR_C_BRKG	产生中止事件, TIMERx(x=0,7,14,16)
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 3-921. 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 3-922. 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=0,7,14..16)	TIMER外设选择

输入参数{in}	
<b>breakpara</b>	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 3-923. 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TIMERx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}	
<b>timer_periph</b>	TIMER外设
TIMERx(x=0,7,14..16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 break function*/
```

```
timer_break_enable(TIMERO);
```

### 函数 timer\_break\_disable

函数timer\_break\_disable描述见下表：

表 3-924. 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TIMERx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=0,7,14..16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMERO break function*/
```

```
timer_break_disable(TIMERO);
```

### 函数 timer\_automatic\_output\_enable

函数timer\_automatic\_output\_enable描述见下表：

表 3-925. 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=0,7,14..16)	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 3-926. 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=0,7,14..16)	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 3-927. 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=0,7,14..16)	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\_control\_shadow\_config

函数timer\_channel\_control\_shadow\_config描述见下表：

**表 3-928. 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, ControlStatus newvalue);
功能描述	通道换相控制影子配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,14..16)	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 3-929. 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, uint32_t ccuctl);
功能描述	通道换相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=0,7,14..16)	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_UPDATECTL_CCUC	CMTG位被置1时更新影子寄存器
TIMER_UPDATECTL_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_CCUC);
```

### 函数 timer\_channel\_output\_struct\_para\_init

函数timer\_channel\_output\_struct\_para\_init描述见下表：

表 3-930. 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* ocpa);
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}	
ocpa	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 3-931. 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TIMERx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	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0, TIMERx(x=0..4,7..16)
TIMER_CH_1	通道1, TIMERx(x=0..4,7,8,11,14)
TIMER_CH_2	通道2, TIMERx(x=0..4,7)
TIMER_CH_3	通道3, TIMERx(x=0..4,7)
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:

```
/* 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 3-932. 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TIMERx通道输出比较模式
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}	
channel	待配置通道
TIMER_CH_0	通道0, TIMERx(x=0..4,7..16)
TIMER_CH_1	通道1, TIMERx(x=0..4,7,8,11,14)
TIMER_CH_2	通道2, TIMERx(x=0..4,7)
TIMER_CH_3	通道3, TIMERx(x=0..4,7)
输入参数{in}	
ocmode	通道输出比较模式
TIMER_OC_MODE_TIMING	冻结模式
TIMER_OC_MODE_ACTIVE	匹配时设置为高
TIMER_OC_MODE_INACTIVE	匹配时设置为低
TIMER_OC_MODE_TOGGLE	匹配时翻转
TIMER_OC_MODE_LOW	强制为低
TIMER_OC_MODE_HIGH	强制为高
TIMER_OC_MODE_PWM0	PWM模式0
TIMER_OC_MODE_PWM1	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 3-933. 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TIMERx的通道输出比较值
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	待配置通道
TIMER_CH_0	通道0, TIMERx(x=0..4,7..16)
TIMER_CH_1	通道1, TIMERx(x=0..4,7,8,11,14)
TIMER_CH_2	通道2, TIMERx(x=0..4,7)
TIMER_CH_3	通道3, TIMERx(x=0..4,7)
输入参数{in}	
pulse	通道输出比较值 (0~0xFFFFFFFF)
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);
```

### 函数 timer\_channel\_output\_shadow\_config

函数timer\_channel\_output\_shadow\_config描述见下表：

**表 3-934. 函数 timer\_channel\_output\_shadow\_config**

函数名称	timer_channel_output_shadow_config
------	------------------------------------

函数原型	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
功能描述	配置TIMERx通道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	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0, TIMERx(x=0..4,7..16)
TIMER_CH_1	通道1, TIMERx(x=0..4,7,8,11,14)
TIMER_CH_2	通道2, TIMERx(x=0..4,7)
TIMER_CH_3	通道3, TIMERx(x=0..4,7)
输入参数{in}	
ocshadow	输出比较影子寄存器功能状态
TIMER_OC_SHADOW_ENABLE	使能输出比较影子寄存器
TIMER_OC_SHADOW_DISABLE	禁能输出比较影子寄存器
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\_fast\_config

函数timer\_channel\_output\_fast\_config描述见下表:

表 3-935. 函数 timer\_channel\_output\_fast\_config

函数名称	timer_channel_output_fast_config
函数原型	void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);
功能描述	配置TIMERx通道输出比较快速功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设

<i>TIMERx</i>	参考具体参数
输入参数{in}	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> ( <i>x</i> =0..4,7..16)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11,14)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> ( <i>x</i> =0..4,7)
输入参数{in}	
<b>ocfast</b>	通道输出比较快速功能状态
<i>TIMER_OC_FAST_ENABLE</i>	通道输出比较快速功能使能
<i>TIMER_OC_FAST_DISABLE</i>	通道输出比较快速功能禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 output fast function */
```

```
timer_channel_output_fast_config(TIMER0, TIMER_CH_0, TIMER_OC_FAST_ENABLE);
```

### 函数 timer\_channel\_output\_clear\_config

函数timer\_channel\_output\_clear\_config描述见下表:

表 3-936. 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 <i>TIMERx</i> 的通道输出比较清0功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	<i>TIMER</i> 外设
<i>TIMERx</i> ( <i>x</i> =0..4,7)	参考具体参数
输入参数{in}	
<b>channel</b>	待配置通道
<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>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
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 3-937. 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>	待配置通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> (x=0..4,7..16)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (x=0..4,7,8,11,14)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (x=0..4,7)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (x=0..4,7)
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
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 3-938. 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</i>	TIMER外设选择
输入参数{in}	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0 (TIMERx(x=0,7,14..16))
<i>TIMER_CH_1</i>	通道1 (TIMERx(x=0,7,14))
<i>TIMER_CH_2</i>	通道2 (TIMERx(x=0,7))
输入参数{in}	
<b>ocnpolarity</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 3-939. 函数 timer\_channel\_output\_state\_config

函数名称	timer_channel_output_state_config
函数原型	void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
功能描述	配置通道状态
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	待配置通道
TIMER_CH_0	通道0, (TIMERx(x=0..4,7..16))
TIMER_CH_1	通道1, (TIMERx(x=0..4,7,8,11,14))
TIMER_CH_2	通道2, (TIMERx(x=0..4,7))
TIMER_CH_3	通道3, (TIMERx(x=0..4,7))
输入参数{in}	
state	通道状态
TIMER_CCX_ENABLE	通道使能
TIMER_CCX_DISABLE	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 3-940. 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}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0 (TIMERx(x=0,7,14..16))
TIMER_CH_1	通道1 (TIMERx(x=0,7))
TIMER_CH_2	通道2 (TIMERx(x=0,7))
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 3-941. 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 3-942. 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TIMERx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	待配置通道
TIMER_CH_0	通道0, (TIMERx(x=0..4,7..16))
TIMER_CH_1	通道1, (TIMERx(x=0..4,7,8,11,14))
TIMER_CH_2	通道2, (TIMERx(x=0..4,7))
TIMER_CH_3	通道3, (TIMERx(x=0..4,7))
输入参数{in}	
icpara	输入捕获结构体, 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 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 3-943. 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TIMERx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	待配置通道
TIMER_CH_0	通道0, TIMERx (x=0..4,7..13)
TIMER_CH_1	通道1, TIMERx (x=0..4,7,8,11)
TIMER_CH_2	通道2, TIMERx (x=0..4,7)
TIMER_CH_3	通道3, TIMERx (x=0..4,7)
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 3-944. 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	待配置通道
TIMER_CH_0	通道0, (TIMERx(x=0..4,7..16))
TIMER_CH_1	通道1, (TIMERx(x=0..4,7,8,11,14))
TIMER_CH_2	通道2, (TIMERx(x=0..4,7))
TIMER_CH_3	通道3, (TIMERx(x=0..4,7))
输出参数{out}	
-	-
返回值	
uint32_t	通道输入捕获值, (0~0xFFFFFFFF)

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 3-945. 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TIMERx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..4,7,8,11,14)	TIMER外设选择
输入参数{in}	
channel	待配置通道

<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
输入参数{in}	
<b>icpwm</b>	输入捕获结构体, 详见 <a href="#">结构体 <i>timer_ic_parameter_struct</i></a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 input pwm capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_pwm_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);
```

### 函数 **timer\_hall\_mode\_config**

函数timer\_hall\_mode\_config描述见下表:

表 3-946. 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TIMERx的HALL接口功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0..4,7)</i>	TIMER外设选择
输入参数{in}	
<b>hallmode</b>	HALL接口功能状态
<i>TIMER_HALLINTE</i> <i>RFACE_ENABLE</i>	HALL接口使能
<i>TIMER_HALLINTE</i> <i>RFACE_DISABLE</i>	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\_input\_trigger\_source\_select

函数timer\_input\_trigger\_source\_select描述见下表：

表 3-947. 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	TIMERx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..4,7,8,11,14)	TIMER外设选择
输入参数{in}	
intrigger	待选择的触发源
TIMER_SMCFG_T RGSEL_ITI0	内部触发输入0(ITI0, TIMERx(x=0..4,8,11,14))
TIMER_SMCFG_T RGSEL_ITI1	内部触发输入1(ITI1, TIMERx(x=0..4,8,11,14))
TIMER_SMCFG_T RGSEL_ITI2	内部触发输入2(ITI2, TIMERx(x=0..4))
TIMER_SMCFG_T RGSEL_ITI3	内部触发输入3(ITI3, TIMERx(x=0..4))
TIMER_SMCFG_T RGSEL_CIOF_ED	CI0的边沿标志位 (CIOF_ED, TIMERx(x=0..4,8,11,14))
TIMER_SMCFG_T RGSEL_CIOFE0	滤波后的通道0输入 (CIOFE0, TIMERx(x=0..4,8,11,14))
TIMER_SMCFG_T RGSEL_CI1FE1	滤波后的通道1输入(CI1FE1, TIMERx(x=0..4,8,11,14))
TIMER_SMCFG_T RGSEL_ETIFP	滤波后的外部触发输入(ETIFP, TIMERx(x=0..4,7))
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 3-948. 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 outrigger);
功能描述	选择TIMERx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	TIMER外设选择
输入参数{in}	
outrigger	主模式输出触发
TIMER_TRI_OUT_SRC_RESET	复位。TIMERx_SWEVG寄存器的UPG位被置1或从模式控制器产生复位触发一次TRGO脉冲，后一种情况下，TRGO上的信号相对实际的复位会有一个延迟。
TIMER_TRI_OUT_SRC_ENABLE	使能。此模式可用于同时启动多个定时器或控制在一段时间内使能从定时器。主模式控制器选择计数器使能信号作为触发输出TRGO。当CEN控制位被置1或者暂停模式下触发输入为高电平时，计数器使能信号被置1。在暂停模式下，计数器使能信号受控于触发输入，在触发输入和TRGO上会有一个延迟，除非选择了主/从模式。
TIMER_TRI_OUT_SRC_UPDATE	更新。主模式控制器选择更新事件作为TRGO。
TIMER_TRI_OUT_SRC_CC0	捕获/比较脉冲.通道0在发生一次捕获或一次比较成功时，主模式控制器产生一个TRGO脉冲
TIMER_TRI_OUT_SRC_O0CPRE	比较。在这种模式下主模式控制器选择O0CPRE信号被用于作为触发输出TRGO
TIMER_TRI_OUT_SRC_O1CPRE	比较。在这种模式下主模式控制器选择O1CPRE信号被用于作为触发输出TRGO
TIMER_TRI_OUT_SRC_O2CPRE	比较。在这种模式下主模式控制器选择O2CPRE信号被用于作为触发输出TRGO
TIMER_TRI_OUT_SRC_O3CPRE	比较。在这种模式下主模式控制器选择O3CPRE信号被用于作为触发输出TRGO
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 3-949. 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	TIMERx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..4,7,8,11,14)	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 3-950. 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, uint8_t masterslave);
功能描述	TIMERx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..4,7,8,11,14)	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 3-951. 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TIMERx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(x=0..4,7)</i>	TIMER外设选择
输入参数{in}	
<b>extprescaler</b>	外部触发预分频
<i>TIMER_EXT_TRI_P</i> <i>SC_OFF</i>	不分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV2</i>	2分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV4</i>	4分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV8</i>	8分频
输入参数{in}	
<b>expolarity</b>	外部触发输入极性
<i>TIMER_ETP_FALLI</i> <i>NG</i>	低电平或者下降沿有效
<i>TIMER_ETP_RISIN</i> <i>G</i>	高电平或者上升沿有效
输入参数{in}	
<b>extfilter</b>	外部触发滤波控制（0~15）
输出参数{out}	
-	-
返回值	
-	-

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 3-952. 函数 timer\_quadrature\_decoder\_mode\_config**

函数名称	timer_quadrature_decoder_mode_config
函数原型	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity);
功能描述	TIMERx配置为编码器模式
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设

TIMERx(x=0..4,7)	TIMER外设选择
输入参数{in}	
decomode	编码器模式
TIMER_QUAD_DECODER_MODE0	根据CI0FE0的电平，计数器在CI1FE1的边沿向上/下计数
TIMER_QUAD_DECODER_MODE1	根据CI1FE1的电平，计数器在CI0FE0的边沿向上/下计数
TIMER_QUAD_DECODER_MODE2	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
输入参数{in}	
ic0polarity	IC0极性
TIMER_IC_POLARITY_RISING	捕获上升边沿
TIMER_IC_POLARITY_FALLING	捕获下降边沿
TIMER_IC_POLARITY_BOTH_EDGE	双边沿有效
输入参数{in}	
ic1polarity	IC1极性
TIMER_IC_POLARITY_RISING	捕获上升边沿
TIMER_IC_POLARITY_FALLING	捕获下降边沿
TIMER_IC_POLARITY_BOTH_EDGE	双边沿有效
输出参数{out}	
-	-
返回值	
-	-

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 3-953. 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	TIMERx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..4,7,8,11,14)	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 3-954. 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TIMERx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..4,7,8,11,14)	TIMER外设选择
输入参数{in}	
intrigger	被选择的内部触发源
TIMER_SMCFG_T RGSEL_ITI0	选择内部触发0 (ITI0)为时钟源
TIMER_SMCFG_T RGSEL_ITI1	选择内部触发1 (ITI1)为时钟源
TIMER_SMCFG_T RGSEL_ITI2	选择内部触发2 (ITI2)为时钟源
TIMER_SMCFG_T 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 3-955. 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TIMERx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..4,7,8,11,14)	TIMER外设选择
输入参数{in}	
extrigger	外部触发源
TIMER_SMCFG_TRGSEL_CIOF_ED	CI0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(CI1FE1)
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 3-956. 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TIMERx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..4,7,8,11,14)	TIMER外设选择
输入参数{in}	
extprescaler	ETI触发源预分频值
TIMER_EXT_TRI_P SC_OFF	不分频
TIMER_EXT_TRI_P SC_DIV2	2分频
TIMER_EXT_TRI_P SC_DIV4	4分频
TIMER_EXT_TRI_P SC_DIV8	8分频
输入参数{in}	
expolarity	ETI触发源极性
TIMER_ETP_FALLI NG	下降沿或者低电平有效
TIMER_ETP_RISIN 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 3-957. 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TIMERx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..4,7)	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 3-958. 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	TIMERx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..4,7)	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 3-959. 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 TIMERxt 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
TIMERx(x=0)	TIMER 外设选择
输入参数{in}	
remap	重映射功能选择
TIMER0_IT0_RMP_ TIMER4_TRGO	内部触发输入 0 重映射到 TIMER4_TRGO
TIMER0_IT0_RMP_ TIMER14_TRGO	内部触发输入 0 重映射到 TIMER14_TRGO

<i>TIMER14_TRGO</i>	
<i>TIMER0_IT3_RMP_</i> <i>TIMER3_TRGO</i>	内部触发输入 3 重映射到 <i>TIMER3_TRGO</i>
<i>TIMER0_IT3_RMP_</i> <i>TIMER16_TRGO</i>	内部触发输入 3 重映射到 <i>TIMER16_TRGO</i>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 internal trigger input0 remap */
```

```
timer_channel_remap_config (TIMER0, TIMER0_IT0_RMP_TIMER4_TRGO);
```

### 函数 timer\_write\_chxval\_register\_config

函数timer\_write\_chxval\_register\_config描述见下表:

表 3-960. 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 ccse1);
功能描述	配置TIMERx写CHxVAL选择位
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (x=0..4,7..13..16)	TIMER外设选择
输入参数{in}	
ccse1	写CHxVAL寄存器选择位
<i>TIMER_CHVSEL_D</i> <i>ISABLE</i>	无影响
<i>TIMER_CHVSEL_E</i> <i>NABLE</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 3-961. 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
TIMEx(x=0,7,14..16)	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\_flag\_get

函数timer\_flag\_get描述见下表：

表 3-962. 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TIMEx的状态标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMEx	参考具体参数
输入参数{in}	

flag	状态标志
<i>TIMER_FLAG_UP</i>	更新标志, <i>TIMERx</i> ( <i>x</i> =0..16)
<i>TIMER_FLAG_CH0</i>	通道0比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0..4,7..16)
<i>TIMER_FLAG_CH1</i>	通道1比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11,14)
<i>TIMER_FLAG_CH2</i>	通道2比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_FLAG_CH3</i>	通道3比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_FLAG_CMT</i>	通道换相更新标志, <i>TIMERx</i> ( <i>x</i> =0,7,14..16)
<i>TIMER_FLAG_TRG</i>	触发标志, <i>TIMERx</i> ( <i>x</i> =0,7,8,11,14)
<i>TIMER_FLAG_BRK</i>	中止标志位, <i>TIMERx</i> ( <i>x</i> =0,7,14..16)
<i>TIMER_FLAG_CH0</i> O	通道0捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0..4,7..11,14..16)
<i>TIMER_FLAG_CH1</i> O	通道1捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11,14)
<i>TIMER_FLAG_CH2</i> O	通道2捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_FLAG_CH3</i> O	通道3捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0..4,7)
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 3-963. 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 <i>TIMERx</i> 状态标志
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}	
flag	状态标志
<i>TIMER_FLAG_UP</i>	更新标志, <i>TIMERx</i> ( <i>x</i> =0..16)

<i>TIMER_FLAG_CH0</i>	通道0比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0..4,7..16)
<i>TIMER_FLAG_CH1</i>	通道1比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11,14)
<i>TIMER_FLAG_CH2</i>	通道2比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_FLAG_CH3</i>	通道3比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_FLAG_CMT</i>	通道换相更新标志, <i>TIMERx</i> ( <i>x</i> =0,7,14..16)
<i>TIMER_FLAG_TRG</i>	触发标志, <i>TIMERx</i> ( <i>x</i> =0,7,8,11,14)
<i>TIMER_FLAG_BRK</i>	中止标志位, <i>TIMERx</i> ( <i>x</i> =0,7,14..16)
<i>TIMER_FLAG_CH0</i> O	通道0捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0..4,7..11,14..16)
<i>TIMER_FLAG_CH1</i> O	通道1捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11,14)
<i>TIMER_FLAG_CH2</i> O	通道2捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_FLAG_CH3</i> O	通道3捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0..4,7)
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 3-964. 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, <i>TIMERx</i> ( <i>x</i> =0..13..16)
TIMER_INT_CH0	通道0比较/捕获中断, <i>TIMERx</i> ( <i>x</i> =0..4,7..13..16)
TIMER_INT_CH1	通道1比较/捕获中断, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11,14)
TIMER_INT_CH2	通道2比较/捕获中断, <i>TIMERx</i> ( <i>x</i> =0..4,7)

<i>TIMER_INT_CH3</i>	通道3比较/捕获中断, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_INT_CMT</i>	换相更新中断, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_TRG</i>	触发中断, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11,14)
<i>TIMER_INT_BRK</i>	中止中断, <i>TIMERx</i> ( <i>x</i> =0,7,14..16)
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 3-965. 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 <i>TIMERx</i> 中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	<i>TIMER</i> 外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
<b>interrupt</b>	中断源
<i>TIMER_INT_UP</i>	更新中断, <i>TIMERx</i> ( <i>x</i> =0..13..16)
<i>TIMER_INT_CH0</i>	通道0比较/捕获中断, <i>TIMERx</i> ( <i>x</i> =0..4,7..13..16)
<i>TIMER_INT_CH1</i>	通道1比较/捕获中断, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11,14)
<i>TIMER_INT_CH2</i>	通道2比较/捕获中断, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_INT_CH3</i>	通道3比较/捕获中断, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_INT_CMT</i>	换相更新中断, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_TRG</i>	触发中断, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11,14)
<i>TIMER_INT_BRK</i>	中止中断, <i>TIMERx</i> ( <i>x</i> =0,7,14..16)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the TIMER0 update interrupt */
```

```
timer_interrupt_disable(TIMERO, TIMER_INT_UP);
```

### 函数 timer\_interrupt\_flag\_get

函数timer\_interrupt\_flag\_get描述见下表:

表 3-966. 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TIMERx中断标志
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}	
int_flag	中断源
TIMER_INT_FLAG_UP	更新中断, TIMERx(x=0..13..16)
TIMER_INT_FLAG_CH0	通道0比较/捕获中断, TIMERx(x=0..4,7..13..16)
TIMER_INT_FLAG_CH1	通道1比较/捕获中断, TIMERx(x=0..4,7,8,11,14)
TIMER_INT_FLAG_CH2	通道2比较/捕获中断, TIMERx(x=0..4,7)
TIMER_INT_FLAG_CH3	通道3比较/捕获中断, TIMERx(x=0..4,7)
TIMER_INT_FLAG_CMT	换相更新中断, TIMERx(x=0,7)
TIMER_INT_FLAG_TRG	触发中断, TIMERx(x=0..4,7,8,11,14)
TIMER_INT_FLAG_BRK	中止中断, TIMERx(x=0,7,14..16)
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如:

```
/* get TIMERO update interrupt flag */
```

```
FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = timer_interrupt_flag_get(TIMERO, TIMER_INT_FLAG_UP);
```

## 函数 timer\_interrupt\_flag\_clear

函数timer\_interrupt\_flag\_clear描述见下表：

**表 3-967. 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TIMERx的中断标志
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}	
int_flag	中断源
TIMER_INT_FLAG_UP	更新中断, TIMERx(x=0..13..16)
TIMER_INT_FLAG_CH0	通道0比较/捕获中断, TIMERx(x=0..4,7..13..16)
TIMER_INT_FLAG_CH1	通道1比较/捕获中断, TIMERx(x=0..4,7,8,11,14)
TIMER_INT_FLAG_CH2	通道2比较/捕获中断, TIMERx(x=0..4,7)
TIMER_INT_FLAG_CH3	通道3比较/捕获中断, TIMERx(x=0..4,7)
TIMER_INT_FLAG_CMT	换相更新中断, TIMERx(x=0,7)
TIMER_INT_FLAG_TRG	触发中断, TIMERx(x=0..4,7,8,11,14)
TIMER_INT_FLAG_BRK	中止中断, TIMERx(x=0,7,14..16)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear TIMER0 update interrupt flag */
```

```
timer_interrupt_flag_clear(TIMER0, TIMER_INT_FLAG_UP);
```

## 3.27. TMU

三角数学单元(TMU)是一个完全可配置的单元,可执行常见的三角运算和算术运算操作。TMU总共有9种运算操作,其操作数据必须符合IEEE-32位单精度浮点格式。章节[3.27.1](#)描述了TMU的寄存器列表,章节[3.27.2](#)对TMU库函数进行说明。

### 3.27.1. 外设寄存器说明

TMU寄存器列表如下表所示:

表 3-968. TMU 寄存器

寄存器名称	寄存器描述
TMU_IDATA0	输入数据寄存器 0
TMU_IDATA1	输入数据寄存 1
TMU_CTL	控制寄存器
TMU_DATA0	数据寄存器 0
TMU_DATA1	数据寄存器 1
TMU_STAT	状态寄存器

### 3.27.2. 外设库函数说明

TMU库函数列表如下表所示:

表 3-969. TMU 库函数

库函数名称	库函数描述
tmu_deinit	复位TMU
tmu_enable	使能TMU
tmu_mode_set	配置TMU模式
tmu_idata0_write	写输入数据0寄存器
tmu_idata1_write	写输入数据1寄存器
tmu_data0_read	读数据0寄存器
tmu_data1_read	读数据1寄存器
tmu_interrupt_enable	使能TMU中断
tmu_interrupt_disable	禁止TMU中断
tmu_flag_get	查询TMU状态标志位
tmu_interrupt_flag_get	查询TMU中断标志位

#### 函数 tmu\_deinit

函数tmu\_deinit描述见下表:

表 3-970. 函数 tmu\_deinit

函数名称	tmu_deinit
函数原形	tmu_deinit(void);
功能描述	复位TMU
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 tmu */
```

```
tmu_deinit();
```

### 函数 tmu\_enable

函数tmu\_enable描述见下表：

表 3-971. 函数 tmu\_enable

函数名称	tmu_enable
函数原形	tmu_enable (void);
功能描述	使能TMU
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 TMU */
```

```
tmu_enable ();
```

### 函数 tmu\_mode\_set

函数tmu\_mode\_set描述见下表：

表 3-972. 函数 tmu\_mode\_set

函数名称	tmu_mode_set
------	--------------

函数原形	void tmu_mode_set(uint32_t modex);
功能描述	配置TMU模式
先决条件	-
被调用函数	-
输入参数{in}	
<b>modex</b>	TMU操作模式
TMU_MODE0	操作模式0
TMU_MODE1	操作模式1
TMU_MODE2	操作模式2
TMU_MODE3	操作模式3
TMU_MODE4	操作模式4
TMU_MODE5	操作模式4
TMU_MODE6	操作模式6
TMU_MODE7	操作模式7
TMU_MODE8	操作模式8
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the TMU mode0 */
tmu_mode_set (TMU_MODE0);
```

### 函数 tmu\_idata0\_write

函数tmu\_idata0\_write描述见下表：

表 3-973. 函数 tmu\_idata0\_write

函数名称	tmu_idata0_write
函数原形	tmu_idata0_write(uint32_t idata0);
功能描述	写输入数据0寄存器
先决条件	-
被调用函数	-
输入参数{in}	
<b>idata0</b>	写入寄存器的值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write the data to TMU input data0 regisetr */
```

```
tmu_idata0_write (0xBF000000);
```

### 函数 tmu\_idata1\_write

函数tmu\_idata1\_write描述见下表:

**表 3-974. 函数 tmu\_idata1\_write**

函数名称	tmu_idata1_write
函数原形	tmu_idata1_write(uint32_t idata1);
功能描述	写输入数据1寄存器
先决条件	-
被调用函数	-
输入参数{in}	
idata1	写入寄存器的值
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* write the data to TMU input data1 regisetr */
```

```
tmu_idata1_write (0xBF000000);
```

### 函数 tmu\_data0\_read

函数tmu\_data0\_read描述见下表:

**表 3-975. 函数 tmu\_data0\_read**

函数名称	tmu_data0_read
函数原形	uint32_t tmu_data0_read(void);
功能描述	读数据0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0寄存器读取32位单精度浮点格式的数据

例如:

```
/* read the data from TMU data0 regisetr*/
```

```
uint32_t tmu_value = 0;
```

```
tmu_value = tmu_data0_read ();
```

### 函数 tmu\_data1\_read

函数tmu\_data1\_read描述见下表:

**表 3-976. 函数 tmu\_data1\_read**

函数名称	tmu_data1_read
函数原形	uint32_t tmu_data1_read(void);
功能描述	读数据1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	
Uin32_t	从数据0寄存器读取32位单精度浮点格式的数据

例如:

```
/* read tmu data1 register */
Uin32_t tmu_value = 0;
tmu_value = tmu_data1_read ();
```

### 函数 tmu\_interrupt\_enable

函数tmu\_interrupt\_enable描述见下表:

**表 3-977. 函数 tmu\_interrupt\_enable**

函数名称	tmu_interrupt_enable
函数原形	void tmu_interrupt_enable(void);
功能描述	使能TMU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 TTMU interrupt */
```

```
tmu_interrupt_enable ();
```

### 函数 tmu\_interrupt\_disable

函数tmu\_interrupt\_disable描述见下表:

表 3-978. 函数 tmu\_interrupt\_disable

函数名称	tmu_interrupt_disable
函数原形	void tmu_interrupt_disable (void);
功能描述	禁止TMU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:

```
/* disable TTMU interrupt */
```

```
tmu_interrupt_disable ();
```

### 函数 tmu\_flag\_get

函数tmu\_flag\_get描述见下表:

表 3-979. 函数 tmu\_flag\_get

函数名称	tmu_flag_get
函数原形	FlagStatus tmu_flag_get(uint32_t flag);
功能描述	查询TMU状态标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	TMU状态标志
TMU_FLAG_OVRF	TMU上溢标志
TMU_FLAG_UDRF	TMU下溢标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET/RESET

例如:

```
/* check the TMU overflow flag */
```

```
tmu_flag_get (TMU_FLAG_OVRF);
```

### 函数 tmu\_interrupt\_flag\_get

函数tmu\_interrupt\_flag\_get描述见下表:

表 3-980. 函数 tmu\_interrupt\_flag\_get

函数名称	tmu_interrupt_flag_get
函数原形	FlagStatus tmu_interrupt_flag_get(uint32_t interrupt);
功能描述	查询TMU中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	中断标志位
TMU_INT_FLAG_C FIF	TMU计算完成中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET/RESET

例如:

```
/* check teh TMU interrupt flag */
```

```
tmu_interrupt_flag_get(TMU_INT_FLAG_CFIF);
```

## 3.28. 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[3.28.1](#)描述了USART的寄存器列表, 章节[3.28.2](#)对USART库函数进行说明。

### 3.28.1. 外设寄存器说明

USART寄存器列表如下表所示:

表 3-981. USART 寄存器

寄存器名称	寄存器描述
USART_STAT0	状态寄存器 0 (USARTx, x=0..4)
USART_DATA	数据寄存器 (USARTx, x=0..4)
USART_BAUD	波特率寄存器 (USARTx, x=0..4)
USART_CTL0	控制寄存器 0 (USARTx, x=0..4)
USART_CTL1	控制寄存器 1 (USARTx, x=0..4)
USART_CTL2	控制寄存器 2 (USARTx, x=0..4)
USART_GP	保护时间和预分频器寄存器 (USARTx, x=0..4)

寄存器名称	寄存器描述
USART_CTL3	控制寄存器 3 (USARTx, x=0..4)
USART_RT	接收超时寄存器 (USARTx, x=0..4)
USART_STAT1	状态寄存器 1 (USARTx, x=0..4)
USART_GDCTL	GD 控制寄存器 (USARTx, x=0..4)
USART5_CTL0	控制寄存器 0 (USARTx, x=5)
USART5_CTL1	控制寄存器 1 (USARTx, x=5)
USART5_CTL2	控制寄存器 2 (USARTx, x=5)
USART5_BAUD	波特率寄存器 (USARTx, x=5)
USART5_GP	保护时间和预分频寄存器 (USARTx, x=5)
USART5_RT	接收超时寄存器 (USARTx, x=5)
USART5_CMD	请求寄存器 (USARTx, x=5)
USART5_STAT	状态寄存器 (USARTx, x=5)
USART5_INTC	中断标志清除寄存器 (USARTx, x=5)
USART5_RDATA	接收数据寄存器 (USARTx, x=5)
USART5_TDATA	发送数据寄存器 (USARTx, x=5)
USART5_CHC	兼容性控制寄存器 (USARTx, x=5)
USART5_RFC5	接收 FIFO 控制和状态寄存器 (USARTx, x=5)

### 3.28.2. 外设库函数说明

USART库函数列表如下表所示：

**表 3-982. USART 库函数**

库函数名称	库函数描述
usart_deinit	复位外设 USART (USARTx, x=0..5)
usart_baudrate_set	配置 USART 波特率 (USARTx, x=0..5)
usart_parity_config	配置 USART 奇偶校验 (USARTx, x=0..5)
usart_word_length_set	配置 USART 字长 (USARTx, x=0..5)
usart_stop_bit_set	配置 USART 停止位 (USARTx, x=0..5)
usart_enable	使能 USART (USARTx, x=0..5)
usart_disable	失能 USART (USARTx, x=0..5)
usart_transmit_config	USART 发送配置 (USARTx, x=0..5)
usart_receive_config	USART 接收配置 (USARTx, x=0..5)
usart_oversample_config	配置 USART 过采样模式 (USARTx, x=0..5)
usart_sample_bit_config	配置 USART 过采样方法 (USARTx, x=0..5)
usart_receiver_timeout_enable	使能 USART 接收超时 (USARTx, x=0..5)
usart_receiver_timeout_disable	失能 USART 接收超时 (USARTx, x=0..5)
usart_receiver_timeout_threshold_config	设置 USART 接收超时阈值 (USARTx, x=0..5)
usart_data_transmit	USART 发送数据功能 (USARTx, x=0..5)
usart_data_receive	USART 接收数据功能 (USARTx, x=0..5)
usart_mute_mode_enable	使能 USART 静默模式 (USARTx, x=0..5)

库函数名称	库函数描述
usart_mute_mode_disable	失能 USART 静默模式 (USARTx, x=0..5)
usart_mute_mode_wakeup_config	配置 USART 静默模式唤醒方式 (USARTx, x=0..5)
usart_lin_mode_enable	使能 USART LIN 模式 (USARTx, x=0..5)
usart_lin_mode_disable	失能 USART LIN 模式 (USARTx, x=0..5)
usart_lin_break_detection_length_config	配置 USART LIN 模式中断帧长度 (USARTx, x=0..5)
usart_halfduplex_enable	使能 USART 半双工模式 (USARTx, x=0..5)
usart_halfduplex_disable	失能 USART 半双工模式 (USARTx, x=0..5)
usart_synchronous_clock_enable	在 USART 同步通讯模式下使能 CK 引脚 (USARTx, x=0..5)
usart_synchronous_clock_disable	在 USART 同步通讯模式下失能 CK 引脚 (USARTx, x=0..5)
usart_synchronous_clock_config	配置 USART 同步通讯模式参数 (USARTx, x=0..5)
usart_guard_time_config	在 USART 智能卡模式下配置保护时间值 (USARTx, x=0..5)
usart_smartcard_mode_enable	使能 USART 智能卡模式 (USARTx, x=0..5)
usart_smartcard_mode_disable	失能 USART 智能卡模式 (USARTx, x=0..5)
usart_smartcard_mode_nack_enable	在 USART 智能卡模式下使能 NACK (USARTx, x=0..5)
usart_smartcard_mode_nack_disable	在 USART 智能卡模式下失能 NACK (USARTx, x=0..5)
usart_smartcard_autoretry_config	配置智能卡自动重试次数 (USARTx, x=0..5)
usart_block_length_config	配置智能卡 T=1 的接收时块的长度 (USARTx, x=0..5)
usart_irda_mode_enable	使能 USART 串行红外编解码功能模块 (USARTx, x=0..5)
usart_irda_mode_disable	失能 USART 串行红外编解码功能模块 (USARTx, x=0..5)
usart_prescaler_config	在 USART IrDA 低功耗模式下配置外设时钟分频系数 (USARTx, x=0..5)
usart_irda_lowpower_config	配置 USART IrDA 低功耗模式 (USARTx, x=0..5)
usart_dma_receive_config	配置 USART DMA 接收功能 (USARTx, x=0..5)
usart_dma_transmit_config	配置 USART DMA 发送功能 (USARTx, x=0..5)
usart_hardware_flow_rts_config	配置 USART RTS 硬件控制流 (USARTx, x=0..4)
usart_hardware_flow_cts_config	配置 USART CTS 硬件控制流 (USARTx, x=0..4)
usart_data_first_config	配置数据传输时低位在前或高位在前 (USARTx, x=0..4)
usart_invert_config	配置 USART 反转功能 (USARTx, x=0..4)
usart_address_config	在地址掩码唤醒模式下配置 USART 地址 (USARTx, x=0..4)
usart_send_break	配置 USART 发送断开帧 (USARTx, x=0..4)
usart_collision_detected_interrupt_enable	使能 USART 冲突检测中断 (USARTx, x=0..4)
usart_collision_detected_interrupt_disable	失能 USART 冲突检测中断 (USARTx, x=0..4)
usart_collision_detection_enable	使能 USART 冲突检测 (USARTx, x=0..4)
usart_collision_detection_disable	失能 USART 冲突检测 (USARTx, x=0..4)
usart_flag_get	获取 USART 状态寄存器标志位 (USARTx, x=0..4)
usart_flag_clear	清除 USART 状态寄存器标志位 (USARTx, x=0..4)
usart_interrupt_enable	使能 USART 中断 (USARTx, x=0..4)
usart_interrupt_disable	失能 USART 中断 (USARTx, x=0..4)

库函数名称	库函数描述
usart_interrupt_flag_get	获取 USART 中断标志位状态 (USARTx, x=0..4)
usart_interrupt_flag_clear	清除 USART 中断标志位状态 (USARTx, x=0..4)
usart5_data_first_config	配置数据传输时低位在前或高位在前 (USARTx, x=5)
usart5_invert_config	配置 USART 反转功能 (USARTx, x=5)
usart5_overrun_enable	使能 USART 溢出禁止功能 (USARTx, x=5)
usart5_overrun_disable	失能 USART 溢出禁止功能 (USARTx, x=5)
usart5_address_config	在地址掩码唤醒模式下配置 USART 地址 (USARTx, x=5)
usart5_address_detection_mode_config	配置 USART 地址检测模式 (USARTx, x=5)
usart5_smartcard_mode_early_nack_enable	使能 USART 智能卡模式提前 NACK (USARTx, x=5)
usart5_smartcard_mode_early_nack_disable	失能 USART 智能卡模式提前 NACK (USARTx, x=5)
usart5_hardware_flow_coherence_config	配置硬件流控兼容模式 (USARTx, x=5)
usart5_rs485_driver_enable	使能 USART rs485 驱动 (USARTx, x=5)
usart5_rs485_driver_disable	失能 USART rs485 驱动 (USARTx, x=5)
usart5_driver_asserttime_config	配置 USART 驱动使能置位时间 (USARTx, x=5)
usart5_driver_deasserttime_config	配置 USART 驱动使能置低时间 (USARTx, x=5)
usart5_depolarity_config	配置 USART 驱动使能极性模式 (USARTx, x=5)
usart5_reception_error_dma_enable	USART 接收错误时使能 DMA (USARTx, x=5)
usart5_reception_error_dma_disable	USART 接收错误时失能 DMA (USARTx, x=5)
usart5_wakeup_enable	使能 USART 唤醒 (USARTx, x=5)
usart5_wakeup_disable	失能 USART 唤醒 (USARTx, x=5)
usart5_wakeup_mode_config	配置 USART 唤醒模式 (USARTx, x=5)
usart5_receive_fifo_enable	使能接收 FIFO (USARTx, x=5)
usart5_receive_fifo_disable	失能接收 FIFO (USARTx, x=5)
usart5_receive_fifo_counter_number	读取接收 FIFO 计数器的值 (USARTx, x=5)
usart5_flag_get	得到 STAT/RFC5 寄存器中的标志 (USARTx, x=5)
usart5_flag_clear	清除 USART 状态 (USARTx, x=5)
usart5_interrupt_enable	使能 USART 中断 (USARTx, x=5)
usart5_interrupt_disable	失能 USART 中断 (USARTx, x=5)
usart5_command_enable	使能 USART 请求 (USARTx, x=5)
usart5_interrupt_flag_get	得到 USART 中断和标志状态 (USARTx, x=5)
usart5_interrupt_flag_clear	清除 USART 中断标志位 (USARTx, x=5)

## 枚举类型 usart\_flag\_enum

表 3-983. 枚举类型 usart\_flag\_enum

成员名称	功能描述
USART_FLAG_CTS	CTS电平
USART_FLAG_LBD	LIN断开检测标志

成员名称	功能描述
USART_FLAG_TBE	发送数据寄存器空
USART_FLAG_TC	发送完成标志
USART_FLAG_RBNE	读数据缓冲区非空标志
USART_FLAG_IDLE	空闲线检测标志
USART_FLAG_ORERR	溢出错误标志
USART_FLAG_NERR	噪声错误标志
USART_FLAG_FERR	帧错误标志
USART_FLAG_PERR	校验错误标志
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_CD	冲突检测标志

### 枚举类型 `usart5_flag_enum`

表 3-984. 枚举类型 `usart5_flag_enum`

成员名称	功能描述
USART5_FLAG_REA	接收使能通知标志
USART5_FLAG_TEA	发送使能通知标志
USART5_FLAG_WU	从深度睡眠模式唤醒标志
USART5_FLAG_RWU	接收器从静默模式唤醒
USART5_FLAG_SB	断开信号发送标识
USART5_FLAG_AM	ADDR匹配标志
USART5_FLAG_BSY	忙标志
USART5_FLAG_EB	块结束标志
USART5_FLAG_RT	接收超时标志
USART5_FLAG_LBD	LIN断开检测标志
USART5_FLAG_TBE	发送数据寄存器空
USART5_FLAG_TC	发送完成标志
USART5_FLAG_RBNE	读数据缓冲区非空标志
USART5_FLAG_IDLE	空闲线检测标志
USART5_FLAG_ORERR	溢出错误标志
USART5_FLAG_NERR	噪声错误标志
USART5_FLAG_FERR	帧错误标志
USART5_FLAG_PERR	校验错误标志
USART5_FLAG_EPERR	校验错误超前检测标志
USART5_FLAG_RFFINT	接收FIFO满中断标志
USART5_FLAG_RFF	接收FIFO满标志
USART5_FLAG_RFE	接收FIFO空标志

## 枚举类型 `usart_interrupt_flag_enum`

表 3-985. 枚举类型 `usart_interrupt_flag_enum`

成员名称	功能描述
USART_INT_FLAG_PERR	校验错误中断标志
USART_INT_FLAG_TBE	发送数据寄存器空中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_RBNE	读数据缓冲区非空中断标志
USART_INT_FLAG_RBNE_ORE RR	溢出错误中断标志
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_CTS	CTS中断中断标志
USART_INT_FLAG_ERR_ORER R	溢出错误中断标志
USART_INT_FLAG_ERR_NERR	噪声错误中断标志
USART_INT_FLAG_ERR_FERR	帧错误中断标志
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_CD	冲突检测中断标志

## 枚举类型 `usart5_interrupt_flag_enum`

表 3-986. 枚举类型 `usart5_interrupt_flag_enum`

成员名称	功能描述
USART5_INT_FLAG_EB	块结束中断标志
USART5_INT_FLAG_RT	接收超时中断标志
USART5_INT_FLAG_AM	ADDR匹配中断标志
USART5_INT_FLAG_PERR	校验错误中断标志
USART5_INT_FLAG_TBE	发送数据寄存器空中断标志
USART5_INT_FLAG_TC	发送完成中断标志
USART5_INT_FLAG_RBNE	读数据缓冲区非空中断标志
USART5_INT_FLAG_RBNE_ORE RR	溢出错误中断标志
USART5_INT_FLAG_IDLE	空闲线检测中断标志
USART5_INT_FLAG_LBD	LIN断开检测中断标志
USART5_INT_FLAG_WU	从深度睡眠模式唤醒中断标志
USART5_INT_FLAG_ERR_NERR	噪声错误中断标志
USART5_INT_FLAG_ERR_ORER R	溢出错误中断标志
USART5_INT_FLAG_ERR_FERR	帧错误中断标志
USART5_INT_FLAG_ERR_RFF	接收FIFO满中断标志

### 枚举类型 `usart_interrupt_enum`

表 3-987. 枚举类型 `usart_interrupt_enum`

Member name	Function description
USART_INT_PERR	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	读数据缓冲区非空中断和过载错误中断使能
USART_INT_IDLE	IDLE线检测中断使能
USART_INT_LBD	LIN断开信号检测中断使能
USART_INT_CTS	CTS中断使能
USART_INT_ERR	错误中断使能
USART_INT_EB	块尾中断使能
USART_INT_RT	接收超时中断使能
USART_INT_CD	冲突检测中断使能

### 枚举类型 `usart5_interrupt_enum`

表 3-988. 枚举类型 `usart5_interrupt_enum`

Member name	Function description
USART5_INT_EB	块尾中断使能
USART5_INT_RT	接收超时中断使能
USART5_INT_AM	ADDR字符匹配中断使能
USART5_INT_PERR	校验错误中断使能
USART5_INT_TBE	发送寄存器空中断使能
USART5_INT_TC	发送完成中断使能
USART5_INT_RBNE	读数据缓冲区非空中断和过载错误中断使能
USART5_INT_IDLE	IDLE线检测中断使能
USART5_INT_LBD	LIN断开信号检测中断使能
USART5_INT_WU	从深度睡眠模式唤醒中断使能
USART5_INT_ERR	错误中断使能
USART5_INT_RFF	接收FIFO满中断使能

### 枚举类型 `usart_invert_enum`

表 3-989. 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

## 枚举类型 `usart5_invert_enum`

表 3-990. 枚举类型 `usart5_invert_enum`

成员名称	功能描述
USART5_DINV_ENABLE	数据位反转
USART5_DINV_DISABLE	数据位不反转
USART5_TXPIN_ENABLE	TX管脚电平反转
USART5_TXPIN_DISABLE	TX管脚电平不反转
USART5_RXPIN_ENABLE	RX管脚电平反转
USART5_RXPIN_DISABLE	RX管脚电平不反转
USART5_SWAP_ENABLE	交换TX/RX管脚
USART5_SWAP_DISABLE	不交换TX/RX管脚

## 函数 `usart_deinit`

函数`usart_deinit`描述见下表：

表 3-991. 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/UARTx
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/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
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 3-992. 函数 `usart_baudrate_set`

函数名称	<code>usart_baudrate_set</code>
函数原型	<code>void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);</code>
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/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
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 3-993. 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/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
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 USART 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 3-994. 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/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
输入参数{in}	
wlen	配置 USART 字长
USART_WL_8BIT	8 bits
USART_WL_9BIT	9 bits
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 3-995. 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/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
输入参数{in}	

stblen	配置 USART 停止位
USART_STB_1BIT	1 bit
USART_STB_0_5BIT	0.5 bit, 该位对 UARTx(x=3,4)无效
USART_STB_2BIT	2 bits
USART_STB_1_5BIT	1.5 bits, 该位对 UARTx(x=3,4)无效
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 3-996. 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/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
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 3-997. 函数 `usart_disable`

函数名称	<code>usart_disable</code>
函数原型	<code>void usart_disable(uint32_t usart_periph);</code>
功能描述	失能 USART
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设 USARTx/UARTx
<code>USARTx</code>	x=0,1,2,5
<code>UARTx</code>	x=3,4
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 3-998. 函数 `usart_transmit_config`

函数名称	<code>usart_transmit_config</code>
函数原型	<code>void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);</code>
功能描述	USART 发送器配置
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设 USARTx/UARTx
<code>USARTx</code>	x=0,1,2,5
<code>UARTx</code>	x=3,4
输入参数{in}	
<code>txconfig</code>	使能/失能 USART 发送器
<code>USART_TRANSMIT_ENABLE</code>	使能 USART 发送
<code>USART_TRANSMIT_DISABLE</code>	失能 USART 发送
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 3-999. 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/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
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\_oversample\_config

函数usart\_oversample\_config描述见下表:

表 3-1000. 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}	
<b>usart_periph</b>	外设 USARTx/UARTx
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
输入参数{in}	
<b>oversamp</b>	过采样值
<i>USART_OVSMOD_8</i>	8 位
<i>USART_OVSMOD_16</i>	16 位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 oversample mode */
```

```
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 3-1001. 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 obsm);
功能描述	配置 USART 采样位方法
先决条件	-
被调用函数	-
输入参数{in}	
<b>usart_periph</b>	外设 USARTx/UARTx
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
输入参数{in}	
<b>obsm</b>	采样位
<i>USART_OSB_1bit</i>	1 位
<i>USART_OSB_3bit</i>	3 位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 sample bit */
```

```
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 3-1002. 函数 usart\_receiver\_timeout\_enable**

函数名称	usart_receiver_timeout_enable
函数原型	void usart_receiver_timeout_enable(uint32_t usart_periph);
功能描述	使能 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=0,1,2,5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable receiver timeout of USART */
```

```
usart_receiver_timeout_enable(USART0);
```

### 函数 usart\_receiver\_timeout\_disable

函数usart\_receiver\_timeout\_disable描述见下表：

**表 3-1003. 函数 usart\_receiver\_timeout\_disable**

函数名称	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=0,1,2,5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable receiver timeout of USART */
```

```
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 3-1004. 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=0,1,2,5
输入参数{in}	
rtimeout	超时时间
0-0xFFFFF	超时时间值
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 3-1005. 函数 usart\_data\_transmit**

函数名称	usart_data_transmit
函数原型	void usart_data_transmit(uint32_t usart_periph, uint32_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/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
输入参数{in}	

<b>data</b>	发送的数据
0-0xFF	发送的数据
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* USART0 transmit data */
usart_data_transmit(USART0, 0xAA);
```

### 函数 usart\_data\_receive

函数usart\_data\_receive描述见下表：

**表 3-1006. 函数 usart\_data\_receive**

<b>函数名称</b>	usart_data_receive
<b>函数原型</b>	uint16_t usart_data_receive(uint32_t usart_periph);
<b>功能描述</b>	USART 接收数据功能
<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/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint32_t</b>	接收的数据（0-0xFF）

例如：

```
/* USART0 receive data */
uint16_t temp;
temp = usart_data_receive(USART0);
```

### 函数 usart\_mute\_mode\_enable

函数usart\_mute\_mode\_enable描述见下表：

**表 3-1007. 函数 usart\_mute\_mode\_enable**

<b>函数名称</b>	usart_mute_mode_enable
<b>函数原型</b>	void usart_mute_mode_enable(uint32_t usart_periph);
<b>功能描述</b>	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/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
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 3-1008. 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/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
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 3-1009. 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/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
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	地址掩码唤醒
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 3-1010. 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/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
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 3-1011. 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/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
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 3-1012. 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/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
输入参数{in}	
lflen	LIN 模式中断帧长度
USART_LBLEN_10 B	断开帧长度为 10 bits

<code>USART_LBLEN_11</code> <i>B</i>	断开帧长度为 11 bits
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 3-1013. 函数 `usart_halfduplex_enable`

函数名称	<code>usart_halfduplex_enable</code>
函数原型	<code>void usart_halfduplex_enable(uint32_t usart_periph);</code>
功能描述	使能 USART 半双工模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设 USARTx/UARTx
<code>USARTx</code>	x=0,1,2,5
<code>UARTx</code>	x=3,4
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 3-1014. 函数 `usart_halfduplex_disable`

函数名称	<code>usart_halfduplex_disable</code>
函数原型	<code>void usart_halfduplex_disable(uint32_t usart_periph);</code>
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}	
<b>usart_periph</b>	外设 USARTx/UARTx
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 half duplex mode*/
```

```
usart_halfduplex_disable(USART0);
```

### 函数 usart\_synchronous\_clock\_enable

函数usart\_synchronous\_clock\_enable描述见下表：

表 3-1015. 函数 usart\_synchronous\_clock\_enable

函数名称	usart_synchronous_clock_enable
函数原型	void usart_synchronous_clock_enable(uint32_t usart_periph);
功能描述	在 USART 同步通讯模式下使能 CK 引脚
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=0,1,2,5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 CK pin in synchronous mode */
```

```
usart_synchronous_clock_enable(USART0);
```

### 函数 usart\_synchronous\_clock\_disable

函数usart\_synchronous\_clock\_disable描述见下表：

表 3-1016. 函数 usart\_synchronous\_clock\_disable

函数名称	usart_synchronous_clock_disable
函数原型	void usart_synchronous_clock_disable(uint32_t usart_periph);
功能描述	在 USART 同步通讯模式下失能 CK 引脚
先决条件	-

被调用函数	-
输入参数{in}	
usart_periph	外设 USARTx
USARTx	x=0,1,2,5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 CK pin in synchronous mode */
```

```
usart_synchronous_clock_disable(USART0);
```

### 函数 usart\_synchronous\_clock\_config

函数 usart\_synchronous\_clock\_config 描述见下表：

表 3-1017. 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=0,1,2,5
输入参数{in}	
clen	CK 信号长度
USART_CLEN_NO 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 3-1018. 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 gaut);
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=0,1,2,5
输入参数{in}	
gaut	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, 0x0000 0055);
```

### 函数 usart\_smartcard\_mode\_enable

函数usart\_smartcard\_mode\_enable描述见下表:

表 3-1019. 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=0,1,2,5

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 3-1020. 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=0,1,2,5
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 3-1021. 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=0,1,2,5

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 3-1022. 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=0,1,2,5
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\_autoretry\_config

函数usart\_smartcard\_autoretry\_config描述见下表：

表 3-1023. 函数 usart\_smartcard\_autoretry\_config

函数名称	usart_smartcard_autoretry_config
函数原型	void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrtnum);
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}	
usart_periph	外设 USARTx

USARTx	x=0,1,2,5
输入参数{in}	
scrtnum	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 3-1024. 函数 usart\_block\_length\_config

函数名称	usart_block_length_config
函数原型	void usart_block_length_config(uint32_t usart_periph, uint32_t bl);
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}	
usart_periph	外设 USARTx
USARTx	x=0,1,2,5
输入参数{in}	
bl	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 3-1025. 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=0,1,2,5
UARTx	x=3,4
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 3-1026. 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/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
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 3-1027. 函数 usart\_prescaler\_config

函数名称	usart_prescaler_config
------	------------------------

函数原型	void usart_prescaler_config(uint32_t usart_periph, uint8_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/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
输入参数{in}	
psc	时钟分频系数
0-0xFF	时钟分频系数
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, 0x00);
```

### 函数 usart\_irda\_lowpower\_config

函数usart\_irda\_lowpower\_config描述见下表：

表 3-1028. 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/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
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_NOR MAL	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\_dma\_receive\_config

函数usart\_dma\_receive\_config描述见下表：

表 3-1029. 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 接收功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设 USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4
输入参数{in}	
dmacmd	使能/失能 DMA 接收功能
USART_RECEIVE_DMA_ENABLE	使能 DMA 接收功能
USART_RECEIVE_DMA_DISABLE	失能 DMA 接收功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 DMA enable for reception */

usart_dma_receive_config(USART0, USART_DENR_ENABLE);
```

### 函数 usart\_dma\_transmit\_config

函数usart\_dma\_transmit\_config描述见下表：

表 3-1030. 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 发送功能
先决条件	-
被调用函数	-

输入参数{in}	
<b>usart_periph</b>	外设 USARTx/UARTx
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
输入参数{in}	
<b>dmacmd</b>	使能/失能 DMA 发送功能
<i>USART_TRANSMIT_DMA_ENABLE</i>	使能 DMA 发送功能
<i>USART_TRANSMIT_DMA_DISABLE</i>	失能 DMA 发送功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 DMA enable for transmission */
```

```
usart_dma_transmit_config(USART0, USART_DENT_ENABLE);
```

### 函数 usart\_hardware\_flow\_rts\_config

函数usart\_hardware\_flow\_rts\_config描述见下表：

表 3-1031. 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}	
<b>usart_periph</b>	外设 USARTx
<i>USARTx</i>	x=0,1,2
输入参数{in}	
<b>rtsconfig</b>	使能/失能 RTS
<i>USART_RTS_ENABLE</i>	使能 RTS
<i>USART_RTS_DISABLE</i>	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 3-1032. 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_ENABLE	使能 CTS
USART_CTS_DISABLE	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\_data\_first\_config

函数usart\_data\_first\_config描述见下表：

表 3-1033. 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}	
<b>msbf</b>	数据传输时低位在前/高位在前
<i>USART_MSBF_LS</i> <i>B</i>	数据传输时低位在前
<i>USART_MSBF_MS</i> <i>B</i>	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 3-1034. 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}	
<b>usart_periph</b>	外设 USARTx
<i>USARTx</i>	x=0,1,2
输入参数{in}	
<b>invertpara</b>	参考 <a href="#">表 3-989. 枚举类型 usart_invert_enum</a>
<i>USART_DINV_ENA</i> <i>BLE</i>	数据位电平反转
<i>USART_DINV_DIS</i> <i>ABLE</i>	数据位电平不反转
<i>USART_TXPIN_EN</i> <i>ABLE</i>	TX 引脚电平反转
<i>USART_TXPIN_DIS</i> <i>ABLE</i>	TX 引脚电平不反转
<i>USART_RXPIN_EN</i> <i>ABLE</i>	RX 引脚电平反转
<i>USART_RXPIN_DI</i> <i>SABLE</i>	RX 引脚电平不反转
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* configure USART inversion */
```

```
usart_invert_config(USART0, USART_DINV_ENABLE);
```

### 函数 usart\_address\_config

函数usart\_address\_config描述见下表:

表 3-1035. 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	在地址掩码唤醒模式下配置 USART 地址
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设 USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
addr	USART/UART 地址(0x00000000 - 0x000000FF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure address of the USART0 */
```

```
usart_address_config(USART0, 0x00000000);
```

### 函数 usart\_send\_break

函数usart\_send\_break描述见下表:

表 3-1036. 函数 usart\_send\_break

函数名称	usart_send_break
函数原型	void usart_send_break(uint32_t usart_periph);
功能描述	配置 USART 发送断开帧
先决条件	-
被调用函数	-
输入参数{in}	

<b>usart_periph</b>	外设 USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* USART0 send break frame */
```

```
usart_send_break(USART0);
```

### 函数 usart\_flag\_get

函数usart\_flag\_get描述见下表：

表 3-1037. 函数 usart\_flag\_get

<b>函数名称</b>	usart_flag_get
<b>函数原型</b>	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
<b>功能描述</b>	获取 USART 状态寄存器标志位
<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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
<b>输入参数{in}</b>	
<b>flag</b>	USART 标志位，参考 <a href="#">表 3-983. 枚举类型 usart_flag_enum</a>
USART_FLAG_CTS	CTS 变化标志
USART_FLAG_LBD	LIN 断开检测标志
USART_FLAG_TBE	发送数据缓冲区空标志
USART_FLAG_TC	发送完成标志
USART_FLAG_RBNE	读数据缓冲区非空标志
USART_FLAG_IDLEF	空闲线检测标志
USART_FLAG_ORERR	溢出错误标志
USART_FLAG_NERR	噪声错误标志
USART_FLAG_FERR	帧错误标志
USART_FLAG_PERR	校验错误标志

RR	
USART_FLAG_BSY	忙状态标志
USART_FLAG_EB	块结束标志
USART_FLAG_RT	接收超时标志
USART_FLAG_CD	冲突检测标志
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 3-1038. 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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
flag	USART 标志位, 参考 <a href="#">表 3-983. 枚举类型 usart_flag_enum</a>
USART_FLAG_CTS	CTS 变化标志
USART_FLAG_LBD	LIN 断开检测标志
USART_FLAG_TC	发送完成
USART_FLAG_RBNE	读数据缓冲区非空
USART_FLAG_EB	块结束标志
USART_FLAG_RT	接收超时标志
USART_FLAG_CD	冲突检测标志
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 3-1039. 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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
interrupt	USART 中断, 参考 <a href="#">表 3-987. 枚举类型 usart_interrupt_enum</a>
USART_INT_PERR	校验错误中断
USART_INT_TBE	发送缓冲区空中断
USART_INT_TC	发送完成中断
USART_INT_RBNE	读数据缓冲区非空中断和过载错误中断
USART_INT_IDLE	IDLE 线检测中断
USART_INT_LBD	LIN 断开信号检测中断
USART_INT_ERR	错误中断
USART_INT_CTS	CTS 中断
USART_INT_RT	接收超时事件中断
USART_INT_EB	块结束事件中断
USART_INT_CD	冲突检测中断
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 3-1040. 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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
interrupt	USART 中断，参考 <a href="#">表 3-987. 枚举类型 usart_interrupt_enum</a>
USART_INT_PERR	校验错误中断
USART_INT_TBE	发送缓冲区空中断
USART_INT_TC	发送完成中断
USART_INT_RBNE	读数据缓冲区非空中断和过载错误中断
USART_INT_IDLE	IDLE 线检测中断
USART_INT_LBD	LIN 断开信号检测中断
USART_INT_ERR	错误中断
USART_INT_CTS	CTS 中断
USART_INT_RT	接收超时事件中断
USART_INT_EB	块结束事件中断
USART_INT_CD	冲突检测中断
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 3-1041. 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,

	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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
int_flag	USART 中断标志, 参考 <a href="#">表 3-985. 枚举类型 usart_interrupt_flag_enum</a>
USART_INT_FLAG_PERR	校验错误中断标志
USART_INT_FLAG_TBE	发送缓冲区空中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_RBNE	读数据缓冲区非空中断标志
USART_INT_FLAG_RBNE_ORERR	读数据缓冲区非空中断和溢出错误中断标志
USART_INT_FLAG_IDLE	IDLE 线检测中断标志
USART_INT_FLAG_LBD	LIN 断开检测中断标志
USART_INT_FLAG_CTS	CTS 中断标志
USART_INT_FLAG_ERR_ORERR	过载错误中断标志
USART_INT_FLAG_ERR_NERR	噪声错误中断标志
USART_INT_FLAG_ERR_FERR	帧错误中断标志
USART_INT_FLAG_EB	块结束事件中断标志
USART_INT_FLAG_RT	超时事件中断标志
USART_INT_FLAG_CD	冲突检测中断标志
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 3-1042. 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, 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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
int_flag	USART 中断标志，参考 <a href="#">表 3-985. 枚举类型 usart_interrupt_flag_enum</a>
USART_INT_FLAG_CTS	CTS 变化中断标志
USART_INT_FLAG_LBD	LIN 断开检测中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_RBNE	读数据缓冲区非空中断标志
USART_INT_FLAG_EB	块结束事件中断标志
USART_INT_FLAG_RT	超时事件中断标志
USART_INT_FLAG_CD	冲突检测中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the USART0 interrupt flag */
```

```
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

### 函数 usart5\_data\_first\_config

函数usart5\_data\_first\_config描述见下表：

表 3-1043. 函数 usart5\_data\_first\_config

函数名称	usart5_data_first_config
函数原型	void usart5_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=5
输入参数{in}	
msbf	数据传输时低位在前/高位在前
USART5_MSBF_LSB	数据传输时低位在前
USART5_MSBF_MSB	数据传输时高位在前
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LSB of data first */
```

```
usart5_data_first_config(USART5, USART5_MSBF_LSB);
```

### 函数 usart5\_invert\_config

函数usart5\_invert\_config描述见下表：

表 3-1044. 函数 usart5\_invert\_config

函数名称	usart5_invert_config
函数原型	void usart5_invert_config(uint32_t usart_periph, usart5_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=5
输入参数{in}	
invertpara	参考 <a href="#">表3-990. 枚举类型usart5_invert_enum</a>
USART5_DINV_ENABLE	数据位电平反转
USART5_DINV_DISABLE	数据位电平不反转
USART5_TXPIN_ENABLE	TX引脚电平反转
USART5_TXPIN_DISABLE	TX引脚电平不反转
USART5_RXPIN_ENABLE	RX引脚电平反转
USART5_RXPIN_DISABLE	RX引脚电平不反转
USART5_SWAP_ENABLE	TX和RX管脚功能被交换
USART5_SWAP_DISABLE	TX和RX管脚功能不被交换
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART5 inversion */
usart5_invert_config(USART5, USART5_DINV_ENABLE);
```

### 函数 usart5\_overrun\_enable

函数usart5\_overrun\_enable描述见下表：

表 3-1045. 函数 usart5\_overrun\_enable

函数名称	usart5_overrun_enable
函数原型	void usart5_overrun_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=5
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable USART5 overrun */
```

```
usart5_overrun_enable (USART5);
```

### 函数 usart5\_overrun\_disable

函数usart5\_overrun\_disable描述见下表：

表 3-1046. 函数 usart5\_overrun\_disable

函数名称	usart5_overrun_disable
函数原型	void usart5_overrun_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=5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART5 overrun */
```

```
usart5_overrun_disable (USART5);
```

### 函数 usart5\_address\_config

函数usart5\_address\_config描述见下表：

表 3-1047. 函数 usart5\_address\_config

函数名称	usart5_address_config
函数原型	void usart5_address_config(uint32_t usart_periph, uint8_t addr);
功能描述	在地址掩码唤醒模式下配置USART地址
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=5
输入参数{in}	
addr	USART地址(0x00000000-0x000000FF)

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure address of the USART5 */
usart5_address_config(USART5, 0x00000000);
```

### 函数 usart5\_address\_detection\_mode\_config

函数usart5\_address\_detection\_mode\_config描述见下表：

表 3-1048. 函数 usart5\_address\_detection\_mode\_config

函数名称	usart5_address_detection_mode_config
函数原型	void usart5_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}	
usart_periph	外设USARTx
USARTx	x=5
输入参数{in}	
addmod	地址检测模式
USART5_ADDM_4BIT	4位地址检测
USART5_ADDM_FULLBIT	全位地址检测
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*configure address detection mode */
usart5_address_detection_mode_config(USART5, USART5_ADDM_4BIT);
```

### 函数 usart5\_smartcard\_mode\_early\_nack\_enable

函数usart5\_smartcard\_mode\_early\_nack\_enable描述见下表：

表 3-1049. 函数 usart5\_smartcard\_mode\_early\_nack\_enable

函数名称	usart5_smartcard_mode_early_nack_enable
------	---

函数原型	void usart5_smartcard_mode_early_nack_enable (uint32_t usart_periph);
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}	
usart_periph	外设USARTx
USARTx	x=5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART5 early NACK in smartcard mode */
```

```
usart5_smartcard_mode_early_nack_enable (USART5);
```

### 函数 usart5\_smartcard\_mode\_early\_nack\_disable

函数usart5\_smartcard\_mode\_early\_nack\_disable描述见下表：

表 3-1050. 函数 usart5\_smartcard\_mode\_early\_nack\_disable

函数名称	usart5_smartcard_mode_early_nack_disable
函数原型	void usart5_smartcard_mode_early_nack_disable (uint32_t usart_periph);
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}	
usart_periph	外设USARTx
USARTx	x=5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART5 early NACK in smartcard mode */
```

```
usart5_smartcard_mode_early_nack_disable(USART5);
```

### 函数 usart5\_reception\_error\_dma\_disable

函数usart5\_reception\_error\_dma\_disable描述见下表：

表 3-1051. 函数 usart5\_reception\_error\_dma\_disable

函数名称	usart5_reception_error_dma_disable
------	------------------------------------

函数原型	void usart5_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=5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA on reception error */
```

```
usart5_reception_error_dma_disable (USART5);
```

### 函数 usart5\_reception\_error\_dma\_enable

函数usart5\_reception\_error\_dma\_enable描述见下表：

表 3-1052. 函数 usart5\_reception\_error\_dma\_enable

函数名称	usart5_reception_error_dma_enable
函数原型	void usart5_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=5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA on reception error */
```

```
usart5_reception_error_dma_enable(USART5);
```

### 函数 usart5\_wakeup\_enable

函数usart5\_reception\_wakeup\_enable描述见下表：

表 3-1053. 函数 usart5\_wakeup\_enable

函数名称	usart5_wakeup_enable
------	----------------------

函数原型	void usart5_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=5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART5 wake up enable */
usart5_wakeup_enable(USART5);
```

### 函数 usart5\_wakeup\_disable

函数usart5\_reception\_wakeup\_disable描述见下表：

表 3-1054. 函数 usart5\_wakeup\_disable

函数名称	usart5_wakeup_disable
函数原型	void usart5_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}	
usart_periph	外设USARTx
USARTx	x=5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART5 wake up disable */
usart5_wakeup_disable(USART5);
```

### 函数 usart5\_wakeup\_mode\_config

函数usart5\_reception\_mode\_config描述见下表：

表 3-1055. 函数 usart5\_wakeup\_mode\_config

函数名称	usart5_wakeup_mode_config
------	---------------------------

函数原型	void usart5_wakeup_mode_config(uint32_t usart_periph, uint32_t wum);
功能描述	配置USART唤醒模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=5
输入参数{in}	
wum	唤醒模式
USART5_WUM_AD DR	WUF在地址匹配时置位
USART5_WUM_ST ARTB	WUF在检测到起始位时置位
USART5_WUM_RB NE	WUF在检测到RBNE时置位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART5 wake up mode */
```

```
usart5_wakeup_mode_config(USART5, USART5_WUM_ADDR);
```

### 函数 usart5\_receive\_fifo\_enable

函数usart5\_receive\_fifo\_enable描述见下表：

表 3-1056. 函数 usart5\_receive\_fifo\_enable

函数名称	usart5_receive_fifo_enable
函数原型	void usart5_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=5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable receive FIFO */
```

```
usart5_receive_fifo_enable (USART5);
```

### 函数 usart5\_receive\_fifo\_disable

函数usart5\_receive\_fifo\_disable描述见下表：

**表 3-1057. 函数 usart5\_receive\_fifo\_disable**

函数名称	usart5_receive_fifo_disable
函数原型	void usart5_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=5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable receive FIFO */
```

```
usart5_receive_fifo_disable(USART5);
```

### 函数 usart5\_receive\_fifo\_counter\_number

函数usart5\_receive\_fifo\_counter\_number描述见下表：

**表 3-1058. 函数 usart5\_receive\_fifo\_counter\_number**

函数名称	usart5_receive_fifo_counter_number
函数原型	uint8_t usart5_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=5
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 = usart5_receive_fifo_counter_number(USART5);
```

### 函数 usart5\_flag\_get

函数usart5\_flag\_get描述见下表:

**表 3-1059. 函数 usart5\_flag\_get**

函数名称	usart5_flag_get
函数原型	FlagStatus usart5_flag_get(uint32_t usart_periph, usart5_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=5
输入参数{in}	
flag	USART标志位, 参考 <a href="#">表3-984. 枚举类型usart5_flag_enum</a>
USART5_FLAG_PERR	校验错误标志
USART5_FLAG_FERF	帧错误标志
USART5_FLAG_NERF	噪声错误标志
USART5_FLAG_ORERR	溢出错误标志
USART5_FLAG_IDLE	空闲线检测标志
USART5_FLAG_RXNE	读数据缓冲区非空标志
USART5_FLAG_TC	发送完成标志
USART5_FLAG_TBE	发送数据缓冲区空标志
USART5_FLAG_LBD	LIN断开检测标志
USART5_FLAG_RTE	接收超时标志
USART5_FLAG_EB	块结束标志
USART5_FLAG_BSY	忙状态标志
USART5_FLAG_ADM	ADDR匹配标志
USART5_FLAG_SB	断开信号发送标识

USART5_FLAG_R WU	接收器从静默模式唤醒
USART5_FLAG_W U	从深度睡眠模式唤醒标志
USART5_FLAG_TE A	发送使能通知标志
USART5_FLAG_RE A	接收使能通知标志
USART5_FLAG_EP ERR	校验错误超前检测标志
USART5_FLAG_RF E	接收FIFO空标志
USART5_FLAG_RF F	接收FIFO满标志
USART5_FLAG_RF FINT	接收FIFO满中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get flag USART5 state */
```

```
FlagStatus status;
```

```
status = usart5_flag_get(USART5, USART5_FLAG_TBE);
```

### 函数 usart5\_flag\_clear

函数usart5\_flag\_clear描述见下表：

表 3-1060. 函数 usart5\_flag\_clear

函数名称	usart5_flag_clear
函数原型	void usart5_flag_clear(uint32_t usart_periph, usart5_flag_enum flag);
功能描述	清除USART状态位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=5
输入参数{in}	
flag	USART标志位，参考 <a href="#">表3-984. 枚举类型usart5_flag_enum</a>

USART5_FLAG_PERR	校验错误标志
USART5_FLAG_FERR	帧错误标志
USART5_FLAG_NERR	噪声错误标志
USART5_FLAG_ORERR	溢出错误标志
USART5_FLAG_IDLE	空闲线检测标志
USART5_FLAG_TC	发送完成标志
USART5_FLAG_LBD	LIN断开检测标志
USART5_FLAG_RTO	接收超时标志
USART5_FLAG_EB	块结束标志
USART5_FLAG_AM	ADDR匹配标志
USART5_FLAG_WU	从深度睡眠模式唤醒标志
USART5_FLAG_EPERR	校验错误超前检测标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear USART5 flag */
```

```
usart5_flag_clear(USART5, USART5_FLAG_TC);
```

### 函数 usart5\_interrupt\_enable

函数usart5\_interrupt\_enable描述见下表：

表 3-1061. 函数 usart5\_interrupt\_enable

函数名称	usart5_interrupt_enable
函数原型	void usart5_interrupt_enable(uint32_t usart_periph, usart5_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=5
输入参数{in}	
interrupt	USART中断, 参考 <a href="#">表3-988. 枚举类型usart5_interrupt_enum</a>
USART5_INT_IDLE	IDLE线检测中断
USART5_INT_RBNE	读数据缓冲区非空中断和过载错误中断
USART5_INT_TC	发送完成中断
USART5_INT_TBE	发送缓冲区空中断
USART5_INT_PERR	校验错误中断
USART5_INT_AM	ADDR匹配中断
USART5_INT_RT	接收超时事件中断
USART5_INT_EB	块结束事件中断
USART5_INT_LBD	LIN断开信号检测中断
USART5_INT_ERR	错误中断
USART5_INT_WU	从深度睡眠模式唤醒中断
USART5_INT_RFF	接收FIFO满中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable USART5 TBE interrupt */
```

```
usart5_interrupt_enable(USART5, USART5_INT_TBE);
```

### 函数 usart5\_interrupt\_disable

函数usart5\_interrupt\_disable描述见下表:

表 3-1062. 函数 usart5\_interrupt\_disable

函数名称	usart5_interrupt_disable
函数原型	void usart5_interrupt_disable(uint32_t usart_periph, usart5_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=5
输入参数{in}	
interrupt	USART中断, 参考 <a href="#">表3-988. 枚举类型usart5_interrupt_enum</a>
USART5_INT_IDLE	IDLE线检测中断

USART5_INT_RBN E	读数据缓冲区非空中断和过载错误中断
USART5_INT_TC	发送完成中断
USART5_INT_TBE	发送缓冲区空中断
USART5_INT_PER R	校验错误中断
USART5_INT_AM	ADDR匹配中断
USART5_INT_RT	接收超时事件中断
USART5_INT_EB	块结束事件中断
USART5_INT_LBD	LIN断开信号检测中断
USART5_INT_ERR	错误中断
USART5_INT_WU	从深度睡眠模式唤醒中断
USART5_INT_RFF	接收FIFO满中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART5 TBE interrupt */
```

```
usart5_interrupt_disable(USART5, USART5_INT_TBE);
```

### 函数 usart5\_command\_enable

函数usart5\_command\_enable描述见下表：

表 3-1063. 函数 usart5\_command\_enable

函数名称	usart5_command_enable
函数原型	void usart5_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=5
输入参数{in}	
cmdtype	请求类型
USART5_CMD_SB KCMD	发送断开帧请求
USART5_CMD_MM CMD	静模式请求
USART5_CMD_RX FCMD	接收数据清空请求

USART5_CMD_TX FCMD	发送数据清空请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART5 command */
```

```
usart5_command_enable(USART5, USART5_CMD_SBKCMD);
```

### 函数 usart5\_interrupt\_flag\_get

函数usart5\_interrupt\_flag\_get描述见下表：

表 3-1064. 函数 usart5\_interrupt\_flag\_get

函数名称	usart5_interrupt_flag_get
函数原型	FlagStatus usart5_interrupt_flag_get(uint32_t usart_periph, usart5_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=5
输入参数{in}	
int_flag	USART中断标志，参考 <a href="#">表3-986. 枚举类型usart5_interrupt_flag_enum</a>
USART5_INT_FLG G_EB	块结束事件中断标志
USART5_INT_FLG G_RT	超时事件中断标志
USART5_INT_FLG G_AM	ADDR匹配中断标志
USART5_INT_FLG G_PERR	校验错误中断标志
USART5_INT_FLG G_TBE	发送缓冲区空中断标志
USART5_INT_FLG G_TC	发送完成中断标志
USART5_INT_FLG G_RBNE	读数据缓冲区非空中断标志
USART5_INT_FLG G_RBNE_ORERR	读数据缓冲区非空中断和溢出错误中断标志

USART5_INT_FLAG_IDLE	IDLE线检测中断标志
USART5_INT_FLAG_LBD	LIN断开检测中断标志
USART5_INT_FLAG_WU	从深度睡眠模式唤醒中断标志
USART5_INT_FLAG_ERR_NERR	噪声错误中断标志
USART5_INT_FLAG_ERR_ORERR	过载错误中断标志
USART5_INT_FLAG_ERR_FERR	帧错误中断标志
USART5_INT_FLAG_RFF	接收FIFO满中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get the USART5 interrupt flag status */
```

```
FlagStatus status;
```

```
status = usart5_interrupt_flag_get(USART5, USART5_INT_FLAG_RBNE);
```

### 函数 usart5\_interrupt\_flag\_clear

函数usart5\_interrupt\_flag\_clear描述见下表:

表 3-1065. 函数 usart5\_interrupt\_flag\_clear

函数名称	usart5_interrupt_flag_clear
函数原型	void usart5_interrupt_flag_clear(uint32_t usart_periph, usart5_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=5
输入参数{in}	
Int_flag	USART中断标志, 参考 <a href="#">表3-986. 枚举类型usart5_interrupt_flag_enum</a>
USART5_INT_FLAG_PERR	校验错误中断标志
USART5_INT_FLAG_FERR	帧错误中断标志

<code>G_ERR_FERR</code>	
<code>USART5_INT_FLG</code> <code>G_ERR_NERR</code>	噪声错误中断标志
<code>USART5_INT_FLG</code> <code>G_RBNE_ORERR</code>	读数据缓冲区非空中断和溢出错误中断标志
<code>USART5_INT_FLG</code> <code>G_ERR_ORERR</code>	过载错误中断标志
<code>USART5_INT_FLG</code> <code>G_IDLE</code>	IDLE线检测中断标志
<code>USART5_INT_FLG</code> <code>G_TC</code>	发送完成中断标志
<code>USART5_INT_FLG</code> <code>G_LBD</code>	LIN断开检测中断标志
<code>USART5_INT_FLG</code> <code>G_RT</code>	接收超时事件中断标志
<code>USART5_INT_FLG</code> <code>G_EB</code>	块结束事件中断标志
<code>USART5_INT_FLG</code> <code>G_AM</code>	ADDR匹配中断标志
<code>USART5_INT_FLG</code> <code>G_WU</code>	从深度睡眠模式唤醒中断标志
<code>USART5_INT_FLG</code> <code>G_RFF</code>	接收FIFO满中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the USART5 interrupt flag */
```

```
usart5_interrupt_flag_clear(USART5, USART5_INT_FLAG_TC);
```

## 3.29. 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 [3.29.1](#) 描述了 WWDGT 的寄存器列表，章节 [3.29.2](#) 对 WWDGT 库函数进行说明。

### 3.29.1. 外设寄存器说明

WWDGT 寄存器列表如下表所示：

表 3-1066. 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

### 3.29.2. 外设库函数说明

WWDGT库函数列表如下表所示：

表 3-1067. 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 3-1068. 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 window watchdog timer configuration */
```

```
wwdgt_deinit( );
```

#### 函数 wwdgt\_enable

函数wwdgt\_enable描述见下表：

表 3-1069. 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 window watchdog timer counter */
```

```
wwdgt_enable( );
```

### 函数 wwdgt\_counter\_update

函数wwdgt\_counter\_update描述见下表：

表 3-1070. 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	0x00 - 0x7F
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 3-1071. 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	0x00 - 0x7F
输入参数{in}	
window	0x00 - 0x7F
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_DIV8	WWDGT计数器时钟为 (PCLK/4096) /8
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 3-1072. 函数 wwdgt\_interrupt\_enable

函数名称	wwdgt_interrupt_enable
函数原型	void wwdgt_interrupt_enable(void);
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 3-1073. 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 ( );

if(status == RESET)
{
...
}else
{
...
}
```

### 函数 wwdgt\_flag\_clear

函数wwdgt\_flag\_clear描述见下表:

表 3-1074. 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	2024 年 06 月 04 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.