

**GigaDevice Semiconductor Inc.**

**GD32F4xx  
ARM® Cortex™-M4 32-bit MCU**

**固件库  
使用指南**

1.0 版本

(2018 年 12 月)

# 目录

目录 .....	2
图索引 .....	5
表索引 .....	6
<b>1. 介绍 .....</b>	<b>29</b>
<b>1.1. 文档和固件库规则 .....</b>	<b>29</b>
1.1.1. 外设缩写 .....	29
1.1.2. 命名规则 .....	30
<b>2. 固件库概述 .....</b>	<b>31</b>
<b>2.1. 文件组织结构 .....</b>	<b>31</b>
2.1.1. Examples 文件夹 .....	32
2.1.2. Firmware 文件夹 .....	32
2.1.3. Project 文件夹 .....	32
2.1.4. Template 文件夹 .....	32
2.1.5. Utilities 文件夹 .....	35
<b>2.2. 固件库文件描述 .....</b>	<b>36</b>
<b>3. 外设固件库 .....</b>	<b>37</b>
<b>3.1. 外设固件库概述 .....</b>	<b>37</b>
<b>3.2. ADC .....</b>	<b>37</b>
3.2.1. 外设寄存器描述 .....	37
3.2.2. 外设库函数说明 .....	38
<b>3.3. CAN .....</b>	<b>83</b>
3.3.1. 外设寄存器说明 .....	83
3.3.2. 外设库函数说明 .....	84
<b>3.4. CRC .....</b>	<b>109</b>
3.4.1. 外设寄存器说明 .....	109
3.4.2. 外设库函数说明 .....	109
<b>3.5. CTC .....</b>	<b>113</b>
3.5.1. 外设寄存器说明 .....	113
3.5.2. 外设库函数说明 .....	114
<b>3.6. DAC .....</b>	<b>131</b>
3.6.1. 外设寄存器说明 .....	131
3.6.2. 外设库函数说明 .....	132
<b>3.7. DBG .....</b>	<b>151</b>
3.7.1. 外设寄存器说明 .....	151

---

3.7.2. 外设库函数说明 .....	151
<b>3.8. DCI .....</b>	<b>158</b>
3.8.1. 外设寄存器说明 .....	159
3.8.2. 外设库函数说明 .....	159
<b>3.9. DMA .....</b>	<b>175</b>
3.9.1. 外设寄存器说明 .....	176
3.9.2. 外设库函数说明 .....	176
<b>3.10. ENET .....</b>	<b>207</b>
3.10.1. 外设寄存器描述 .....	207
3.10.2. 外设库函数说明 .....	210
<b>3.11. EXMC .....</b>	<b>304</b>
3.11.1. 外设寄存器说明 .....	304
3.11.2. 外设库函数说明 .....	305
<b>3.12. EXTI.....</b>	<b>340</b>
3.12.1. 外设寄存器说明 .....	341
3.12.2. 外设库函数说明 .....	341
<b>3.13. FMC .....</b>	<b>350</b>
3.13.1. 外设寄存器说明 .....	350
3.13.2. 外设库函数说明 .....	350
<b>3.14. FWDGT.....</b>	<b>350</b>
3.14.1. 外设寄存器说明 .....	375
3.14.2. 外设库函数说明 .....	375
<b>3.15. GPIO .....</b>	<b>380</b>
3.15.1. 外设寄存器说明 .....	380
3.15.2. 外设库函数说明 .....	错误!未定义书签。
<b>3.16. I2C .....</b>	<b>391</b>
3.16.1. 外设寄存器说明 .....	391
3.16.2. 外设库函数说明 .....	392
<b>3.17. IPA .....</b>	<b>425</b>
3.17.1. 外设寄存器说明 .....	425
3.17.2. 外设库函数说明 .....	426
<b>3.18. IREF.....</b>	<b>444</b>
3.18.1. 外设寄存器说明 .....	444
3.18.2. 外设库函数说明 .....	444
<b>3.19. MISC .....</b>	<b>448</b>
3.19.1. 外设寄存器说明 .....	448
3.19.2. 外设库函数说明 .....	450
<b>3.20. PMU.....</b>	<b>458</b>

---

3.20.1. 外设寄存器说明 .....	459
3.20.2. 外设库函数说明 .....	459
<b>3.21. RCU .....</b>	<b>474</b>
3.21.1. 外设寄存器说明 .....	474
3.21.2. 外设库函数说明 .....	475
<b>3.22. RTC .....</b>	<b>521</b>
3.22.1. 外设寄存器描述 .....	521
3.22.2. 外设库函数描述 .....	522
<b>3.23. SDIO .....</b>	<b>556</b>
3.23.1. 外设寄存器说明 .....	557
3.23.2. 外设库函数说明 .....	557
<b>3.24. SPI .....</b>	<b>597</b>
3.24.1. 外设寄存器说明 .....	597
3.24.2. 外设库函数说明 .....	598
<b>3.25. SYSCFG .....</b>	<b>624</b>
3.25.1. 外设寄存器说明 .....	624
3.25.2. 外设库函数说明 .....	625
<b>3.26. TIMER .....</b>	<b>631</b>
3.26.1. 外设寄存器说明 .....	631
3.26.2. 外设库函数说明 .....	632
<b>3.27. TLI .....</b>	<b>690</b>
3.27.1. 外设寄存器说明 .....	690
3.27.2. 外设库函数说明 .....	691
<b>3.28. TRNG .....</b>	<b>714</b>
3.28.1. 外设寄存器说明 .....	714
3.28.2. 外设库函数说明 .....	714
<b>3.29. USART .....</b>	<b>721</b>
3.29.1. 外设寄存器说明 .....	721
3.29.2. 外设库函数说明 .....	721
<b>3.30. WWDGT .....</b>	<b>758</b>
3.30.1. 外设寄存器说明 .....	758
3.30.2. 外设库函数说明 .....	759
<b>4. 版本历史 .....</b>	<b>765</b>

## 图索引

图 2-1. GD32F4xx 固件库文件组织结构 .....	31
图 2-2. 选择外设例程文件 .....	33
图 2-3. 拷贝外设例程文件 .....	34
图 2-4. 打开工程文件 .....	34
图 2-5. 配置工程文件 .....	35
图 2-6. 编译调试下载 .....	35

# 表索引

表 1-1. 外设缩写.....	29
表 2-1. 固件函数库文件描述.....	36
表 3-1. 外设固件库函数描述格式.....	37
表 3-2. ADC 寄存器 .....	37
表 3-3. ADC 库函数 .....	38
表 3-4. 函数 adc_deinit .....	40
表 3-5. 函数 adc_clock_config .....	41
表 3-6. 函数 adc_special_function_config .....	42
表 3-7. 函数 adc_data_alignment_config .....	43
表 3-8. 函数 adc_enable.....	44
表 3-9. 函数 adc_disable.....	45
表 3-10. 函数 adc_calibration_enable .....	45
表 3-11. 函数 adc_channel_16_to_18 .....	46
表 3-12. 函数 adc_resolution_config.....	47
表 3-13. 函数 adc_oversample_mode_config.....	48
表 3-14. 函数 adc_oversample_mode_enable .....	50
表 3-15. 函数 adc_oversample_mode_disable .....	51
表 3-16. 函数 adc_dma_mode_enable .....	51
表 3-17. 函数 adc_dma_mode_disable .....	52
表 3-18. 函数 adc_dma_request_after_last_enable.....	53
表 3-19. 函数 adc_dma_request_after_last_disable.....	53
表 3-20. 函数 adc_discontinuous_mode_config.....	54
表 3-21. 函数 adc_channel_length_config .....	55
表 3-22. 函数 adc_regular_channel_config.....	56
表 3-23. 函数 adc_inserted_channel_config .....	57
表 3-24. 函数 adc_inserted_channel_offset_config .....	59
表 3-25. 函数 adc_external_trigger_source_config .....	60
表 3-26. 函数 adc_external_trigger_config .....	63
表 3-27. 函数 adc_software_trigger_enable .....	64
表 3-28. 函数 adc_end_of_conversion_config .....	65
表 3-29. 函数 adc_regular_data_read.....	66
表 3-30. 函数 adc_inserted_data_read.....	66
表 3-31. 函数 adc_watchdog_single_channel_disable .....	67
表 3-32. 函数 adc_watchdog_single_channel_enable .....	68
表 3-33. 函数 adc_watchdog_group_channel_enable .....	69
表 3-34. 函数 adc_watchdog_disable.....	70
表 3-35. 函数 adc_watchdog_threshold_config .....	70
表 3-36. 函数 adc_flag_get.....	71
表 3-37. 函数 adc_flag_clear.....	72
表 3-38. 函数 adc_regular_software_startconv_flag_get .....	73

---

表 3-39. 函数 adc_inserted_software_startconv_flag_get .....	74
表 3-40. 函数 adc_interrupt_flag_get .....	74
表 3-41. 函数 adc_interrupt_flag_clear .....	75
表 3-42. 函数 adc_interrupt_enable .....	76
表 3-43. 函数 adc_interrupt_disable .....	77
表 3-44. 函数 adc_sync_mode_config .....	78
表 3-45. 函数 adc_sync_delay_config .....	79
表 3-46. 函数 adc_sync_dma_config .....	80
表 3-47. 函数 adc_sync_dma_request_after_last_enable .....	81
表 3-48. 函数 adc_sync_dma_request_after_last_disable .....	81
表 3-49. 函数 adc_sync_regular_data_read .....	82
表 3-50. CAN 寄存器 .....	83
表 3-51. CAN 库函数 .....	84
表 3-52. 结构体 can_parameter_struct .....	85
表 3-53. 结构体 can_trasnmit_message_struct .....	85
表 3-54. 结构体 can_receive_message_struct .....	86
表 3-55. 结构体 can_filter_parameter_struct .....	86
表 3-56. 函数 can_deinit .....	87
表 3-57. 函数 can_struct_para_init .....	87
表 3-58. 函数 can_init .....	88
表 3-59. 函数 can_filter_init .....	89
表 3-60. 函数 can1_filter_start_bank .....	90
表 3-61. 函数 can_debug_freeze_enable .....	90
表 3-62. 函数 can_debug_freeze_disable .....	91
表 3-63. 函数 can_time_trigger_mode_enable .....	91
表 3-64. 函数 can_time_trigger_mode_disable .....	92
表 3-65. 函数 can_message_transmit .....	93
表 3-66. 函数 can_transmit_states .....	94
表 3-67. 函数 can_transmission_stop .....	94
表 3-68. 函数 can_message_receive .....	95
表 3-69. 函数 can_fifo_release .....	96
表 3-70. 函数 can_receive_message_length_get .....	97
表 3-71. 函数 can_working_mode_set .....	98
表 3-72. 函数 can_wakeup .....	99
表 3-73. 函数 can_error_get .....	99
表 3-74. 函数 can_receive_error_number_get .....	100
表 3-75. 函数 can_transmit_error_number_get .....	101
表 3-76. 函数 can_interrupt_enable .....	104
表 3-77. 函数 can_interrupt_disable .....	105
表 3-78. 函数 can_flag_get .....	101
表 3-79. 函数 can_flag_clear .....	103
表 3-80. 函数 can_interrupt_flag_get .....	106
表 3-81. 函数 can_interrupt_flag_clear .....	107

---

表 3-82. CRC 寄存器 .....	109
表 3-83. CRC 库函数 .....	109
表 3-84. 函数 <code>crc_deinit</code> .....	109
表 3-85. 函数 <code>crc_data_register_reset</code> .....	110
表 3-86. 函数 <code>crc_data_register_read</code> .....	110
表 3-87. 函数 <code>crc_free_data_register_read</code> .....	111
表 3-88. 函数 <code>crc_free_data_register_write</code> .....	111
表 3-89. 函数 <code>crc_single_data_calculate</code> .....	112
表 3-90. 函数 <code>crc_block_data_calculate</code> .....	112
表 3-91. CTC 寄存器 .....	113
表 3-92. CTC 库函数 .....	114
表 3-93. 函数 <code>ctc_deinit</code> .....	115
表 3-94. 函数 <code>ctc_counter_enable</code> .....	115
表 3-95. 函数 <code>ctc_counter_disable</code> .....	116
表 3-96. 函数 <code>ctc_irc48m_trim_value_config</code> .....	116
表 3-97. 函数 <code>ctc_software_refsource_pulse_generate</code> .....	117
表 3-98. 函数 <code>ctc_hardware_trim_mode_config</code> .....	118
表 3-99. 函数 <code>ctc_refsource_polarity_config</code> .....	118
表 3-100. 函数 <code>ctc_usbsof_signal_select</code> .....	119
表 3-101. 函数 <code>ctc_refsource_signal_select</code> .....	120
表 3-102. 函数 <code>ctc_refsource_prescaler_config</code> .....	121
表 3-103. 函数 <code>ctc_clock_limit_value_config</code> .....	122
表 3-104. 函数 <code>ctc_counter_reload_value_config</code> .....	122
表 3-105. 函数 <code>ctc_counter_capture_value_read</code> .....	123
表 3-106. 函数 <code>ctc_counter_direction_read</code> .....	124
表 3-107. 函数 <code>ctc_counter_reload_value_read</code> .....	124
表 3-108. 函数 <code>ctc_irc48m_trim_value_read</code> .....	125
表 3-109. 函数 <code>ctc_interrupt_enable</code> .....	126
表 3-110. 函数 <code>ctc_interrupt_disable</code> .....	126
表 3-111. 函数 <code>ctc_interrupt_flag_get</code> .....	127
表 3-112. 函数 <code>ctc_interrupt_flag_clear</code> .....	128
表 3-113. 函数 <code>ctc_flag_get</code> .....	129
表 3-114. 函数 <code>ctc_flag_clear</code> .....	130
表 3-115. DAC 寄存器 .....	131
表 3-116. DAC 库函数 .....	132
表 3-117. 函数 <code>dac_deinit</code> .....	133
表 3-118. 函数 <code>dac_enable</code> .....	133
表 3-119. 函数 <code>dac_disable</code> .....	134
表 3-120. 函数 <code>dac_dma_enable</code> .....	134
表 3-121. 函数 <code>dac_dma_disable</code> .....	135
表 3-122. 函数 <code>dac_output_buffer_enable</code> .....	135
表 3-123. 函数 <code>dac_output_buffer_disable</code> .....	136
表 3-124. 函数 <code>dac_output_value_get</code> .....	136

---

表 3-125. 函数 dac_data_set .....	137
表 3-126. 函数 dac_trigger_enable .....	138
表 3-127. 函数 dac_trigger_disable .....	138
表 3-128. 函数 dac_trigger_source_config .....	139
表 3-129. 函数 dac_software_trigger_enable .....	140
表 3-130. 函数 dac_software_trigger_disable .....	140
表 3-131. 函数 dac_wave_mode_config .....	141
表 3-132. 函数 dac_wave_bit_width_config .....	141
表 3-133. 函数 dac_lfsr_noise_config .....	142
表 3-134. 函数 dac_triangle_noise_config .....	142
表 3-135. 函数 dac_concurrent_enable .....	143
表 3-136. 函数 dac_concurrent_disable .....	144
表 3-137. 函数 dac_concurrent_software_trigger_enable .....	144
表 3-138. 函数 dac_concurrent_software_trigger_disable .....	145
表 3-139. 函数 dac_concurrent_output_buffer_enable .....	145
表 3-140. 函数 dac_concurrent_output_buffer_disable .....	146
表 3-141. 函数 dac_concurrent_data_set .....	146
表 3-142. 函数 dac_concurrent_interrupt_enable .....	147
表 3-143. 函数 dac_concurrent_interrupt_disable .....	147
表 3-144. 函数 dac_interrupt_enable .....	149
表 3-145. 函数 dac_interrupt_disable .....	149
表 3-146. 函数 dac_flag_get .....	148
表 3-147. 函数 dac_flag_clear .....	148
表 3-148. 函数 dac_interrupt_flag_get .....	150
表 3-149. 函数 dac_interrupt_flag_clear .....	150
表 3-150. DBG 寄存器 .....	151
表 3-151. DBG 库函数 .....	151
表 3-152. 枚举类型 dbg_periph_enum .....	152
表 3-153. 函数 dbg_id_get .....	152
表 3-154. 函数 dbg_low_power_enable .....	153
表 3-155. 函数 dbg_low_power_disable .....	154
表 3-156. 函数 dbg_periph_enable .....	155
表 3-157. 函数 dbg_periph_disable .....	156
表 3-158. 函数 dbg_trace_pin_enable .....	156
表 3-159. 函数 dbg_trace_pin_disable .....	157
表 3-160. 函数 dbg_trace_pin_mode_set .....	158
表 3-161. DCI 寄存器 .....	159
表 3-162. DCI 库函数 .....	159
表 3-163. 结构体 dci_parameter_struct .....	160
表 3-164. 函数 dci_deinit .....	161
表 3-165. 函数 dci_init .....	161
表 3-166. 函数 dci_enable .....	162
表 3-167. 函数 dci_disable .....	163

---

表 3-168. 函数 dci_capture_enable.....	163
表 3-169. 函数 dci_capture_disable.....	164
表 3-170. 函数 dci_jpeg_enable .....	165
表 3-171. 函数 dci_jpeg_disable .....	165
表 3-172. 函数 dci_crop_window_enable .....	166
表 3-173. 函数 dci_crop_window_disable .....	166
表 3-174. 函数 dci_crop_window_config.....	167
表 3-175. 函数 dci_embedded_sync_enable.....	168
表 3-176. 函数 dci_embedded_sync_disable.....	168
表 3-177. 函数 dci_sync_codes_config .....	169
表 3-178. 函数 dci_sync_codes_unmask_config.....	170
表 3-179. 函数 dci_data_read .....	171
表 3-180. 函数 dci_flag_get.....	171
表 3-181. 函数 dci_interrupt_enable.....	172
表 3-182. 函数 dci_interrupt_disable.....	173
表 3-183. 函数 dci_interrupt_flag_get .....	174
表 3-184. 函数 dci_interrupt_flag_clear .....	175
表 3-185. DMA 寄存器 .....	176
表 3-186. DMA 库函数 .....	176
表 3-187. 结构体 dma_multi_data_parameter_struct .....	178
表 3-188. 结构体 dma_single_data_parameter_struct .....	178
表 3-189. 函数 dma_deinit .....	179
表 3-190. 函数 dma_single_data_mode_init .....	179
表 3-191. 函数 dma_multi_data_mode_init .....	180
表 3-192. 函数 dma_periph_address_config .....	181
表 3-193. 函数 dma_memory_address_config .....	182
表 3-194. 函数 dma_transfer_number_config.....	183
表 3-195. 函数 dma_transfer_number_get .....	184
表 3-196. 函数 dma_priority_config.....	185
表 3-197. 函数 dma_memory_burst_beats_config .....	186
表 3-198. 函数 dma_periph_burst_beats_config .....	187
表 3-199. 函数 dma_memory_width_config .....	188
表 3-200. 函数 dma_periph_width_config .....	189
表 3-201. 函数 dma_memory_address_generation_config .....	190
表 3-202. 函数 dma_peripheral_address_generation_config .....	191
表 3-203. 函数 dma_circulation_enable.....	192
表 3-204. 函数 dma_circulation_disable.....	193
表 3-205. 函数 dma_channel_enable.....	193
表 3-206. 函数 dma_channel_disable.....	194
表 3-207. 函数 dma_transfer_direction_config .....	195
表 3-208. 函数 dma_switch_buffer_mode_config .....	196
表 3-209. 函数 dma_using_memory_get .....	197
表 3-210. 函数 dma_channel_subperipheral_select.....	197

---

表 3-211. 函数 dma_flow_controller_config.....	198
表 3-212. 函数 dma_switch_buffer_mode_enable .....	199
表 3-213. 函数 dma_fifo_status_get .....	200
表 3-214. 函数 dma_flag_get.....	201
表 3-215. 函数 dma_flag_clear .....	202
表 3-216. 函数 dma_interrupt_flag_get.....	203
表 3-217. 函数 dma_interrupt_flag_clear.....	204
表 3-218. 函数 dma_interrupt_enable .....	205
表 3-219. 函数 dma_interrupt_disable .....	206
表 3-220. ENET 寄存器.....	207
表 3-221. ENET 库函数.....	210
表 3-222. 结构体 enet_initpara_struct.....	214
表 3-223. 结构体 enet_descriptors_struct.....	215
表 3-224. 结构体 enet_ptp_systime_struct.....	216
表 3-225. 函数 enet_deinit .....	216
表 3-226. 函数 enet_initpara_config .....	216
表 3-227. 函数 enet_init .....	221
表 3-228. 函数 enet_software_reset .....	222
表 3-229. 函数 enet_rxframe_size_get .....	223
表 3-230. 函数 enet_descriptors_chain_init.....	224
表 3-231. 函数 enet_descriptors_ring_init .....	224
表 3-232. 函数 enet_frame_receive.....	225
表 3-233. 函数 enet_frame_transmit.....	226
表 3-234. 函数 enet_transmit_checksum_config .....	226
表 3-235. 函数 enet_enable .....	228
表 3-236. 函数 enet_disable .....	228
表 3-237. 函数 enet_mac_address_set .....	229
表 3-238. 函数 enet_mac_address_get .....	230
表 3-239. 函数 enet_flag_get.....	231
表 3-240. 函数 enet_flag_clear .....	234
表 3-241. 函数 enet_interrupt_enable .....	235
表 3-242. 函数 enet_interrupt_disable .....	237
表 3-243. 函数 enet_interrupt_flag_get .....	239
表 3-244. 函数 enet_interrupt_flag_clear .....	241
表 3-245. 函数 enet_tx_enable.....	243
表 3-246. 函数 enet_tx_disable.....	244
表 3-247. 函数 enet_rx_enable .....	244
表 3-248. 函数 enet_rx_disable .....	245
表 3-249. 函数 enet_registers_get .....	245
表 3-250. 函数 enet_debug_status_get .....	246
表 3-251. 函数 enet_address_filter_enable .....	248
表 3-252. 函数 enet_address_filter_disable .....	248
表 3-253. 函数 enet_address_filter_config .....	249

---

表 3-254. 函数 enet_phy_config .....	251
表 3-255. 函数 enet_phy_write_read .....	251
表 3-256. 函数 enet_phyloopback_enable.....	252
表 3-257. 函数 enet_phyloopback_disable.....	253
表 3-258. 函数 enet_forward_feature_enable .....	254
表 3-259. 函数 enet_forward_feature_disable .....	254
表 3-260. 函数 enet_filter_feature_enable .....	255
表 3-261. 函数 enet_filter_feature_disable .....	256
表 3-262. 函数 enet_pauseframe_generate .....	257
表 3-263. 函数 enet_pauseframe_detect_config .....	258
表 3-264. 函数 enet_pauseframe_config .....	259
表 3-265. 函数 enet_flowcontrol_threshold_config .....	260
表 3-266. 函数 enet_flowcontrol_feature_enable .....	262
表 3-267. 函数 enet_flowcontrol_feature_disable .....	262
表 3-268. 函数 enet_dmaprocess_state_get .....	263
表 3-269. 函数 enet_dmaprocess_resume .....	264
表 3-270. 函数 enet_rxprocess_check_recovery .....	265
表 3-271. 函数 enet_txfifo_flush.....	266
表 3-272. 函数 enet_current_desc_address_get.....	266
表 3-273. 函数 enet_desc_information_get.....	267
表 3-274. 函数 enet_missed_frame_counter_get.....	268
表 3-275. 函数 enet_desc_flag_get.....	269
表 3-276. 函数 enet_desc_flag_set .....	272
表 3-277. 函数 enet_desc_flag_clear.....	273
表 3-278. 函数 enet_rx_desc_immediate_receive_complete_interrupt .....	275
表 3-279. 函数 enet_rx_desc_delay_receive_complete_interrupt .....	276
表 3-280. 函数 enet_rxframe_drop .....	276
表 3-281. 函数 enet_dma_feature_enable .....	277
表 3-282. 函数 enet_dma_feature_disable .....	278
表 3-283. 函数 enet_rx_desc_enhanced_status_get.....	278
表 3-284. 函数 enet_desc_select_enhanced_mode.....	280
表 3-285. 函数 enet_ptp_enhanced_descriptors_chain_init.....	280
表 3-286. 函数 enet_ptp_enhanced_descriptors_ring_init.....	281
表 3-287. 函数 enet_ptpframe_receive_enhanced_mode.....	282
表 3-288. 函数 enet_ptpframe_transmit_enhanced_mode.....	283
表 3-289. 函数 enet_desc_select_normal_mode.....	283
表 3-290. 函数 enet_ptp_normal_descriptors_chain_init.....	284
表 3-291. 函数 enet_ptp_normal_descriptors_ring_init.....	285
表 3-292. 函数 enet_ptpframe_receive_normal_mode.....	286
表 3-293. 函数 enet_ptpframe_transmit_normal_mode.....	287
表 3-294. 函数 enet_wum_filter_register_pointer_reset .....	287
表 3-295. 函数 enet_wum_filter_config.....	288
表 3-296. 函数 enet_wum_feature_enable.....	289

---

表 3-297. 函数 enet_wum_feature_disable.....	290
表 3-298. 函数 enet_msc_counters_reset .....	290
表 3-299. 函数 enet_msc_feature_enable.....	291
表 3-300. 函数 enet_msc_feature_disable.....	292
表 3-301. 函数 enet_msc_counters_preset_config .....	293
表 3-302. 函数 enet_msc_counters_get.....	293
表 3-303. 函数 enet_ptp_feature_enable .....	294
表 3-304. 函数 enet_ptp_feature_disable .....	295
表 3-305. 函数 enet_ptp_timestamp_function_config.....	296
表 3-306. 函数 enet_ptp_subsecond_increment_config .....	298
表 3-307. 函数 enet_ptp_timestamp_addend_config .....	299
表 3-308. 函数 enet_ptp_timestamp_update_config .....	299
表 3-309. 函数 enet_ptp_expected_time_config .....	300
表 3-310. 函数 enet_ptp_system_time_get .....	301
表 3-311. 函数 enet_ptp_pps_output_frequency_config .....	302
表 3-312. 函数 enet_initpara_reset .....	303
表 3-313. EXMC 寄存器 .....	304
表 3-314. EXMC 库函数 .....	305
表 3-315. 结构体 exmc_norsram_timing_parameter_struct.....	306
表 3-316. 结构体 exmc_norsram_parameter_struct .....	306
表 3-317. 结构体 exmc_nand_pccard_timing_parameter_struct .....	307
表 3-318. 结构体 exmc_nand_parameter_struct .....	307
表 3-319. 结构体 exmc_pccard_parameter_struct .....	307
表 3-320. 结构体 exmc_sdram_timing_parameter_struct .....	308
表 3-321. 结构体 exmc_sdram_parameter_struct.....	308
表 3-322. 结构体 exmc_sdram_command_parameter_struct .....	309
表 3-323. 结构体 exmc_sqipssram_parameter_struct .....	309
表 3-324. 函数 exmc_norsram_deinit.....	309
表 3-325. 函数 exmc_norsram_struct_para_init .....	310
表 3-326. 函数 exmc_norsram_init .....	310
表 3-327. 函数 exmc_norsram_enable .....	312
表 3-328. 函数 exmc_norsram_disable .....	312
表 3-329. 函数 exmc_nand_deinit .....	313
表 3-330. 函数 exmc_nand_struct_para_init .....	313
表 3-331. 函数 exmc_nand_init .....	314
表 3-332. 函数 exmc_nand_enable .....	315
表 3-333. 函数 exmc_nand_disable .....	315
表 3-334. 函数 exmc_pccard_deinit .....	316
表 3-335. 函数 exmc_pccard_struct_para_init .....	316
表 3-336. 函数 exmc_pccard_init .....	317
表 3-337. 函数 exmc_pccard_enable .....	318
表 3-338. 函数 exmc_pccard_disable .....	318
表 3-339. 函数 exmc_sdram_deinit .....	319

---

表 3-340. 函数 exmc_sdram_struct_para_init .....	319
表 3-341. 函数 exmc_sdram_init .....	320
表 3-342. 函数 exmc_sqpipsram_deinit.....	321
表 3-343. 函数 exmc_sqpipsram_struct_para_init .....	322
表 3-344. 函数 exmc_sqpipsram_init .....	322
表 3-345. 函数 exmc_norsram_consecutive_clock_config .....	323
表 3-346. 函数 exmc_norsram_page_size_config .....	324
表 3-347. 函数 exmc_nand_ecc_config .....	325
表 3-348. 函数 exmc_ecc_get .....	325
表 3-349. 函数 exmc_sdram_readsampel_enable.....	326
表 3-350. 函数 exmc_sdram_readsampel_config .....	326
表 3-351. 函数 exmc_sdram_command_config .....	327
表 3-352. 函数 exmc_sdram_refresh_count_set.....	328
表 3-353. 函数 exmc_sdram_autorefresh_number_set .....	328
表 3-354. 函数 exmc_sdram_write_protection_config.....	329
表 3-355. 函数 exmc_sdram_bankstatus_get.....	329
表 3-356. 函数 exmc_sqpipsram_read_command_set .....	330
表 3-357. 函数 exmc_sqpipsram_write_command_set.....	331
表 3-358. 函数 exmc_sqpipsram_read_id_command_send .....	332
表 3-359. 函数 exmc_sqpipsram_write_cmd_send .....	332
表 3-360. 函数 exmc_sqpipsram_low_id_get .....	333
表 3-361. 函数 exmc_sqpipsram_low_id_get .....	333
表 3-362. 函数 exmc_sqpipsram_send_command_state_get .....	334
表 3-363. 函数 exmc_interrupt_enable .....	334
表 3-364. 函数 exmc_interrupt_disable .....	335
表 3-365. 函数 exmc_flag_get.....	336
表 3-366. 函数 exmc_flag_clear .....	337
表 3-367. 函数 exmc_interrupt_flag_get .....	338
表 3-368. 函数 exmc_interrupt_flag_clear .....	339
表 3-369. EXTI 寄存器.....	341
表 3-370. EXTI 库函数.....	341
表 3-371. 函数 exti_deinit .....	342
表 3-372. 函数 exti_init.....	342
表 3-373. 函数 exti_interrupt_enable.....	343
表 3-374. 函数 exti_interrupt_disable.....	344
表 3-375. 函数 exti_event_enable .....	345
表 3-376. 函数 exti_event_disable .....	345
表 3-377. 函数 exti_software_interrupt_enable .....	346
表 3-378. 函数 exti_software_interrupt_disable .....	346
表 3-379. 函数 exti_flag_get.....	347
表 3-380. 函数 exti_flag_clear.....	348
表 3-381. 函数 exti_interrupt_flag_get .....	348
表 3-382. 函数 exti_interrupt_flag_clear .....	349

---

表 3-383. FMC 寄存器.....	350
表 3-384. FMC 固件库函数.....	350
表 3-385. fmc_state_enum .....	351
表 3-386. 函数 fmc_wscnt_set.....	352
表 3-387. 函数 fmc_unlock.....	352
表 3-388. 函数 fmc_lock.....	353
表 3-389. 函数 fmc_sector_erase.....	353
表 3-390. 函数 fmc_mass_erase .....	354
表 3-391. 函数 fmc_bank0_erase .....	355
表 3-392. 函数 fmc_bank1_erase .....	355
表 3-393. 函数 fmc_word_program.....	356
表 3-394. 函数 fmc_halfword_program.....	356
表 3-395. 函数 fmc_byte_program.....	357
表 3-396. 函数 ob_unlock .....	358
表 3-397. 函数 ob_lock.....	358
表 3-398. 函数 ob_start .....	359
表 3-399. 函数 ob_write_protection_enable.....	360
表 3-400. 函数 ob_write_protection_disable.....	360
表 3-401. 函数 ob_drp_enable .....	361
表 3-402. 函数 ob_drp_disable .....	362
表 3-403. 函数 ob_security_protection_config.....	363
表 3-404. 函数 ob_user_write .....	363
表 3-405. 函数 ob_user_bor_threshold.....	364
表 3-406. 函数 ob_boot_mode_config .....	365
表 3-407. 函数 ob_user_get .....	366
表 3-408. 函数 ob_write_protection0_get.....	366
表 3-409. 函数 ob_write_protection1_get.....	367
表 3-410. 函数 ob_drp0_get .....	367
表 3-411. 函数 ob_drp1_get .....	368
表 3-412. 函数 ob_spc_get.....	368
表 3-413. 函数 ob_user_bor_threshold_get.....	369
表 3-414. 函数 fmc_interrupt_enable .....	371
表 3-415. 函数 fmc_interrupt_disable .....	371
表 3-416. 函数 fmc_flag_get.....	372
表 3-417. 函数 fmc_flag_clear .....	373
表 3-418. 函数 fmc_state_get .....	374
表 3-419. 函数 fmc_ready_wait.....	374
表 3-420. FWDGT 寄存器 .....	375
表 3-421. FWDGT 库函数 .....	375
表 3-422. 函数 fwdgt_write_enable.....	375
表 3-423. 函数 fwdgt_write_disable.....	376
表 3-424. 函数 fwdgt_enable.....	377
表 3-425. 函数 fwdgt_counter_reload .....	377

---

表 3-426. 函数 fwdgt_config .....	378
表 3-427. 函数 fwdgt_flag_get .....	379
表 3-428. GPIO 寄存器.....	错误!未定义书签。
表 3-429. GPIO 库函数.....	错误!未定义书签。
表 3-430. 函数 gpio_deinit.....	错误!未定义书签。
表 3-431. 函数 gpio_mode_set .....	错误!未定义书签。
表 3-432. 函数 gpio_output_options_set.....	错误!未定义书签。
表 3-433. 函数 gpio_bit_set.....	错误!未定义书签。
表 3-434. 函数 gpio_bit_reset .....	错误!未定义书签。
表 3-435. 函数 gpio_bit_write .....	错误!未定义书签。
表 3-436. 函数 gpio_port_write.....	错误!未定义书签。
表 3-437. 函数 gpio_input_bit_get .....	错误!未定义书签。
表 3-438. 函数 gpio_input_port_get .....	错误!未定义书签。
表 3-439. 函数 gpio_output_bit_get.....	错误!未定义书签。
表 3-440. 函数 gpio_output_port_get.....	错误!未定义书签。
表 3-441. 函数 gpio_af_set.....	错误!未定义书签。
表 3-442. 函数 gpio_pin_lock .....	错误!未定义书签。
表 3-443. 函数 gpio_bit_toggle .....	错误!未定义书签。
表 3-444. 函数 gpio_port_toggle .....	错误!未定义书签。
表 3-445. I2C 寄存器 .....	391
表 3-446. I2C 库函数 .....	392
表 3-447. 函数 i2c_deinit.....	393
表 3-448. 函数 i2c_clock_config .....	394
表 3-449. 函数 i2c_mode_addr_config.....	395
表 3-450. 函数 i2c_smbus_type_config .....	396
表 3-451. 函数 i2c_ack_config.....	397
表 3-452. 函数 i2c_ackpos_config .....	398
表 3-453. 函数 i2c_master_addressing .....	398
表 3-454. 函数 i2c_dualaddr_enable .....	399
表 3-455. 函数 i2c_enable.....	400
表 3-456. 函数 i2c_disable.....	401
表 3-457. 函数 i2c_start_on_bus .....	402
表 3-458. 函数 i2c_stop_on_bus .....	402
表 3-459. 函数 i2c_data_transmit.....	403
表 3-460. 函数 i2c_data_receive.....	404
表 3-461. 函数 i2c_dma_enable.....	404
表 3-462. 函数 i2c_dma_last_transfer_enable .....	405
表 3-463. 函数 i2c_stretch_scl_low_config .....	406
表 3-464. 函数 i2c_slave_response_to_gcall_config .....	407
表 3-465. 函数 i2c_software_reset_config .....	407
表 3-466. 函数 i2c_pec_enable .....	408
表 3-467. 函数 i2c_pec_transfer_enable .....	409
表 3-468. 函数 i2c_pec_value_get.....	410

---

表 3-469. 函数 i2c_smbus_issue_alert .....	411
表 3-470. 函数 i2c_smbus_arp_enable .....	411
表 3-471. 函数 i2c_analog_noise_filter_disable .....	412
表 3-472. 函数 i2c_analog_noise_filter_enable .....	413
表 3-473. 函数 i2c_digital_noise_filter_config .....	414
表 3-474. 函数 i2c_sam_enable .....	414
表 3-475. 函数 i2c_sam_disable .....	415
表 3-476. 函数 i2c_sam_timeout_enable .....	416
表 3-477. 函数 i2c_sam_timeout_disable .....	416
表 3-478. 函数 i2c_flag_get .....	417
表 3-479. 函数 i2c_flag_clear .....	419
表 3-480. 函数 i2c_interrupt_enable .....	420
表 3-481. 函数 i2c_interrupt_disable .....	421
表 3-482. 函数 i2c_interrupt_flag_get .....	422
表 3-483. 函数 i2c_interrupt_flag_clear .....	424
表 3-484. IPA 寄存器 .....	425
表 3-485. IPA 库函数 .....	426
表 3-486. 结构体 ipa_foreground_parameter_struct .....	427
表 3-487. 结构体 ipa_background_parameter_struct .....	427
表 3-488. 结构体 ipa_destination_parameter_struct .....	428
表 3-489. 函数 ipa_deinit .....	428
表 3-490. 函数 ipa_transfer_enable .....	428
表 3-491. 函数 ipa_transfer_hangup_enable .....	429
表 3-492. 函数 ipa_transfer_hangup_disable .....	429
表 3-493. 函数 ipa_transfer_stop_enable .....	430
表 3-494. 函数 ipa_transfer_stop_disable .....	430
表 3-495. 函数 ipa_foreground_lut_loading_enable .....	431
表 3-496. 函数 ipa_background_lut_loading_enable .....	431
表 3-497. 函数 ipa_pixel_format_convert_mode_set .....	432
表 3-498. 函数 ipa_foreground_struct_para_init .....	432
表 3-499. 函数 ipa_foreground_init .....	433
表 3-500. 函数 ipa_background_struct_para_init .....	434
表 3-501. 函数 ipa_background_init .....	434
表 3-502. 函数 ipa_destination_struct_para_init .....	435
表 3-503. 函数 ipa_destination_init .....	436
表 3-504. 函数 ipa_foreground_lut_init .....	437
表 3-505. 函数 ipa_background_lut_init .....	438
表 3-506. 函数 ipa_line_mark_config .....	438
表 3-507. 函数 ipa_inter_timer_config .....	439
表 3-508. 函数 ipa_interval_clock_num_config .....	439
表 3-509. 函数 .....	440
表 3-510. 函数 .....	440
表 3-511. 函数 ipa_interrupt_enable .....	441

---

表 3-512. 函数 ipa_interrupt_disable.....	442
表 3-513. 函数 ipa_interrupt_flag_get .....	442
表 3-514. 函数 ipa_interrupt_flag_clear.....	443
表 3-515. IREF 寄存器 .....	444
表 3-516. IREF 库函数 .....	444
表 3-517. 函数 iref_deinit.....	444
表 3-518. 函数 iref_enable .....	445
表 3-519. 函数 iref_disable.....	445
表 3-520. 函数 iref_mode_set .....	446
表 3-521. 函数 iref_sink_set.....	446
表 3-522. 函数 iref_precision_trim_value_set.....	447
表 3-523 函数 iref_step_data_config.....	448
表 3-524. NVIC 寄存器 .....	448
表 3-525. Systick 寄存器.....	449
表 3-526. 枚举类型 IRQn_Type.....	450
表 3-527. MISC 库函数.....	453
表 3-528. 函数 nvic_priority_group_set.....	453
表 3-529. 函数 nvic_irq_enable .....	454
表 3-530. 函数 nvic_irq_disable .....	455
表 3-531. 函数 nvic_vector_table_set .....	455
表 3-532. 函数 system_lowpower_set.....	456
表 3-533. 函数 system_lowpower_reset .....	457
表 3-534. 函数 systick_clksource_set.....	458
表 3-535. PMU 寄存器.....	459
表 3-536. PMU 库函数.....	459
表 3-537. 函数 pmu_deinit.....	460
表 3-538. 函数 pmu_lvd_select.....	460
表 3-539. 函数 pmu_ldo_output_select .....	461
表 3-540. 函数 pmu_low_driver_mode_enable .....	462
表 3-541. 函数 pmu_highdriver_switch_select.....	463
表 3-542. 函数 pmu_highdriver_mode_enable .....	463
表 3-543. 函数 pmu_highdriver_mode_disable .....	464
表 3-544. 函数 pmu_lvd_disable .....	465
表 3-545. 函数 pmu_lowdriver_lowpower_config .....	465
表 3-546. 函数 pmu_lowdriver_normalpower_config .....	466
表 3-547. 函数 pmu_to_sleepmode .....	467
表 3-548. 函数 pmu_to_deepsleepmode.....	467
表 3-549. 函数 pmu_to_standbymode.....	468
表 3-550. 函数 pmu_backup_ldo_config .....	469
表 3-551. 函数 pmu_wakeup_pin_enable .....	470
表 3-552. 函数 pmu_wakeup_pin_disable .....	470
表 3-553. 函数 pmu_backup_write_enable.....	471
表 3-554. 函数 pmu_backup_write_disable.....	472

---

表 3-555. 函数 pmu_flag_get .....	472
表 3-556. 函数 pmu_flag_reset .....	473
表 3-557. RCU 寄存器 .....	474
表 3-558. RCU 库函数 .....	476
表 3-559. 函数 rcu_deinit .....	477
表 3-560. 函数 rcu_periph_clock_enable .....	478
表 3-561. 函数 rcu_periph_clock_disable .....	480
表 3-562. 函数 rcu_periph_clock_sleep_enable .....	481
表 3-563. 函数 rcu_periph_clock_sleep_disable .....	483
表 3-564. 函数 rcu_periph_reset_enable .....	485
表 3-565. 函数 rcu_periph_reset_disable .....	487
表 3-566. 函数 rcu_bkp_reset_enable .....	488
表 3-567. 函数 rcu_bkp_reset_disable .....	489
表 3-568. 函数 rcu_system_clock_source_config .....	490
表 3-569. 函数 rcu_system_clock_source_get .....	490
表 3-570. 函数 rcu_ahb_clock_config .....	491
表 3-571. 函数 rcu_apb1_clock_config .....	492
表 3-572. 函数 rcu_apb2_clock_config .....	493
表 3-573. 函数 rcu_ckout0_config .....	494
表 3-574. 函数 rcu_ckout1_config .....	495
表 3-575. 函数 rcu_pll_config .....	496
表 3-576. 函数 rcu_plli2s_config .....	497
表 3-577. 函数 rcu_pll sai_config .....	498
表 3-578. 函数 rcu_rtc_clock_config .....	498
表 3-579. 函数 rcu_i2s_clock_config .....	499
表 3-580. 函数 rcu_ck48m_clock_config .....	500
表 3-581. 函数 rcu_pll48m_clock_config .....	501
表 3-582. 函数 rcu_timer_clock_prescaler_config .....	502
表 3-583. 函数 rcu_tli_clock_div_config .....	502
表 3-584. 函数 rcu_flag_get .....	503
表 3-585. 函数 rcu_all_reset_flag_clear .....	505
表 3-586. 函数 rcu_interrupt_flag_get .....	505
表 3-587. 函数 rcu_interrupt_flag_clear .....	506
表 3-588. 函数 rcu_interrupt_enable .....	508
表 3-589. 函数 rcu_interrupt_disable .....	509
表 3-590. 函数 rcu_lxtal_drive_capability_config .....	510
表 3-591. 函数 rcu_oscstab_wait .....	510
表 3-592. 函数 rcu_oscion .....	511
表 3-593. 函数 rcu_oscloff .....	512
表 3-594. 函数 rcu_oscibypass_mode_enable .....	513
表 3-595. 函数 rcu_oscibypass_mode_disable .....	514
表 3-596. 函数 rcu_hxtal_clock_monitor_enable .....	514
表 3-597. 函数 rcu_hxtal_clock_monitor_disable .....	515

---

表 3-598. 函数 rcu_irc16m_adjust_value_set .....	516
表 3-599. 函数 rcu_spread_spectrum_config .....	516
表 3-600. 函数 rcu_spread_spectrum_enable .....	517
表 3-601. 函数 rcu_spread_spectrum_disable .....	518
表 3-602. 函数 rcu_voltage_key_unlock .....	519
表 3-603. 函数 rcu_deepsleep_voltage_set .....	519
表 3-604. 函数 rcu_clock_freq_get .....	520
表 3-605. RTC 寄存器 .....	521
表 3-606. RTC 库函数 .....	522
表 3-607. 结构体 rtc_parameter_struct .....	524
表 3-608. 结构体 rtc_alarm_struct .....	524
表 3-609. 结构体 rtc_timestamp_struct .....	524
表 3-610. 结构体 rtc_tamper_struct .....	525
表 3-611. 函数 rtc_deinit .....	525
表 3-612. 函数 rtc_init .....	526
表 3-613. 函数 rtc_init_mode_enter .....	527
表 3-614. 函数 rtc_init_mode_exit .....	527
表 3-615. 函数 rtc_register_sync_wait .....	528
表 3-616. 函数 rtc_current_time_get .....	529
表 3-617. 函数 rtc_subsecond_get .....	529
表 3-618. 函数 rtc_alarm_config .....	530
表 3-619. 函数 rtc_alarm_subsecond_config .....	531
表 3-620. 函数 rtc_alarm_get .....	532
表 3-621. 函数 rtc_alarm_subsecond_get .....	533
表 3-622. 函数 rtc_alarm_enable .....	534
表 3-623. 函数 rtc_alarm_disable .....	534
表 3-624. 函数 rtc_timestamp_enable .....	535
表 3-625. 函数 rtc_timestamp_disable .....	536
表 3-626. 函数 rtc_timestamp_get .....	536
表 3-627. 函数 rtc_timestamp_subsecond_get .....	537
表 3-628. 函数 rtc_timestamp_pin_map .....	538
表 3-629. 函数 rtc_timestamp_enable .....	538
表 3-630. 函数 rtc_tamper_disable .....	539
表 3-631. 函数 rtc_tamper0_pin_map .....	540
表 3-632. 函数 rtc_interrupt_enable .....	540
表 3-633. 函数 rtc_interrupt_disable .....	541
表 3-634. 函数 rtc_flag_get .....	542
表 3-635. 函数 rtc_flag_clear .....	543
表 3-636. 函数 rtc_alter_output_config .....	544
表 3-637. 函数 rtc_calibration_output_config .....	545
表 3-638. 函数 rtc_hour_adjust .....	546
表 3-639. 函数 rtc_second_adjust .....	546
表 3-640. 函数 rtc_bypass_shadow_enable .....	547

---

表 3-641. 函数 <code>rtc_bypass_shadow_disable</code> .....	548
表 3-642. 函数 <code>rtc_refclock_detection_enable</code> .....	548
表 3-643. 函数 <code>rtc_refclock_detection_disable</code> .....	549
表 3-644. 函数 <code>rtc_wakeup_enable</code> .....	550
表 3-645. 函数 <code>rtc_wakeup_disable</code> .....	550
表 3-646. 函数 <code>rtc_wakeup_clock_set</code> .....	551
表 3-647. 函数 <code>rtc_wakeup_timer_set</code> .....	552
表 3-648. 函数 <code>rtc_wakeup_timer_get</code> .....	552
表 3-649. 函数 <code>rtc_smooth_calibration_config</code> .....	553
表 3-650. 函数 <code>rtc_coarse_calibration_enable</code> .....	554
表 3-651. 函数 <code>rtc_coarse_calibration_disable</code> .....	555
表 3-652. 函数 <code>rtc_coarse_calibration_config</code> .....	555
表 3-653. SDIO 寄存器.....	557
表 3-654. SDIO 库函数.....	557
表 3-655. 函数 <code>sdio_deinit</code> .....	559
表 3-656. 函数 <code>sdio_clock_config</code> .....	560
表 3-657. 函数 <code>sdio_hardware_clock_enable</code> .....	561
表 3-658. 函数 <code>sdio_hardware_clock_disable</code> .....	562
表 3-659. 函数 <code>sdio_bus_mode_set</code> .....	562
表 3-660. 函数 <code>sdio_power_state_set</code> .....	563
表 3-661. 函数 <code>sdio_power_state_get</code> .....	564
表 3-662. 函数 <code>sdio_clock_enable</code> .....	564
表 3-663. 函数 <code>sdio_clock_disable</code> .....	565
表 3-664. 函数 <code>sdio_command_response_config</code> .....	566
表 3-665. 函数 <code>sdio_wait_type_set</code> .....	567
表 3-666. 函数 <code>sdio_csm_enable</code> .....	567
表 3-667. 函数 <code>sdio_csm_disable</code> .....	568
表 3-668. 函数 <code>sdio_command_index_get</code> .....	569
表 3-669. 函数 <code>sdio_response_get</code> .....	569
表 3-670. 函数 <code>sdio_data_config</code> .....	570
表 3-671. 函数 <code>sdio_data_transfer_config</code> .....	572
表 3-672. 函数 <code>sdio_dsm_enable</code> .....	573
表 3-673. 函数 <code>sdio_dsm_disable</code> .....	573
表 3-674. 函数 <code>sdio_data_write</code> .....	574
表 3-675. 函数 <code>sdio_data_read</code> .....	574
表 3-676. 函数 <code>sdio_data_counter_get</code> .....	575
表 3-677. 函数 <code>sdio_data_counter_get</code> .....	576
表 3-678. 函数 <code>sdio_dma_enable</code> .....	576
表 3-679. 函数 <code>sdio_dma_disable</code> .....	577
表 3-680. 函数 <code>sdio_flag_get</code> .....	578
表 3-681. 函数 <code>sdio_flag_clear</code> .....	580
表 3-682. 函数 <code>sdio_interrupt_enable</code> .....	581
表 3-683. 函数 <code>sdio_interrupt_disable</code> .....	583

---

表 3-684. 函数 sdio_interrupt_flag_get.....	584
表 3-685. 函数 sdio_interrupt_flag_clear.....	586
表 3-686. 函数 sdio_readwait_enable.....	588
表 3-687. 函数 sdio_readwait_disable.....	588
表 3-688. 函数 sdio_stop_readwait_enable.....	589
表 3-689. 函数 sdio_stop_readwait_disable .....	590
表 3-690. 函数 sdio_readwait_type_set .....	590
表 3-691. 函数 sdio_operation_enable.....	591
表 3-692. 函数 sdio_operation_disable.....	592
表 3-693. 函数 sdio_suspend_enable .....	592
表 3-694. 函数 sdio_suspend_disable .....	593
表 3-695. 函数 sdio_ceata_command_enable .....	593
表 3-696. 函数 sdio_ceata_command_disable .....	594
表 3-697. 函数 sdio_ceata_interrupt_enable.....	595
表 3-698. 函数 sdio_ceata_interrupt_disable.....	595
表 3-699. 函数 sdio_ceata_command_completion_enable .....	596
表 3-700. 函数 sdio_ceata_command_completion_disable .....	596
表 3-701. SPI/I2S 寄存器 .....	597
表 3-702. SPI/I2S 库函数 .....	598
表 3-703. 结构体 spi_parameter_struct.....	599
表 3-704. 函数 spi_i2s_deinit.....	599
表 3-705. 函数 spi_struct_para_init .....	600
表 3-706. 函数 spi_init.....	600
表 3-707. 函数 spi_enable .....	601
表 3-708. 函数 spi_disable .....	602
表 3-709. 函数 i2s_init .....	602
表 3-710. 函数 i2s_psc_config.....	603
表 3-711. 函数 i2s_enable .....	605
表 3-712. 函数 i2s_disable.....	605
表 3-713. 函数 spi_nss_output_enable .....	606
表 3-714. 函数 spi_nss_output_disable .....	606
表 3-715. 函数 spi_nss_internal_high .....	607
表 3-716. 函数 spi_nss_internal_low.....	607
表 3-717. 函数 spi_dma_enable .....	608
表 3-718. 函数 spi_dma_disable .....	609
表 3-719. 函数 spi_i2s_data_frame_format_config .....	609
表 3-720. 函数 spi_i2s_data_transmit .....	610
表 3-721. 函数 spi_i2s_data_receive .....	610
表 3-722. 函数 spi_bidirectional_transfer_config .....	611
表 3-723. 函数 spi_crc_polynomial_set .....	612
表 3-724. 函数 spi_crc_polynomial_get.....	612
表 3-725. 函数 spi_crc_on .....	613
表 3-726. 函数 spi_crc_off.....	613

---

表 3-727. 函数 spi_crc_next.....	614
表 3-728. 函数 spi_crc_get.....	614
表 3-729. 函数 spi_ti_mode_enable.....	615
表 3-730. 函数 spi_ti_mode_disable.....	615
表 3-731. 函数 i2s_full_duplex_mode_config .....	616
表 3-732. 函数 qspi_enable .....	617
表 3-733. 函数 qspi_disable .....	618
表 3-734. 函数 qspi_write_enable .....	618
表 3-735. 函数 qspi_read_enable .....	619
表 3-736. 函数 qspi_io23_output_enable .....	619
表 3-737. 函数 qspi_io23_output_disable .....	620
表 3-738. 函数 spi_i2s_interrupt_enable .....	620
表 3-739. 函数 spi_i2s_interrupt_disable .....	621
表 3-740. 函数 spi_i2s_interrupt_flag_get.....	622
表 3-741. 函数 spi_i2s_flag_get.....	623
表 3-742. 函数 spi_crc_error_clear.....	624
表 3-743. SYSCFG 寄存器.....	624
表 3-744. SYSCFG 库函数 .....	625
表 3-745. 函数 syscfg_deinit.....	625
表 3-746. 函数 syscfg_bootmode_config .....	626
表 3-747. 函数 syscfg_fmc_swap_config .....	627
表 3-748. 函数 syscfg_exmc_swap_config .....	627
表 3-749. 函数 syscfg_exti_line_config .....	628
表 3-750. 函数 syscfg_enet_phy_interface_config .....	629
表 3-751. 函数 syscfg_compensation_config .....	630
表 3-752. 函数 syscfg_flag_get .....	631
表 3-753. TIMER 寄存器 .....	631
表 3-754. TIMER 库函数 .....	632
表 3-755. 结构体 timer_parameter_struct.....	634
表 3-756. 结构体 timer_break_parameter_struct.....	635
表 3-757. 结构体 timer_oc_parameter_struct .....	635
表 3-758. 结构体 timer_ic_parameter_struct.....	635
表 3-759. 函数 timer_deinit.....	636
表 3-760. Function timer_struct_para_init.....	636
表 3-761. 函数 timer_init.....	637
表 3-762. 函数 timer_enable.....	638
表 3-763. 函数 timer_disable.....	638
表 3-764. 函数 timer_auto_reload_shadow_enable .....	639
表 3-765. 函数 timer_auto_reload_shadow_disable .....	639
表 3-766. 函数 timer_update_event_enable .....	640
表 3-767. 函数 timer_update_event_disable .....	640
表 3-768. 函数 timer_counter_alignment.....	641
表 3-769. 函数 timer_counter_up_direction .....	641

---

表 3-770. 函数 timer_counter_down_direction.....	642
表 3-771. 函数 timer_prescaler_config .....	642
表 3-772. 函数 timer_repetition_value_config .....	643
表 3-773. 函数 timer_autoreload_value_config .....	644
表 3-774. 函数 timer_counter_value_config .....	644
表 3-775. 函数 timer_counter_read.....	645
表 3-776. 函数 timer_prescaler_read.....	645
表 3-777. 函数 timer_single_pulse_mode_config .....	646
表 3-778. 函数 timer_update_source_config .....	647
表 3-779. 函数 timer_dma_enable.....	647
表 3-780. 函数 timer_dma_disable.....	648
表 3-781. 函数 timer_channel_dma_request_source_select .....	649
表 3-782. 函数 timer_dma_transfer_config .....	650
表 3-783. 函数 timer_event_software_generate .....	651
表 3-784. 函数 timer_break_struct_para_init .....	652
表 3-785. 函数 timer_break_config .....	653
表 3-786. 函数 timer_break_enable .....	654
表 3-787. 函数 timer_break_disable .....	654
表 3-788. 函数 timer_automatic_output_enable.....	655
表 3-789. 函数 timer_automatic_output_disable.....	655
表 3-790. 函数 timer_primary_output_config .....	656
表 3-791. 函数 timer_channel_control_shadow_config.....	656
表 3-792. 函数 timer_channel_control_shadow_update_config .....	657
表 3-793. 函数 timer_channel_output_struct_para_init.....	658
表 3-794. 函数 timer_channel_output_config .....	658
表 3-795. 函数 timer_channel_output_mode_config .....	659
表 3-796. 函数 timer_channel_output_pulse_value_config .....	660
表 3-797. 函数 timer_channel_output_shadow_config.....	661
表 3-798. 函数 timer_channel_output_fast_config .....	662
表 3-799. 函数 timer_channel_output_clear_config .....	663
表 3-800. 函数 timer_channel_output_polarity_config .....	664
表 3-801. 函数 timer_channel_complementary_output_polarity_config .....	664
表 3-802. 函数 timer_channel_output_state_config .....	665
表 3-803. 函数 timer_channel_complementary_output_state_config .....	666
表 3-804. 函数 timer_channel_input_struct_para_init .....	667
表 3-805. 函数 timer_input_capture_config .....	667
表 3-806. 函数 timer_channel_input_capture_prescaler_config .....	668
表 3-807. 函数 timer_channel_capture_value_register_read.....	669
表 3-808. 函数 timer_input_pwm_capture_config .....	670
表 3-809. 函数 timer_hall_mode_config .....	671
表 3-810. 函数 timer_input_trigger_source_select.....	671
表 3-811. 函数 timer_master_output_trigger_source_select .....	672
表 3-812. 函数 timer_slave_mode_select .....	673

---

表 3-813. 函数 timer_master_slave_mode_config .....	674
表 3-814. 函数 timer_external_trigger_config .....	675
表 3-815. 函数 timer_quadrature_decoder_mode_config .....	676
表 3-816. 函数 timer_internal_clock_config .....	677
表 3-817. 函数 timer_internal_trigger_as_external_clock_config .....	678
表 3-818. 函数 timer_external_trigger_as_external_clock_config .....	678
表 3-819. 函数 timer_external_clock_mode0_config .....	679
表 3-820. 函数 timer_external_clock_mode1_config .....	680
表 3-821. 函数 timer_external_clock_mode1_disable .....	681
表 3-822. 函数 timer_channel_remap_config .....	682
表 3-823. 函数 timer_write_chxval_register_config .....	683
表 3-824. 函数 timer_output_value_selection_config .....	684
表 3-825. 函数 timer_interrupt_enable .....	686
表 3-826. 函数 timer_interrupt_disable .....	687
表 3-827. 函数 timer_interrupt_flag_get .....	688
表 3-828. 函数 timer_interrupt_flag_clear .....	689
表 3-829. 函数 timer_flag_get .....	684
表 3-830. 函数 timer_flag_clear .....	685
表 3-831. TLI 寄存器 .....	690
表 3-832. TLI 库函数 .....	691
表 3-833. 结构体 tli_parameter_struct .....	692
表 3-834. 结构体 tli_layer_parameter_struct .....	693
表 3-835. 结构体 tli_layer_lut_parameter_struct .....	693
表 3-836. 函数 tli_deinit .....	694
表 3-837. 函数 tli_struct_para_init .....	694
表 3-838. 函数 tli_init .....	695
表 3-839. 函数 tli_dither_config .....	696
表 3-840. 函数 tli_enable .....	697
表 3-841. 函数 tli_disable .....	698
表 3-842. 函数 tli_reload_config .....	698
表 3-843. 函数 tli_layer_struct_para_init .....	699
表 3-844. 函数 tli_layer_init .....	700
表 3-845. 函数 tli_layer_window_offset_modify .....	701
表 3-846. 函数 tli_lut_struct_para_init .....	702
表 3-847. 函数 tli_lut_init .....	703
表 3-848. 函数 tli_color_key_init .....	704
表 3-849. 函数 tli_layer_enable .....	705
表 3-850. 函数 tli_layer_disable .....	705
表 3-851. 函数 tli_color_key_enable .....	706
表 3-852. 函数 tli_color_key_disable .....	707
表 3-853. 函数 tli_lut_enable .....	707
表 3-854. 函数 tli_lut_disable .....	708
表 3-855. 函数 tli_line_mark_set .....	709

---

表 3-856. 函数 tli_current_pos_get.....	709
表 3-857. 函数 tli_interrupt_enable.....	710
表 3-858. 函数 tli_interrupt_disable.....	711
表 3-859. 函数 tli_interrupt_flag_get .....	711
表 3-860. 函数 tli_interrupt_flag_clear .....	712
表 3-861. 函数 tli_flag_get.....	713
表 3-862. TRNG 寄存器 .....	714
表 3-863. TRNG 库函数 .....	714
表 3-864 函数 trng_deinit .....	715
表 3-865 函数 trng_enable.....	716
表 3-866 函数 trng_disable.....	716
表 3-867 函数 trng_get_true_random_data.....	717
表 3-868 函数 trng_interrupt_enable.....	717
表 3-869 函数 trng_interrupt_disable.....	718
表 3-870 函数 trng_flag_get.....	719
表 3-871 函数 trng_interrupt_flag_get .....	719
表 3-872 函数 trng_interrupt_flag_clear .....	720
表 3-873. USART 寄存器 .....	721
表 3-874. USART 库函数 .....	722
表 3-875. 函数 usart_deinit.....	723
表 3-876. 函数 usart_baudrate_set .....	724
表 3-877. 函数 usart_parity_config .....	724
表 3-878. 函数 usart_word_length_set.....	725
表 3-879. 函数 usart_stop_bit_set .....	725
表 3-880. 函数 usart_enable.....	726
表 3-881. 函数 usart_disable.....	727
表 3-882. 函数 usart_transmit_config .....	727
表 3-883. 函数 usart_receive_config .....	728
表 3-884. 函数 usart_data_first_config .....	729
表 3-885. 函数 usart_invert_config .....	729
表 3-886. 函数 usart_oversample_config .....	730
表 3-887. 函数 usart_sample_bit_config .....	731
表 3-888. 函数 usart_receiver_timeout_enable .....	731
表 3-889. 函数 usart_receiver_timeout_disable .....	732
表 3-890. 函数 usart_receiver_timeout_threshold_config .....	732
表 3-891. 函数 usart_data_transmit.....	733
表 3-892. 函数 usart_data_receive.....	734
表 3-893. 函数 usart_address_config.....	734
表 3-894. 函数 usart_mute_mode_enable .....	735
表 3-895. 函数 usart_mute_mode_disable .....	735
表 3-896. 函数 usart_mute_mode_wakeup_config .....	736
表 3-897. 函数 usart_lin_mode_enable .....	737
表 3-898. 函数 usart_lin_mode_disable .....	737

---

表 3-899. 函数 usart_lin_break_dection_length_config .....	738
表 3-900. 函数 usart_send_break.....	738
表 3-901. 函数 usart_halfduplex_enable.....	739
表 3-902. 函数 usart_halfduplex_disable.....	739
表 3-903. 函数 usart_synchronous_clock_enable.....	740
表 3-904. 函数 usart_synchronous_clock_disable.....	740
表 3-905. 函数 usart_synchronous_clock_config.....	741
表 3-906. 函数 usart_guard_time_config.....	742
表 3-907. 函数 usart_smartcard_mode_enable.....	742
表 3-908. 函数 usart_smartcard_mode_disable .....	743
表 3-909. 函数 usart_smartcard_mode_nack_enable .....	743
表 3-910. 函数 usart_smartcard_mode_nack_disable .....	744
表 3-911. 函数 usart_smartcard_autoretry_config.....	744
表 3-912. 函数 usart_block_length_config.....	745
表 3-913. 函数 usart_irda_mode_enable .....	746
表 3-914. 函数 usart_irda_mode_disable .....	746
表 3-915. 函数 usart_prescaler_config .....	747
表 3-916. 函数 usart_irda_lowpower_config.....	747
表 3-917. 函数 usart_hardware_flow_rts_config.....	748
表 3-918. 函数 usart_hardware_flow_cts_config .....	749
表 3-919. 函数 usart_break_frame_coherence_config.....	749
表 3-920. 函数 usart_parity_check_coherence_config .....	750
表 3-921. 函数 usart_hardware_flow_coherence_config .....	751
表 3-922. 函数 usart_dma_receive_config .....	751
表 3-923. 函数 usart_dma_transmit_config .....	752
表 3-924. 函数 usart_flag_get .....	753
表 3-925. 函数 usart_flag_clear .....	754
表 3-926. 函数 usart_interrupt_enable .....	754
表 3-927. 函数 usart_interrupt_disable .....	755
表 3-928. 函数 usart_interrupt_flag_get .....	756
表 3-929. 函数 usart_interrupt_flag_clear .....	757
表 3-930. WWDGT 寄存器 .....	758
表 3-931. WWDGT 库函数 .....	759
表 3-932. 函数 wwdgt_deinit .....	759
表 3-933. 函数 wwdgt_enable .....	760
表 3-934. 函数 wwdgt_counter_update .....	760
表 3-935. 函数 wwdgt_config .....	761
表 3-936. 函数 wwdgt_interrupt_enable .....	762
表 3-937. 函数 wwdgt_flag_get .....	762
表 3-938. 函数 wwdgt_flag_clear .....	763
表 4-1. 版本历史 .....	765



## 1. 介绍

本手册介绍了32位基于ARM微控制器GD32F4xx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GD32F4xx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)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
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
DAC	数模转换器
DBG	调试模块
DCI	数字摄像头接口
DMA	直接存储器访问控制器
ENET	以太网
EXMC	外部存储器控制器

外设缩写	说明
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
I2C	内部集成电路总线接口
IPA	图像处理加速器
IREF	可编程参考电流
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
SYSCFG	系统配置
TIMER	定时器
TLI	TFT-LCD接口
TRNG	真随机数生成器
USART	通用同步异步收发器
WWDGT	窗口看门狗
USBFS	通用串行总线全速接口

### 1.1.2. 命名规则

固件库遵从以下命名规则：

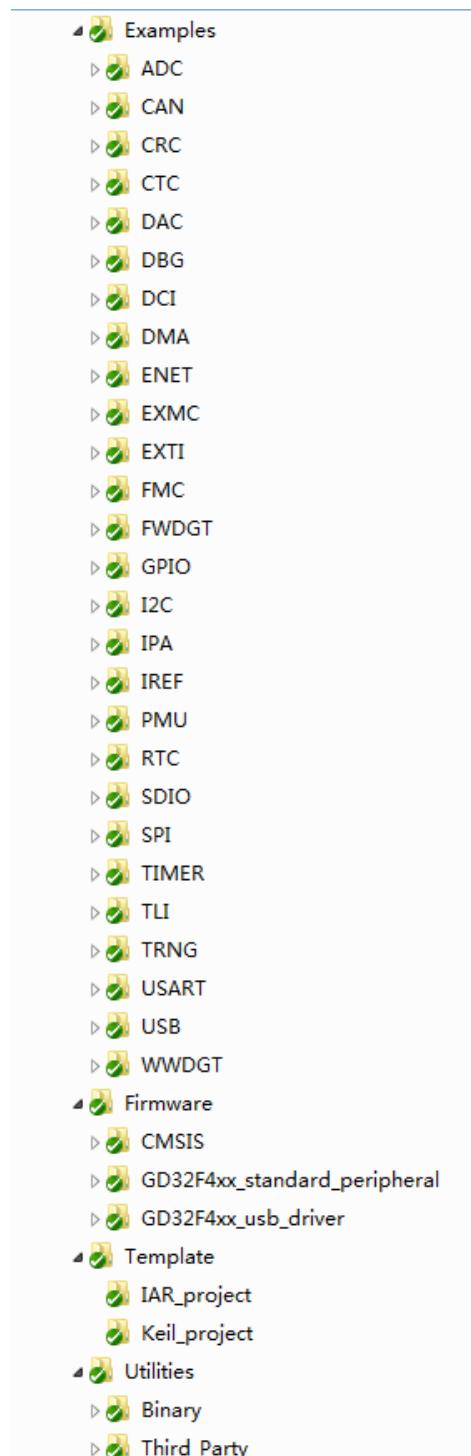
- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；
- 源文件和头文件命名都以“gd32f4xx\_”作为开头，例如：gd32f4xx\_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

## 2. 固件库概述

### 2.1. 文件组织结构

GD32F4xx\_Firmware\_Library, 文件组织结构见下图:

图 2-1. GD32F4xx 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;
- **gd32f4xx\_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);
- **gd32f4xx\_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);
- **gd32f4xx\_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 M4**内核的支持文件、基于**Cortex M4**内核处理器的启动代码和库引导文件以及基于**GD32f4xx**的全局头文件和系统配置文件;
- **GD32F4xx\_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;
- **GD32F4xx\_usbfs\_driver**子文件夹包含了关于**USBFS**外设的所有文件;
  - **Include**子文件夹包含了**USBFS**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**USBFS**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. Project 文件夹

**Project**文件夹包含关于**USBFS**外设的使用例程(EWARM用于IAR编译环境, MDK-ARM用于Keil4编译环境)。

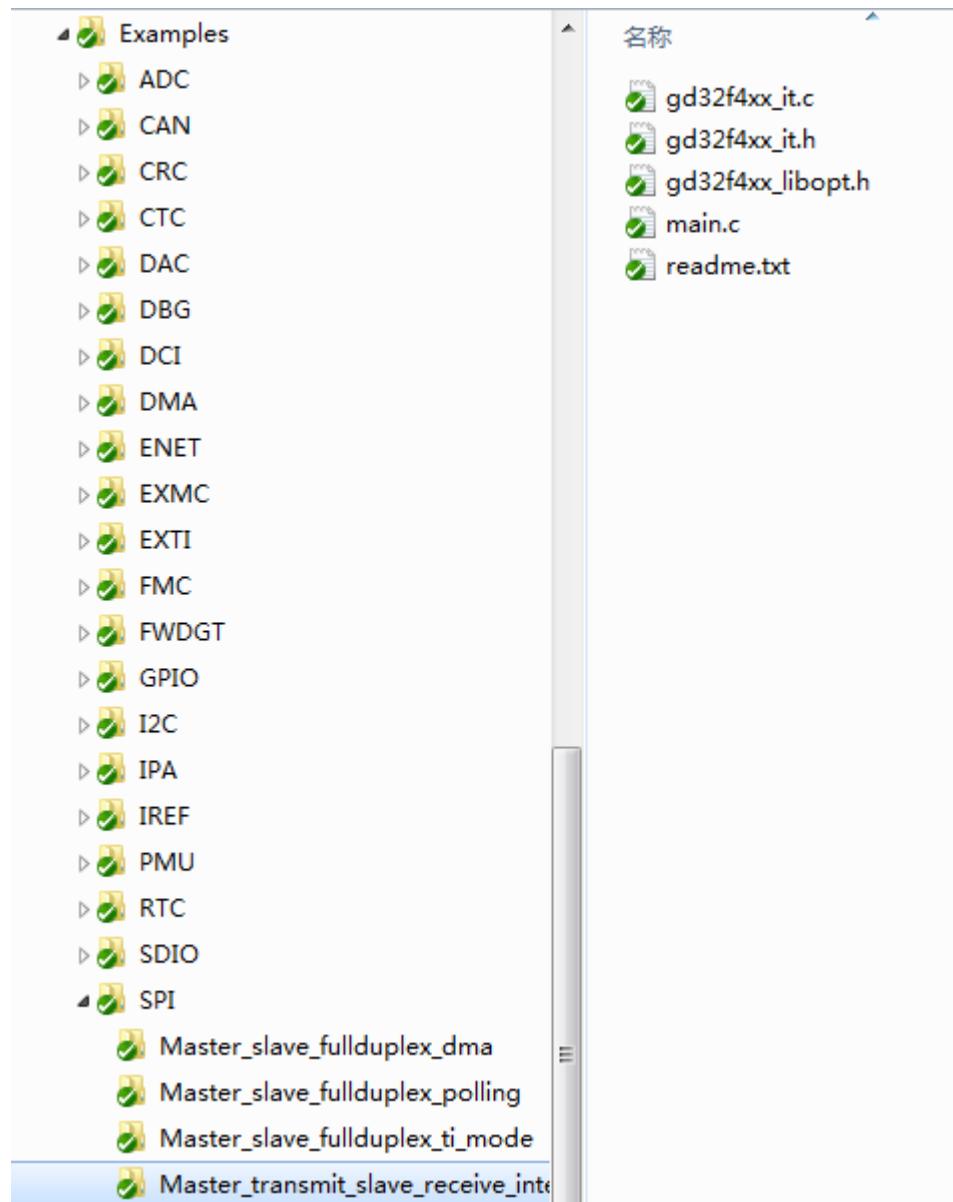
### 2.1.4. Template 文件夹

**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Keil4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：

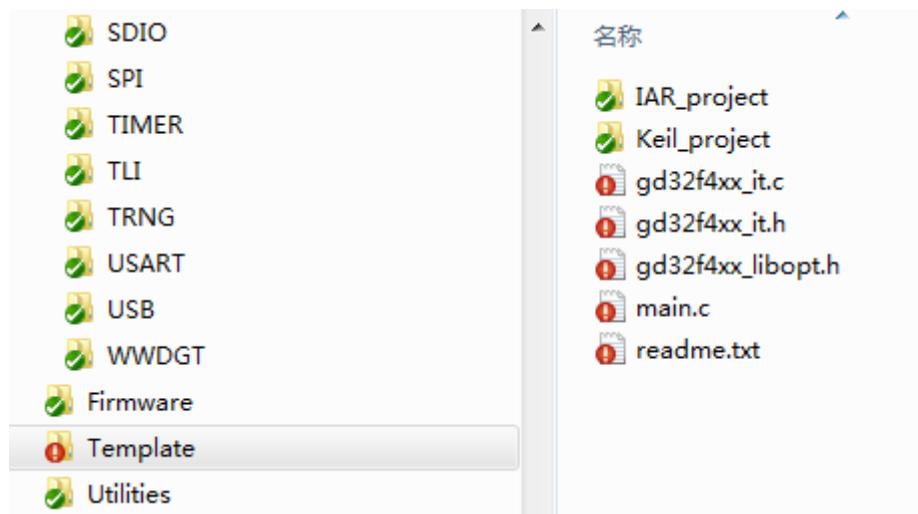
图 2-2. 选择外设例程文件



## 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：

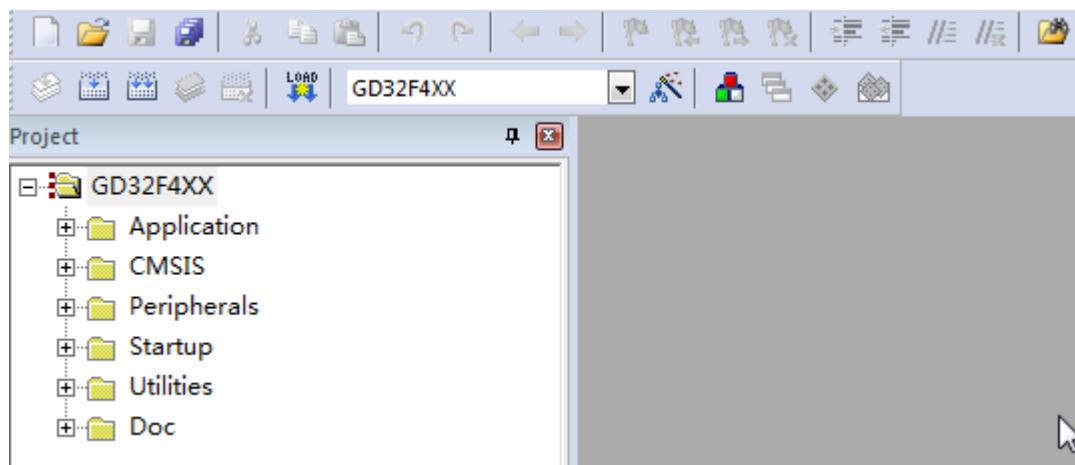
图 2-3. 拷贝外设例程文件



### 打开工程

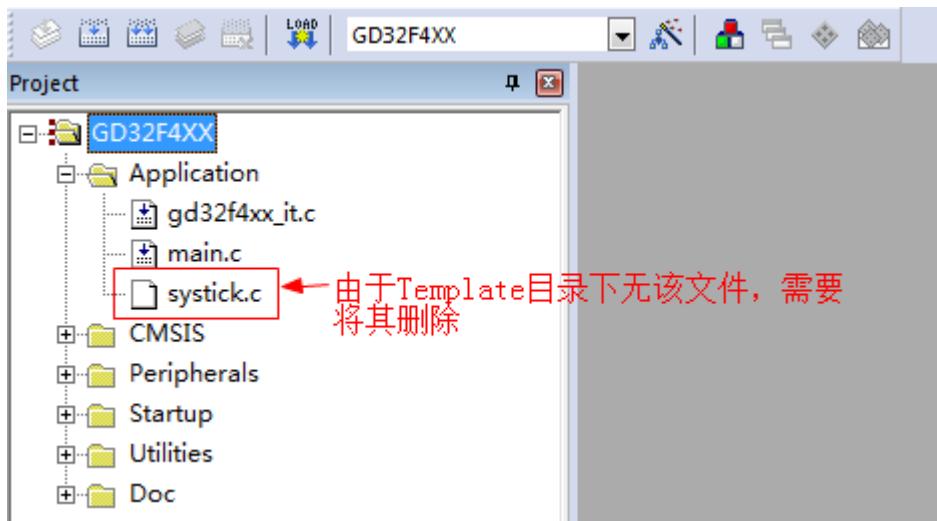
GD 提供 Keil 和 IAR 两种版本的工程，根据客户所安装的软件，打开不同的 project，如“Keil\_project”，打开\Template\Keil\_project\Project.uvproj，如下图所示：

图 2-4. 打开工程文件



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

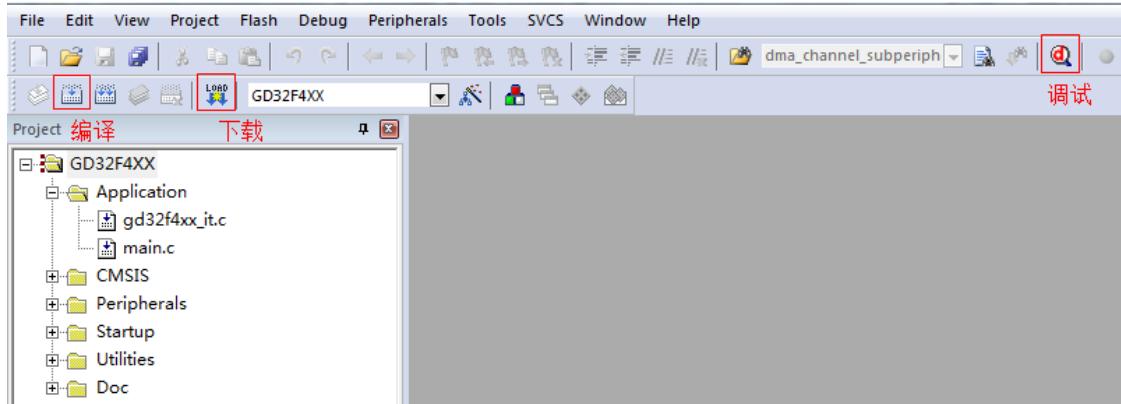
图 2-5. 配置工程文件



### 编译调试下载

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

图 2-6. 编译调试下载



### 2.1.5. Utilities 文件夹

Utilities文件夹包含运行固件库例程评估板的文件：

- Binary、Third\_Party子文件夹包含有USB测试所需文件；
- gd32f4xx\_eval.h文件是运行固件库例程所需关于评估板的头文件；
- gd32f4xx\_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
gd32f4xx_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。
gd32f4xx_it.h	头文件，包含所有中断处理函数原形。
gd32f4xx_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。
gd32f4xx_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。
gd32f4xx_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
<b>输入参数{in}</b>	
XXX	输入参数描述
Xx	输入参数可选宏描述
<b>输出参数{out}</b>	
XXX	输出参数描述
<b>返回值</b>	
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

寄存器名称	寄存器描述
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_WDHT	看门狗高阈值寄存器
ADC_WDLT	看门狗低阈值寄存器
ADC_RSQ0	规则序列寄存器0
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_SSTAT	摘要状态寄存器
ADC_SYNCCTL	同步控制寄存器
ADC_SYNCDATA	同步规则数据寄存器

### 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ADCx外设
adc_clock_config	ADC时钟配置
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_enable	使能ADCx外设
adc_disable	禁能ADCx外设

库函数名称	库函数描述
adc_calibration_enable	ADCx校准复位
adc_channel_16_to_18	配置内部温度传感器采集通道, Vrefint电压采集通道通道, VBAT电压采集通道
adc_resolution_config	配置ADCx分辨率
adc_oversample_mode_config	配置ADCx过采样模式
adc_oversample_mode_enable	使能ADCx过采样
adc_oversample_mode_disable	禁能ADCx过采样
adc_dma_mode_enable	ADCx DMA请求使能
adc_dma_mode_disable	ADCx DMA请求禁能
adc_dma_request_after_last_enable	当DMA=1时, 在每个规则通道转换结束后提出一个DMA请求
adc_dma_request_after_last_disable	DMA机制在DMA控制器的传输结束信号之后失能
adc_discontinuous_mode_config	配置ADC间断模式
adc_channel_length_config	配置规则通道组或注入通道组的长度
adc_regular_channel_config	配置ADC规则通道组
adc_inserted_channel_config	配置ADC注入通道组
adc_inserted_channel_offset_config	配置ADC注入通道组数据偏移值
adc_external_trigger_source_config	配置ADC外部触发源
adc_external_trigger_config	配置ADC外部触发
adc_software_trigger_enable	ADC软件触发使能
adc_end_of_conversion_config	配置规则组转换结束模式
adc_regular_data_read	读ADC规则组数据寄存器
adc_inserted_data_read	读ADC注入组数据寄存器
adc_watchdog_single_channel_disable	配置ADC模拟看门狗单通道无效
adc_watchdog_single_channel_enable	配置ADC模拟看门狗单通道有效
adc_watchdog_group_channel_enable	配置ADC模拟看门狗在通道组有效

库函数名称	库函数描述
adc_watchdog_disable	ADC模拟看门狗禁能
adc_watchdog_threshold_config	配置ADC模拟看门狗阈值
adc_flag_get	获取ADC标志位
adc_flag_clear	清除ADC标志位
adc_regular_software_startconv_flag_get	获取ADC规则通道组软件触发转换开始位
adc_inserted_software_startconv_flag_get	获取ADC注入通道组软件触发转换开始位
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
adc_sync_mode_config	ADC同步模式配置
adc_sync_delay_config	配置ADC同步模式两次采样之间的延时
adc_sync_dma_config	ADC同步DMA模式选择
adc_sync_dma_request_after_last_enable	当SYNCDMA不为00时，根据SYNCDMA位来产生DMA请求
adc_sync_dma_request_after_last_disable	当检测到来自DMA控制器的传输结束信号后，DMA机制失能
adc_sync_regular_data_read	ADC同步模式规则数据读取

### 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(void);
功能描述	复位所有的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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset ADC */

adc_deinit();
```

### 函数 **adc\_clock\_config**

函数adc\_clock\_config描述见下表：

表 3-5. 函数 **adc\_clock\_config**

输入参数{in}	
<b>prescaler</b>	预分频
<i>ADC_ADCCK_PCL_K2_DIV2</i>	PCLK2时钟2分频
<i>ADC_ADCCK_PCL_K2_DIV4</i>	PCLK2时钟4分频
<i>ADC_ADCCK_PCL_K2_DIV6</i>	PCLK2时钟6分频
<i>ADC_ADCCK_PCL_K2_DIV8</i>	PCLK2时钟8分频
<i>ADC_ADCCK_HCL_K_DIV5</i>	HCLK时钟5分频
<i>ADC_ADCCK_HCL</i>	HCLK时钟6分频

<i>K_DIV6</i>	
<i>ADC_ADCCK_HCLK_DIV10</i>	HCLK时钟10分频
<i>ADC_ADCCK_HCLK_DIV20</i>	HCLK时钟20分频
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the ADC clock: PCLK2 div2 */
adc_clock_config(ADC_ADCCK_PCLK2_DIV2);
```

### 函数 **adc\_special\_function\_config**

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

**表 3-6. 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	-
<b>输入参数{in}</b>	
<b>adc_periph</b>	ADC外设
<i>ADCx(x=0..2)</i>	ADC外设选择
<b>输入参数{in}</b>	
<b>function</b>	功能配置
<i>ADC_SCAN_MODE</i>	扫描模式选择
<i>ADC_INSERTED_CHANNEL_AUTO</i>	注入组自动转换
<i>ADC_CONTINUOUS</i>	连续模式选择

<b>S_MODE</b>	
<b>输入参数{in}</b>	
<b>newvalue</b>	功能使能禁能
<b>ENABLE</b>	使能
<b>DISABLE</b>	禁能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-7. 函数 **adc\_data\_alignment\_config**

函数名称	adc_data_alignment_config
函数原形	void adc_data_alignment_config(uint32_t adc_periph, uint32_t data_alignment);
功能描述	配置ADCx数据对齐方式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>adc_periph</b>	ADC外设
<b>ADCx(x=0..2)</b>	ADC外设选择
<b>输入参数{in}</b>	
<b>data_alignment</b>	数据对齐方式选择
<b>ADC_DATAALIGN_RIGHT</b>	LSB 对齐
<b>ADC_DATAALIGN_</b>	MSB 对齐

<i>LEFT</i>	
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\_enable**

函数adc\_enable描述见下表：

表 3-8. 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ADCx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 3-9. 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ADCx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 3-10. 函数 adc\_calibration\_enable**

函数名称	adc_calibration_enable
函数原形	void adc_calibration_enable(uint32_t adc_periph);
功能描述	ADCx校准复位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	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\_channel\_16\_to\_18**

函数adc\_channel\_16\_to\_18描述见下表：

**表 3-11. 函数 adc\_channel\_16\_to\_18**

函数名称	adc_channel_16_to_18
函数原形	void adc_channel_16_to_18(uint32_t function, ControlStatus newvalue);
功能描述	配置内部温度传感器采集通道, Vrefint电压采集通道通道, VBAT电压采集通道
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>function</b>	选择被配置的内部通道
<b>ADC_VBAT_CHAN NEL_SWITCH</b>	选择配置内部通道18 (1/4VBAT电压) 通道
<b>ADC_TEMP_VREF _CHANNEL_SWITC H</b>	选择配置内部通道16 (温度传感器) 通道和通道17 (Vrefint电压) 通道
<b>输入参数{in}</b>	
<b>newvalue</b>	功能使能禁能
<b>ENABLE</b>	使能
<b>DISABLE</b>	禁能
<b>输出参数{out}</b>	
-	-
返回值	
-	-

例如：

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

adc_channel_16_to_18(ADC_TEMP_VREF_CHANNEL_SWITCH, ENABLE);
```

### 函数 **adc\_resolution\_config**

函数adc\_resolution\_config描述见下表：

**表 3-12. 函数 adc\_resolution\_config**

函数名称	adc_resolution_config
函数原形	void adc_resolution_config(uint32_t adc_periph , uint32_t resolution);
功能描述	配置ADCx分辨率
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}	
resolution	ADC分辨率
ADC_RESOLUTION_12B	12位分辨率
ADC_RESOLUTION_10B	10位分辨率
ADC_RESOLUTION_8B	8位分辨率
ADC_RESOLUTION_6B	6位分辨率
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure ADC0 resolution: 10 bits */
```

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

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

函数adc\_oversample\_mode\_config描述见下表：

**表 3-13. 函数 adc\_oversample\_mode\_config**

函数名称	adc_oversample_mode_config
函数原形	void adc_oversample_mode_config(uint32_t adc_periph, uint32_t mode, uint16_t shift, uint8_t ratio);
功能描述	配置ADCx过采样模式
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}	
mode	ADC过采样触发模式
ADC_OVERSAMPLING_ALL_CHANNELS_CONVERT	在一个触发之后，对一个通道连续进行过采样转换
ADC_OVERSAMPLING_ONE_CONVERT	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}	
shift	ADC过滤采样移位
ADC_OVERSAMPLING_SHIFT_NONE	不移位
ADC_OVERSAMPLING_SHIFT_1B	移1位
ADC_OVERSAMPLING_SHIFT_2B	移2位

<i>ADC_OVERSA</i>	
<i>MPLING_SHIFT_3B</i>	移3位
<i>ADC_OVERSA</i>	
<i>MPLING_SHIFT_4B</i>	移4位
<i>ADC_OVERSA</i>	
<i>MPLING_SHIFT_5B</i>	移5位
<i>ADC_OVERSA</i>	
<i>MPLING_SHIFT_6B</i>	移6位
<i>ADC_OVERSA</i>	
<i>MPLING_SHIFT_7B</i>	移7位
<i>ADC_OVERSA</i>	
<i>MPLING_SHIFT_8B</i>	移8位
<b>输入参数{in}</b>	
<b>ratio</b>	ADC过采样率
<i>ADC_OVERSA</i>	
<i>MPLING_RATIO_MUL2</i>	2x
<i>ADC_OVERSA</i>	
<i>MPLING_RATIO_MUL4</i>	4x
<i>ADC_OVERSA</i>	
<i>MPLING_RATIO_MUL8</i>	8x
<i>ADC_OVERSA</i>	
<i>MPLING_RATIO_MUL16</i>	16x
<i>ADC_OVERSA</i>	
<i>MPLING_RATIO_MUL32</i>	32x
<i>ADC_OVERSA</i>	64x

<i>MPLING_RATIO_MUL64</i>	
<i>ADC_OVERSAMPLING_RATIO_MUL128</i>	128x
<i>ADC_OVERSAMPLING_RATIO_MUL256</i>	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 3-14. 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ADCx过采样
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0..2)</code>	ADC外设选择
输出参数{out}	
-	-
返回值	

-	-
---	---

Example:

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

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

函数adc\_oversample\_mode\_disable描述见下表:

**表 3-15. 函数 adc\_oversample\_mode\_disable**

函数名称	adc_oversample_mode_disable
函数原形	void adc_oversample_mode_disable(uint32_t adc_periph);
功能描述	禁能ADCx过采样
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	
-	-

Example:

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

#### 函数 **adc\_dma\_mode\_enable**

函数adc\_dma\_mode\_enable描述见下表:

**表 3-16. 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	ADCx 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 3-17. 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	ADCx 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_dma_request_after_last_enable`

函数`adc_dma_request_after_last_enable`描述见下表:

表 3-18. 函数 `adc_dma_request_after_last_enable`

函数名称	adc_dma_request_after_last_enable
函数原形	<code>void adc_dma_request_after_last_enable(uint32_t adc_periph);</code>
功能描述	当DMA=1时，在每个规则通道转换结束后提出一个DMA请求
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:

```
/* when DMA=1, the DMA engine issues a request at end of each regular conversion for ADC0 */

```

```
adc_dma_request_after_last_enable (ADC0);
```

#### 函数 `adc_dma_request_after_last_disable`

函数`adc_dma_request_after_last_disable`描述见下表:

表 3-19. 函数 `adc_dma_request_after_last_disable`

函数名称	adc_dma_request_after_last_disable
函数原形	<code>void adc_dma_request_after_last_disable(uint32_t adc_periph);</code>
功能描述	DMA机制在DMA控制器的传输结束信号之后失能
先决条件	-

被调用函数	-
<b>输入参数{in}</b>	
<b>adc_periph</b>	ADC外设
<b>ADCx(x=0..2)</b>	ADC外设选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* the DMA engine is disabled after the end of transfer signal from DMA controller is detected
for ADC0 */

adc_dma_request_after_last_disable (ADC0);
```

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

函数adc\_discontinuous\_mode\_config描述见下表：

表 3-20. 函数 **adc\_discontinuous\_mode\_config**

函数名称	adc_discontinuous_mode_config
函数原形	void adc_discontinuous_mode_config(uint32_t adc_periph, uint8_t adc_channel_group, uint8_t length);
功能描述	配置ADC间断模式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>adc_periph</b>	ADC外设
<b>ADCx(x=0..2)</b>	ADC外设选择
<b>输入参数{in}</b>	
<b>adc_channel_group</b>	通道组选择
<b>ADC_REGULAR_CHANNEL</b>	规则通道组

<i>ADC_INSERTED_CHANNEL</i>	注入通道组
<i>ADC_CHANNEL_DISABLE</i>	规则通道组和注入通道组间断模式禁能
<b>输入参数{in}</b>	
<b>length</b>	间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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

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

函数adc\_channel\_length\_config描述见下表：

表 3-21. 函数 **adc\_channel\_length\_config**

函数名称	adc_channel_length_config
函数原形	void adc_channel_length_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t length);
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	-
<b>输入参数{in}</b>	
<b>adc_periph</b>	ADC外设
<i>ADCx(x=0..2)</i>	ADC外设选择
<b>输入参数{in}</b>	
<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
<b>输入参数{in}</b>	
<b>length</b>	通道长度，规则通道组为1-16，注入通道组为1-4
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-22. 函数 adc\_regular\_channel\_config**

函数名称	adc_regular_channel_config
函数原形	void adc_regular_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
功能描述	配置ADC规则通道组
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>adc_periph</b>	ADC外设
<i>ADCx(x=0..2)</i>	ADC外设选择
<b>输入参数{in}</b>	
<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(x=0..18)</i>	ADC 通道x (x=0..18)(只有ADC0, 可取值x=16..18 )

输入参数{in}	
<b>sample_time</b>	采样时间
<i>ADC_SAMPLETIME_3</i>	3 周期
<i>ADC_SAMPLETIME_15</i>	15 周期
<i>ADC_SAMPLETIME_28</i>	28 周期
<i>ADC_SAMPLETIME_56</i>	56 周期
<i>ADC_SAMPLETIME_84</i>	84 周期
<i>ADC_SAMPLETIME_112</i>	112 周期
<i>ADC_SAMPLETIME_144</i>	144 周期
<i>ADC_SAMPLETIME_480</i>	480 周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC0 regular channel */
adc_regular_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_56);
```

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

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

表 3-23. 函数 **adc\_inserted\_channel\_config**

函数名称	adc_inserted_channel_config
函数原形	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>	-
<b>输入参数{in}</b>	
<b>adc_periph</b>	ADC外设
<b>ADCx(x=0..2)</b>	ADC外设选择
<b>输入参数{in}</b>	
<b>rank</b>	注入组通道序列，取值范围为0~3
<b>输入参数{in}</b>	
<b>adc_channel</b>	ADC通道选择
<b>ADC_CHANNEL_x(x=0..18)</b>	ADC 通道x (x=0..18)(只有ADC0, 可取值x=16..18)
<b>输入参数{in}</b>	
<b>sample_time</b>	采样时间
<b>ADC_SAMPLETIME_3</b>	3 周期
<b>ADC_SAMPLETIME_15</b>	15 周期
<b>ADC_SAMPLETIME_28</b>	28 周期
<b>ADC_SAMPLETIME_56</b>	56 周期
<b>ADC_SAMPLETIME_84</b>	84 周期
<b>ADC_SAMPLETIME_112</b>	112 周期
<b>ADC_SAMPLETIME_144</b>	144 周期
<b>ADC_SAMPLETIME_480</b>	480 周期
<b>输出参数{out}</b>	

-	-
返回值	
-	-

例如：

```
/* configure ADC0 inserted channel */
adc_inserted_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_56);
```

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

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

**表 3-24. 函数 adc\_inserted\_channel\_offset\_config**

函数名称	adc_inserted_channel_offset_config
函数原形	void adc_inserted_channel_offset_config(uint32_t adc_periph, uint8_t inserted_channel, uint16_t offset);
功能描述	配置ADC注入通道组数据偏移值
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
<b>输入参数{in}</b>	
inserted_channel	注入通道选择
ADC_INSERTED_CHANNEL_x(x=0..3)	注入通道, x=0,1,2,3
<b>输入参数{in}</b>	
offset	数据偏移值, 取值范围为0~4095
<b>输出参数{out}</b>	
-	-
返回值	
-	-

例如：

---

```

/* configure ADC0 inserted channel offset */

adc_inserted_channel_offset_config(ADC0, ADC_INSERTED_CHANNEL_0, 100);

```

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

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

**表 3-25. 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
ADC_EXTTRIG_RULEAR_T0_CH0	TIMER0 CH0事件（规则组）
ADC_EXTTRIG_RULEAR_T0_CH1	TIMER0 CH1事件（规则组）
ADC_EXTTRIG_RULEAR_T0_CH2	TIMER0 CH2事件（规则组）
ADC_EXTTRIG_RULEAR_T1_CH1	TIMER1 CH1事件（规则组）

<i>GULAR_T1_CH1</i>	
<i>ADC_EXTTRIG_RE</i> <i>GULAR_T1_CH2</i>	TIMER1 CH2事件（规则组）
<i>ADC_EXTTRIG_RE</i> <i>GULAR_T1_CH3</i>	TIMER3 CH3事件（规则组）
<i>ADC_EXTTRIG_RE</i> <i>GULAR_T1_TRGO</i>	TIMER1 TRGO事件（规则组）
<i>ADC_EXTTRIG_RE</i> <i>GULAR_T2_CH0</i>	TIMER2 CH0事件（规则组）
<i>ADC_EXTTRIG_RE</i> <i>GULAR_T2_TRGO</i>	TIMER2 TRGO事件（规则组）
<i>ADC_EXTTRIG_RE</i> <i>GULAR_T3_CH3</i>	TIMER3 CH3事件（规则组）
<i>ADC_EXTTRIG_RE</i> <i>GULAR_T4_CH0</i>	TIMER4 CH0事件（规则组）
<i>ADC_EXTTRIG_RE</i> <i>GULAR_T4_CH1</i>	TIMER4 CH1事件（规则组）
<i>ADC_EXTTRIG_RE</i> <i>GULAR_T4_CH2</i>	TIMER4 CH2事件（规则组）
<i>ADC_EXTTRIG_RE</i> <i>GULAR_T7_CH0</i>	TIMER7 CH0事件（规则组）
<i>ADC_EXTTRIG_RE</i> <i>GULAR_T7_TRGO</i>	TIMER7 TRGO事件（规则组）
<i>ADC_EXTTRIG_RE</i> <i>GULAR_EXTI_11</i>	外部中断线11（规则组）
<i>ADC_EXTTRIG_IN</i> <i>SERTED_T0_CH3</i>	TIMER0 CH3事件（注入组）
<i>ADC_EXTTRIG_IN</i> <i>SERTED_T0_TRGO</i>	TIMER0 TRGO事件（注入组）
<i>ADC_EXTTRIG_IN</i> <i>SERTED_T1_CH0</i>	TIMER1 CH0事件（注入组）
<i>ADC_EXTTRIG_IN</i> <i>SERTED_T1_TRGO</i>	TIMER1 TRGO事件（注入组）

<i>ADC_EXTTRIG_IN SERTED_T2_CH1</i>	TIMER2 CH1事件（注入组）
<i>ADC_EXTTRIG_IN SERTED_T2_CH3</i>	TIMER2 CH3事件（注入组）
<i>ADC_EXTTRIG_IN SERTED_T3_CH0</i>	TIMER3 CH0事件（注入组）
<i>ADC_EXTTRIG_IN SERTED_T3_CH1</i>	TIMER3 CH1事件（注入组）
<i>ADC_EXTTRIG_IN SERTED_T3_CH2</i>	TIMER3 CH2事件（注入组）
<i>ADC_EXTTRIG_IN SERTED_T3_TRGO</i>	TIMER3 TRGO事件（注入组）
<i>ADC_EXTTRIG_IN SERTED_T4_CH3</i>	TIMER4 CH3事件（注入组）
<i>ADC_EXTTRIG_IN SERTED_T4_TRGO</i>	TIMER4 TRGO事件（注入组）
<i>ADC_EXTTRIG_IN SERTED_T7_CH1</i>	TIMER7 CH1事件（注入组）
<i>ADC_EXTTRIG_IN SERTED_T7_CH2</i>	TIMER7 CH2事件（注入组）
<i>ADC_EXTTRIG_IN SERTED_T7_CH3</i>	TIMER7 CH3事件（注入组）
<i>ADC_EXTTRIG_IN SERTED_EXTI_15</i>	外部中断线15（注入组）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure ADC0 regular channel external trigger source */
adc_external_trigger_source_config(ADC0,ADC_REGULAR_CHANNEL,
```

ADC\_EXTTRIG\_REGULAR\_T0\_CH0);

### 函数 `adc_external_trigger_config`

函数`adc_external_trigger_config`描述见下表：

**表 3-26. 函数 `adc_external_trigger_config`**

函数名称	adc_external_trigger_config
函数原形	<code>void adc_external_trigger_config(uint32_t adc_periph , uint8_t adc_channel_group , uint32_t trigger_mode);</code>
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}	
<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>trigger_mode</code>	通道使能禁能
<code>EXTERNAL_TRIGGER_DISABLE</code>	外部触发禁能
<code>EXTERNAL_TRIGGER_RISING</code>	上升沿触发
<code>EXTERNAL_TRIGGER_FALLING</code>	下降沿触发
<code>EXTERNAL_TRIGGER_RISING_FALLING</code>	双边沿触发

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,
EXTERNAL_TRIGGER_RISING);
```

#### 函数 **adc\_software\_trigger\_enable**

函数adc\_software\_trigger\_enable描述见下表：

**表 3-27. 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\_end\_of\_conversion\_config**

函数adc\_end\_of\_conversion\_config描述见下表：

**表 3-28. 函数 adc\_end\_of\_conversion\_config**

函数名称	adc_end_of_conversion_config
函数原形	void adc_end_of_conversion_config(uint32_t adc_periph , uint8_t end_selection);
功能描述	配置转换结束模式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
<b>输入参数{in}</b>	
end_selection	通道组选择
ADC_EOC_SET_SEQUENCE	只有在规则转换序列转换结束时，才将EOC置1。如果不设置DMA=1，则溢出检测失能
ADC_EOC_SET_CONVERSION	在每个规则转换结束时，将EOC置1。溢出检测自动使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* at the end of each regular conversion, the EOC bit is set. */

adc_end_of_conversion_config (ADC0, ADC_EOC_SET_CONVERSION);
```

### 函数 **adc\_regular\_data\_read**

函数adc\_inserted\_regular\_data\_read描述见下表:

**表 3-29. 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 3-30. 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}	
<b>adc_periph</b>	ADC外设
<i>ADCx(x=0..2)</i>	ADC外设选择
输入参数{in}	
<b>inserted_channel</b>	注入通道选择
<i>ADC_INSERTED_CHANNEL_x(x=0..3)</i>	注入通道x, x=0,1,2,3
输出参数{out}	
-	-
返回值	
<b>uint16_t</b>	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\_watchdog\_single\_channel\_disable**

函数**adc\_watchdog\_single\_channel\_disable**描述见下表：

**表 3-31. 函数 **adc\_watchdog\_single\_channel\_disable****

函数名称	<b>adc_watchdog_single_channel_disable</b>
函数原形	void adc_watchdog_single_channel_disable(uint32_t adc_periph );
功能描述	配置ADC模拟看门狗单通道无效
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
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable ADC0 analog watchdog single channel */
adc_watchdog_single_channel_disable(ADC0);
```

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

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

表 3-32. 函数 **adc\_watchdog\_single\_channel\_enable**

函数名称	adc_watchdog_single_channel_enable
函数原形	void adc_watchdog_single_channel_enable(uint32_t adc_periph, uint8_t adc_channel);
功能描述	配置ADC模拟看门狗单通道有效
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_CHANNEL_x(x=0..17)	ADC Channelx(x=0..17) (只有ADC0, 可取值x=16和17)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC0 analog watchdog single channel */
adc_watchdog_single_channel_enable(ADC0, ADC_CHANNEL_1);
```

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

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

**表 3-33. 函数 adc\_watchdog\_group\_channel\_enable**

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

例如：

```
/* configure ADC0 analog watchdog group channel */
adc_watchdog_group_channel_enable(ADC0, ADC_REGULAR_CHANNEL);
```

### **函数 adc\_watchdog\_disable**

函数adc\_watchdog\_disable描述见下表：

**表 3-34. 函数 adc\_watchdog\_disable**

函数名称	adc_watchdog_disable
函数原形	void adc_watchdog_disable(uint32_t adc_periph);
功能描述	ADC模拟看门狗禁能
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 */
adc_watchdog_disable(ADC0);
```

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

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

**表 3-35. 函数 adc\_watchdog\_threshold\_config**

函数名称	adc_watchdog_threshold_config
函数原形	void adc_watchdog_threshold_config(uint32_t adc_periph, uint16_t low_threshold, uint16_t high_threshold);
功能描述	配置ADC模拟看门狗阈值
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
<b>输入参数{in}</b>	
<b>low_threshold</b>	模拟看门狗低阈值, 0..4095
<b>输入参数{in}</b>	
<b>high_threshold</b>	模拟看门狗高阈值, 0..4095
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

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

adc_watchdog_threshold_config(ADC0, 0x0400, 0xA00);
```

### 函数 **adc\_flag\_get**

函数**adc\_flag\_get**描述见下表:

表 3-36. 函数 **adc\_flag\_get**

函数名称	adc_flag_get
函数原形	FlagStatus adc_flag_get(uint32_t adc_periph , uint32_t adc_flag);
功能描述	获取ADC标志位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>adc_periph</b>	ADC外设
<i>ADCx(x=0..2)</i>	ADC外设选择
<b>输入参数{in}</b>	
<b>adc_flag</b>	ADC标志位
<i>ADC_FLAG_WDE</i>	模拟看门狗事件标志位
<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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET 或 RESET

例如：

```
/* get the ADC0 analog watchdog flag bits*/
FlagStatus flag_value;
flag_value = adc_flag_get(ADC0, ADC_FLAG_WDE);
```

### 函数 **adc\_flag\_clear**

函数adc\_flag\_clear描述见下表：

**表 3-37. 函数 **adc\_flag\_clear****

函数名称	adc_flag_clear
函数原形	void adc_flag_clear(uint32_t adc_periph, uint32_t adc_flag);
功能描述	清除ADC标志位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>adc_periph</b>	ADC外设
<b>ADCx(x=0..2)</b>	ADC外设选择
<b>输入参数{in}</b>	
<b>adc_flag</b>	ADC标志位
<i>ADC_FLAG_WDE</i>	模拟看门狗事件标志位
<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
<b>输出参数{out}</b>	
-	-
返回值	
-	-

例如：

```
/* clear the ADC0 analog watchdog flag bits*/
adc_flag_clear(ADC0, ADC_FLAG_WDE);
```

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

函数adc\_regular\_software\_startconv\_flag\_get描述见下表：

表 3-38. 函数 **adc\_regular\_software\_startconv\_flag\_get**

函数名称	adc_regular_software_startconv_flag_get
函数原形	FlagStatus adc_regular_software_startconv_flag_get(uint32_t adc_periph);
功能描述	获取ADC规则通道组软件触发转换开始位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<i>adc_periph</i>	ADC外设
<i>ADCx(x=0..2)</i>	ADC外设选择
<b>输出参数{out}</b>	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如：

```
/* get the bit state of ADC0 software regular channel start conversion */
FlagStatus flag_value;
flag_value = adc_regular_software_startconv_flag_get(ADC0);
```

### **函数 `adc_inserted_software_startconv_flag_get`**

函数`adc_inserted_software_startconv_flag_get`描述见下表：

**表 3-39. 函数 `adc_inserted_software_startconv_flag_get`**

函数名称	adc_inserted_software_startconv_flag_get
函数原形	FlagStatus adc_inserted_software_startconv_flag_get(uint32_t adc_periph);
功能描述	获取ADC规则注入组软件触发转换开始位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>adc_periph</b>	ADC外设
<b>ADCx(x=0..2)</b>	ADC外设选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET或RESET

例如：

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

FlagStatus flag_value;

flag_value = adc_inserted_software_startconv_flag_get(ADC0);
```

### **函数 `adc_interrupt_flag_get`**

函数`adc_interrupt_flag_get`描述见下表：

**表 3-40. 函数 `adc_interrupt_flag_get`**

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

<b>adc_periph</b>	ADC外设
<i>ADCx(x=0..2)</i>	ADC外设选择
<b>输入参数{in}</b>	
<b>adc_interrupt</b>	ADC中断标志位
<i>ADC_INT_WDE</i>	模拟看门狗中断标志位
<i>ADC_INT_EOC</i>	组转换结束中断标志位
<i>ADC_INT_EOIC</i>	注入通道组转换结束中断标志位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET 或 RESET

例如：

```
/* get the ADC0 analog watchdog interrupt bits*/
FlagStatus flag_value;
flag_value = adc_interrupt_flag_get(ADC0, ADC_INT_WDE);
```

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

函数adc\_interrupt\_flag\_clear描述见下表：

**表 3-41. 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 adc_interrupt);
功能描述	清除ADC中断标志位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>adc_periph</b>	ADC外设
<i>ADCx(x=0..2)</i>	ADC外设选择
<b>输入参数{in}</b>	
<b>adc_interrupt</b>	ADC中断标志位

<i>ADC_INT_WDE</i>	模拟看门狗中断标志位
<i>ADC_INT_EOC</i>	组转换结束中断标志位
<i>ADC_INT_EOIC</i>	注入通道组转换结束中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

### 函数 **adc\_interrupt\_enable**

函数adc\_interrupt\_enable描述见下表：

表 3-42. 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 adc_interrupt);
功能描述	ADC中断使能
先决条件	-
被调用函数	-
输入参数{in}	
<b>adc_periph</b>	ADC外设
<i>ADCx(x=0..2)</i>	ADC外设选择
输入参数{in}	
<b>adc_interrupt</b>	ADC中断标志位
<i>ADC_INT_WDE</i>	模拟看门狗中断标志位
<i>ADC_INT_EOC</i>	组转换结束中断标志位
<i>ADC_INT_EOIC</i>	注入通道组转换结束中断标志位
输出参数{out}	
-	-

返回值	
-	-

例如：

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

adc_interrupt_enable(ADC0, ADC_INT_WDE);
```

### 函数 **adc\_interrupt\_disable**

函数adc\_interrupt\_disable描述见下表：

**表 3-43. 函数 adc\_interrupt\_disable**

函数名称	adc_interrupt_disable
函数原形	void adc_interrupt_enable(uint32_t adc_periph, uint32_t adc_interrupt);
功能描述	ADC中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
adc_interrupt	ADC中断标志位
ADC_INT_WDE	模拟看门狗中断标志位
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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC0 interrupt */

adc_interrupt_disable(ADC0, ADC_INT_WDE);
```

### 函数 `adc_sync_mode_config`

函数`adc_sync_mode_config`描述见下表：

**表 3-44. 函数 `adc_sync_mode_config`**

函数名称	<code>adc_sync_mode_config</code>
函数原形	<code>void adc_sync_mode_config(uint32_t sync_mode);</code>
功能描述	ADC同步模式配置
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>sync_mode</code>	同步模式
<code>ADC_SYNC_MODE_INDEPENDENT</code>	ADC同步模式失能。所有的ADC都独立工作
<code>ADC_DAUL_REGU_LAL_PARALLEL_INSERTED_PARALLEL_L</code>	ADC0和ADC1工作在规则并行和注入并行的组合模式。ADC2 独立工作。
<code>ADC_DAUL_REGU_LAL_PARALLEL_INSERTED_ROTATION_N</code>	ADC0和ADC1工作在规则并行和交替触发的组合模式。ADC2 独立工作。
<code>ADC_DAUL_INSERTED_PARALLEL</code>	ADC0和ADC1工作在注入并行模式。ADC2 独立工作。
<code>ADC_DAUL_REGU_LAL_PARALLEL</code>	ADC0和ADC1工作在规则并行模式。ADC2 独立工作。
<code>ADC_DAUL_REGU_LAL_FOLLOW_UP</code>	ADC0和ADC1工作在跟随模式。ADC2 独立工作。
<code>ADC_DAUL_INSERTED_TRIGGER_ROTATION</code>	ADC0和ADC1工作在交替触发模式。ADC2 独立工作

<i>ADC_ALL_REGULAR_PARALLEL_INSERTED_PARALLEL</i>	所有的ADC都工作在规则并行和注入并行的组合模式
<i>ADC_ALL_REGULAR_PARALLEL_INSERTED_ROTATION</i>	所有的ADC都工作在规则并行和交替触发的组合模式
<i>ADC_ALL_INSERTED_PARALLEL</i>	所有的ADC都工作在注入并行模式
<i>ADC_ALL_REGULAR_PARALLEL</i>	所有的ADC都工作在规则并行模式
<i>ADC_ALL_REGULAR_FOLLOW_UP</i>	所有的ADC都工作在跟随模式
<i>ADC_ALL_INSERTED_TRIGGER_ROTATION</i>	所有的ADC都工作在交替触发模式
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* ADC0 and ADC1 work in combined regular parallel & inserted parallel mode */
adc_sync_mode_config (ADC_DAUL_REGULAR_PARALLEL_INSERTED_PARALLEL);
```

### 函数 **adc\_sync\_delay\_config**

函数adc\_sync\_delay\_config描述见下表：

**表 3-45. 函数 adc\_sync\_delay\_config**

函数名称	adc_sync_delay_config
函数原形	void adc_sync_delay_config(uint32_t sample_delay);
功能描述	配置ADC同步模式两次采样之间的延时
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	

<b>sample_delay</b>	采样阶段之间的延迟
<b>ADC_SYNC_DELAY_XCYCLE</b> (x=5..20)	配置两个采样阶段之间的延迟为x个时钟周期
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the delay between 2 sampling phases in ADC sync modes */

adc_sync_delay_config(ADC_SYNC_DELAY_5CYCLE);
```

### 函数 **adc\_sync\_dma\_config**

函数adc\_sync\_dma\_config描述见下表：

**表 3-46. 函数 adc\_sync\_dma\_config**

<b>函数名称</b>	adc_sync_dma_config
<b>函数原形</b>	void adc_sync_dma_config(uint32_t dma_mode );
<b>功能描述</b>	ADC同步DMA模式选择
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>dma_mode</b>	DMA模式
<b>ADC_SYNC_DMA_DISABLE</b>	ADC同步DMA失能
<b>ADC_SYNC_DMA_MODE0</b>	ADC同步DMA模式0
<b>ADC_SYNC_DMA_MODE1</b>	ADC同步DMA模式1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	

-	-
---	---

例如：

```
/* configure ADC sync DMA mode selection */

adc_sync_dma_config(ADC_SYNC_DMA_MODE0);
```

#### 函数 **adc\_sync\_dma\_request\_after\_last\_enable**

函数adc\_sync\_dma\_request\_after\_last\_enable描述见下表：

**表 3-47. 函数 adc\_sync\_dma\_request\_after\_last\_enable**

函数名称	adc_sync_dma_request_after_last_enable
函数原形	void adc_sync_dma_request_after_last_enable(void);
功能描述	当SYNCDMA不为00时，根据SYNCDMA位来产生DMA请求
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure ADC sync DMA engine is disabled after the end of transfer signal from DMA
controller is detected */

adc_sync_dma_request_after_last_enable();
```

#### 函数 **adc\_sync\_dma\_request\_after\_last\_disable**

函数adc\_sync\_dma\_request\_after\_last\_disable描述见下表：

**表 3-48. 函数 adc\_sync\_dma\_request\_after\_last\_disable**

函数名称	adc_sync_dma_request_after_last_disable
函数原形	void adc_sync_dma_request_after_last_disable(void);
功能描述	当检测到来自DMA控制器的传输结束信号后，DMA机制失能

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

例如：

```
/* the DMA engine is disabled after the end of transfer signal from DMA controller is detected
*/
adc_sync_dma_request_after_last_disable();
```

#### 函数 **adc\_sync\_regular\_data\_read**

函数adc\_sync\_regular\_data\_read描述见下表：

表 3-49. 函数 **adc\_sync\_regular\_data\_read**

函数名称	adc_sync_regular_data_read
函数原形	uint32_t adc_sync_regular_data_read(void);
功能描述	ADC同步模式规则数据读取
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>	返回同步模式的ADC规则通道值

例如：

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

---

```
adc_sync_regular_data_read();
```

### 3.3. 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[3.3.1](#)描述了CAN的寄存器列表，章节[3.3.2](#)对CAN库函数进行说明

#### 3.3.1. 外设寄存器说明

CAN寄存器列表如下表所示：

**表 3-50. 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_RFIFODAT A0x	接收FIFO邮箱data0寄存器
CAN_RFIFODAT A1x	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

寄存器名称	寄存器描述
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

### 3.3.2. 外设库函数说明

CAN库函数列表如下表所示：

**表 3-51. CAN 库函数**

库函数名称	库函数描述
can_deinit	复位外设CAN
can_struct_para_init	初始化结构体
can_init	初始化外设CAN
can_filter_init	CAN过滤器初始化
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
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

库函数名称	库函数描述
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 3-52. 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 3-53. 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

### 结构体 `can_receive_message_struct`

表 3-54. 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[8]	数据值
rx_fi	过滤器索引

### 结构体 `can_filter_parameter_struct`

表 3-55. 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

### 函数 can\_deinit

函数can\_deinit描述见下表:

**表 3-56. 函数 can\_deinit**

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

例如:

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

```
can_deinit (CAN0);
```

### 函数 can\_struct\_para\_init

函数can\_struct\_para\_init描述见下表:

**表 3-57. 函数 can\_struct\_para\_init**

函数名称	can_struct_para_init
函数原型	void can_struct_para_init(can_struct_type_enum type, void* p_struct)
功能描述	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, 仅可选择唯一参数

<i>CAN_INIT_STRUCT</i>	初始化结构体
<i>CAN_FILTER_STRUCT</i>	过滤器初始化结构体
<i>CAN_TX_MESSAGE_STRUCT</i>	存储发送帧结构体
<i>CAN_RX_MESSAGE_STRUCT</i>	接收帧结构体
<b>输出参数{out}</b>	
<i>p_struct</i>	对应的需要初始化的结构体指针
<b>返回值</b>	
-	-

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 3-58. 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	-
<b>输入参数{in}</b>	
<i>can_periph</i>	CAN 外设
<i>CANx(x=0,1)</i>	CAN外设选择
<b>输入参数{in}</b>	
<i>can_parameter_init</i>	初始化结构体，结构体成员参考 <a href="#">表 3-52. 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 3-59. 函数 can\_filter\_init

函数名称	can_filter_init
函数原型	void can_filter_init(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_filter_parameter_init	过滤器初始化结构体，结构体成员参考 <a href="#">表 3-55. 结构体 can_filter_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
can_filter_init(&can_filter);
```

### 函数 can1\_filter\_start\_bank

函数can1\_filter\_start\_bank描述见下表：

**表 3-60. 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 3-61. 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)	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 3-62. 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)	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 3-63. 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)	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 3-64. 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)	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 3-65. 函数 can\_message\_transmit**

函数名称	can_message_transmit
函数原型	uint8_t can_message_transmit(uint32_t can_periph, can_trasnmit_message_struct* transmit_message);
功能描述	CAN传输报文
先决条件	can_struct_para_init()
被调用函数	-
<b>输入参数{in}</b>	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择
<b>输入参数{in}</b>	
transmit_message	报文发送结构体，结构体成员参考 <a href="#">表 3-53. 结构体 can_trasnmit_message_struct</a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
uint8_t	0x00-0x03

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 3-66. 函数 can\_transmit\_states**

函数名称	can_transmit_states
函数原型	can_transmit_state_enum can_transmit_states(uint32_t can_periph, uint8_t mailbox_number);
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}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择
输入参数{in}	
mailbox_number	邮箱标号
CAN_MAILBOXx	CAN_MAILBOXx(x=0,1,2)
输出参数{out}	
-	-
返回值	
can_transmit_state_enum	0..4

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 3-67. 函数 can\_transmission\_stop**

函数名称	can_transmission_stop
函数原型	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>	-
<b>输入参数{in}</b>	
<b>can_periph</b>	CAN 外设
<b>CANx(x=0,1)</b>	CAN外设选择
<b>输入参数{in}</b>	
<b>mailbox_number</b>	邮箱标号
<b>CAN_MAILBOXx</b>	CAN_MAILBOXx(x=0,1,2)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-68. 函数 **can\_message\_receive**

<b>函数名称</b>	can_message_receive
<b>函数原型</b>	void can_message_receive(uint32_t can_periph, uint8_t fifo_number, can_receive_message_struct* receive_message);
<b>功能描述</b>	CAN接收报文
<b>先决条件</b>	can_struct_para_init()
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>can_periph</b>	CAN 外设
<b>CANx(x=0,1)</b>	CAN外设选择

输入参数{in}	
<b>fifo_number</b>	FIFO编号
<b>CAN_FIFOx</b>	CAN_FIFOx(x=0,1)
输入参数{in}	
<b>receive_message</b>	接收报文结构体, 结构体成员参考 <a href="#">表 3-54. 结构体 can_receive_message_struct</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 3-69. 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}	
<b>can_periph</b>	CAN 外设
<b>CANx(x=0,1)</b>	CAN外设选择
输入参数{in}	
<b>fifo_number</b>	FIFO编号
<b>CAN_FIFOx</b>	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 3-70. 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)	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 3-71. 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	-
<b>输入参数{in}</b>	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择
<b>输入参数{in}</b>	
can_working_mod e	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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
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 3-72. 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)	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 3-73. 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

<code>CANx(x=0,1)</code>	CAN外设选择
<b>输出参数{out}</b>	
-	-
返回值	
<code>can_error_enum</code>	0..7

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 3-74. 函数 `can_receive_error_number_get`

函数名称	<code>can_receive_error_number_get</code>
函数原型	<code>uint8_t can_receive_error_number_get(uint32_t can_periph);</code>
功能描述	获取CAN接收错误
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<code>can_periph</code>	CAN 外设
<code>CANx(x=0,1)</code>	CAN外设选择
<b>输出参数{out}</b>	
-	-
返回值	
<code>uint8_t</code>	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 3-75. 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)	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\_flag\_get

函数can\_flag\_get描述见下表:

表 3-76. 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}	

<b>can_periph</b>	CAN 外设
<b>CANx(x=0,1)</b>	CAN外设选择
<b>输入参数{in}</b>	
<b>flag</b>	CAN 标志位
<b>CAN_FLAG_MTE2</b>	邮箱2发送错误
<b>CAN_FLAG_MTE1</b>	邮箱1发送错误
<b>CAN_FLAG_MTE0</b>	邮箱0发送错误
<b>CAN_FLAG_MTF2</b>	邮箱2发送完成
<b>CAN_FLAG_MTF1</b>	邮箱1发送完成
<b>CAN_FLAG_MTF0</b>	邮箱0发送完成
<b>CAN_FLAG_RFO0</b>	接收FIFO0溢出
<b>CAN_FLAG_RFF0</b>	接收FIFO0满
<b>CAN_FLAG_RFO1</b>	接收FIFO1溢出
<b>CAN_FLAG_RFF1</b>	接收FIFO1满
<b>CAN_FLAG_BOER</b> <i>R</i>	离线错误
<b>CAN_FLAG_PERR</b>	被动错误
<b>CAN_FLAG_WERR</b>	警告错误
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	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 3-77. 函数 can\_flag\_clear

函数名称	can_flag_clear
函数原型	void can_flag_clear(uint32_t can_periph, can_flag_enum flag);
功能描述	清除CAN标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择
输入参数{in}	
flag	CAN 标志位
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FIFO溢出
CAN_FLAG_RFF0	接收FIFO满
CAN_FLAG_RFO1	接收FIFO溢出
CAN_FLAG_RFF1	接收FIFO满
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\_enable

函数can\_interrupt\_enable描述见下表：

**表 3-78. 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)	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_RF00	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_RF01	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 3-79. 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)	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_RF00	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_RF01	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

<b>CAN_INT_BO</b>	离线中断使能
<b>CAN_INT_ERRN</b>	错误种类中断使能
<b>CAN_INT_ERR</b>	错误中断使能
<b>CAN_INT_WU</b>	唤醒中断使能
<b>CAN_INT_SLPW</b>	睡眠中断使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

#### 函数 **can\_interrupt\_flag\_get**

函数can\_interrupt\_flag\_get描述见下表：

**表 3-80. 函数 can\_interrupt\_flag\_get**

<b>函数名称</b>	can_interrupt_flag_get
<b>函数原型</b>	FlagStatus can_interrupt_flag_get(uint32_t can_periph, can_interrupt_flag_enum flag);
<b>功能描述</b>	获取CAN中断标志位状态
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>can_periph</b>	CAN 外设
CANx(x=0,1)	CAN外设选择
<b>输入参数{in}</b>	
<b>flag</b>	CAN中断标志位
<b>CAN_INT_FLAG_SLPF</b>	进入睡眠工作模式的状态改变中断标志

<code>CAN_INT_FLAG_W UIF</code>	从睡眠工作模式唤醒的状态改变中断标志
<code>CAN_INT_FLAG_E RRIF</code>	错误中断标志
<code>CAN_INT_FLAG_M TF2</code>	邮箱2发送完成中断标志
<code>CAN_INT_FLAG_M TF1</code>	邮箱1发送完成中断标志
<code>CAN_INT_FLAG_M TF0</code>	邮箱0发送完成中断标志
<code>CAN_INT_FLAG_R FO0</code>	接收FIFO0溢出中断标志
<code>CAN_INT_FLAG_R FF0</code>	接收FIFO0满中断标志
<code>CAN_INT_FLAG_R FO1</code>	接收FIFO1溢出中断标志
<code>CAN_INT_FLAG_R FF1</code>	接收FIFO1满中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	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 3-81. 函数 `can_interrupt_flag_clear`

函数名称	<code>can_interrupt_flag_clear</code>
函数原型	<code>void can_interrupt_flag_clear(uint32_t can_periph, can_interrupt_flag_enum flag);</code>

<b>功能描述</b>	清除CAN中断标志位状态
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>can_periph</b>	CAN 外设
<b>CANx(x=0,1)</b>	CAN外设选择
<b>输入参数{in}</b>	
<b>flag</b>	CAN中断标志位
<b>CAN_INT_FLAG_S_LPIF</b>	进入睡眠工作模式的状态改变中断标志
<b>CAN_INT_FLAG_W_UIF</b>	从睡眠工作模式唤醒的状态改变中断标志
<b>CAN_INT_FLAG_E_RRIF</b>	错误中断标志
<b>CAN_INT_FLAG_M_TF2</b>	邮箱2发送完成中断标志
<b>CAN_INT_FLAG_M_TF1</b>	邮箱1发送完成中断标志
<b>CAN_INT_FLAG_M_TF0</b>	邮箱0发送完成中断标志
<b>CAN_INT_FLAG_R_FO0</b>	接收FIFO0溢出中断标志
<b>CAN_INT_FLAG_R_FF0</b>	接收FIFO0满中断标志
<b>CAN_INT_FLAG_R_FO1</b>	接收FIFO1溢出中断标志
<b>CAN_INT_FLAG_R_FF1</b>	接收FIFO1满中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

can_interrupt_flag_clear (CAN0, CAN_INT_FLAG_MTF0);
```

## 3.4. 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[3.4.1](#)描述了CRC的寄存器列表，章节[3.4.2](#)对CRC库函数进行说明。

### 3.4.1. 外设寄存器说明

CRC寄存器列表如下表所示：

**表 3-82. 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

### 3.4.2. 外设库函数说明

CRC库函数列表如下表所示：

**表 3-83. 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	复位数据寄存器，复位后的值为0xFFFFFFFF
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_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 3-84. 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 3-85. 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	复位数据寄存器，复位后的值为0xFFFFFFFF
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
crc_data_register_reset();
```

### 函数 **crc\_data\_register\_read**

函数crc\_data\_register\_read描述见下表：

**表 3-86. 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}	

-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<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 3-87. 函数 crc\_free\_data\_register\_read**

函数名称	crc_free_data_register_read	
函数原形	uint8_t crc_free_data_register_read(void);	
功能描述	读独立数据寄存器	
先决条件	-	
被调用函数	-	
<b>输入参数{in}</b>		
-	-	
<b>输出参数{out}</b>		
-	-	
<b>返回值</b>		
<b>uint8_t</b>	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 3-88. 函数 crc\_free\_data\_register\_write**

函数名称	crc_free_data_register_write	
函数原形	void crc_free_data_register_write(uint8_t free_data);	
功能描述	写独立数据寄存器	
先决条件	-	

被调用函数	-
<b>输入参数{in}</b>	
<b>free_data</b>	设定的8位数据
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

crc_free_data_register_write(0x11);
```

### 函数 **crc\_single\_data\_calculate**

函数crc\_single\_data\_calculate描述见下表：

**表 3-89. 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	-
<b>输入参数{in}</b>	
<b>sdata</b>	设定的32位数据
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint32_t</b>	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 3-90. 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	-
<b>输入参数{in}</b>	
<b>array</b>	32位数据数组的指针
<b>输入参数{in}</b>	
<b>size</b>	数据长度
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint32_t</b>	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);
```

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

### 3.5.1. 外设寄存器说明

CTC寄存器列表如下表所示：

表 3-91. 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

### 3.5.2. 外设库函数说明

CTC库函数列表如下表所示：

**表 3-92. 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_generator	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_usbsof_signal_select	CTC USB SOF 信号选择
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_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
ctc_flag_get	CTC状态标志获取
ctc_flag_clear	CTC状态标志清除

### 函数 **ctc\_deinit**

函数**ctc\_deinit**描述见下表：

**表 3-93. 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 3-94. 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 3-95. 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 3-96. 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 3-97. 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 3-98. 函数 `ctc.hardware.trim_mode_config`**

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

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 3-99. 函数 `ctc.refsource.polarity_config`**

函数名称	<code>ctc.refsource.polarity_config</code>
函数原形	<code>void ctc.refsource.polarity_config(uint32_t polarity);</code>

功能描述	CTC参考时钟极性配置
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>polarity</b>	时钟极性
<i>CTC_REFRESOURCE_POLARITY_FALLING</i>	参考信号源的同步极性为下降沿
<i>CTC_REFRESOURCE_POLARITY_RISING</i>	参考信号源的同步极性为上升沿
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

#### 函数 **ctc\_usbsof\_signal\_select**

函数**ctc\_usbsof\_signal\_select**描述见下表：

**表 3-100. 函数 ctc\_usbsof\_signal\_select**

函数名称	ctc_usbsof_signal_select
函数原形	void ctc_usbsof_signal_select(uint32_t usbsof);
功能描述	USB SOF 信号源选择
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>usbsof</b>	USB SOF 信号源
<i>CTC_USBSOFSEL_USBHS</i>	USBHS SOF被选为USB SOF 信号源

<b>CTC_USBSOFSEL_USBFS</b>	USBFS SOF被选为USB SOF 信号源
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* select USB SOF signal */
ctc_usbsof_signal_select (CTC_USBSOFSEL_USBFS);
```

#### 函数 **ctc\_refsource\_signal\_select**

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

**表 3-101. 函数 **ctc\_refsource\_signal\_select****

<b>函数名称</b>	ctc_refsource_signal_select
<b>函数原形</b>	void ctc_refsource_signal_select(uint32_t refs);
<b>功能描述</b>	CTC参考信号源选择
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>refs</b>	参考信号源
<b>CTC_REFSOURCE_GPIO</b>	选择GPIO输入信号
<b>CTC_REFSOURCE_LXTAL</b>	选择LXTAL时钟
<b>CTC_REFSOURCE_USBSOF</b>	选择USB SOF信号
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-102. 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 3-103. 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 3-104. 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	-
<b>输入参数{in}</b>	
reload_value	0x0000 - 0xFFFF
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-105. 函数 **ctc\_counter\_capture\_value\_read****

函数名称	ctc_counter_capture_value_read
函数原形	uint16_t ctc_counter_capture_value_read(void);
功能描述	读取计数器捕获值
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
uint16_t	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 3-106. 函数 ctc\_counter\_direction\_read**

函数名称	ctc_counter_direction_read
函数原形	FlagStatus ctc_counter_direction_read(void);
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	
<b>FlagStatus</b>	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 3-107. 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	
<b>uint16_t</b>	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 3-108. 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	
<b>uint8_t</b>	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_interrupt_enable`

函数`ctc_interrupt_enable`描述见下表：

**表 3-109. 函数 `ctc_interrupt_enable`**

函数名称	ctc_interrupt_enable
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}	
<code>interrupt</code>	CTC中断
<code>CTC_INT_CKOK</code>	时钟校准完成中断
<code>CTC_INT_CKWAR<sub>N</sub></code>	时钟校准警告中断
<code>CTC_INT_ERR</code>	错误中断
<code>CTC_INT_EREF</code>	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 3-110. 函数 `ctc_interrupt_disable`**

函数名称	ctc_interrupt_disable
函数原形	<code>void ctc_interrupt_disable(uint32_t interrupt);</code>
功能描述	禁能外设CTC中断

先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>interrupt</b>	CTC中断
<b>CTC_INT_CKOK</b>	时钟校准完成中断
<b>CTC_INT_CKWAR N</b>	时钟校准警告中断
<b>CTC_INT_ERR</b>	错误中断
<b>CTC_INT_EREF</b>	期望参考信号中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-111. 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	-
<b>输入参数{in}</b>	
<b>int_flag</b>	CTC中断标志
<b>CTC_INT_FLAG_C KOK</b>	时钟校准完成中断标志位

<i>CTC_INT_FLAG_C_KWARN</i>	时钟校准警告中断标志位
<i>CTC_INT_FLAG_E_RR</i>	错误中断标志位
<i>CTC_INT_FLAG_E_REF</i>	期望参考信号中断标志位
<i>CTC_INT_FLAG_C_KERR</i>	时钟校准错误位
<i>CTC_INT_FLAG_R_EFMISS</i>	参考同步脉冲信号丢失
<i>CTC_INT_FLAG_T_RIMERR</i>	校准值错误位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	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 3-112. 函数 **ctc\_interrupt\_flag\_clear**

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

<i>KOK</i>	
<i>CTC_INT_FLAG_C KWARN</i>	时钟校准警告中断标志位
<i>CTC_INT_FLAG_E RR</i>	错误中断标志位
<i>CTC_INT_FLAG_E REF</i>	期望参考信号中断标志位
<i>CTC_INT_FLAG_C KERR</i>	时钟校准错误位
<i>CTC_INT_FLAG_R EFMISS</i>	参考同步脉冲信号丢失
<i>CTC_INT_FLAG_T RIMERR</i>	校准值错误位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

#### 函数 **ctc\_flag\_get**

函数**ctc\_flag\_get**描述见下表：

**表 3-113. 函数 **ctc\_flag\_get****

函数名称	<b>ctc_flag_get</b>
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	-
<b>输入参数{in}</b>	
<b>flag</b>	CTC状态标志

<b>CTC_FLAG_CKOK</b>	时钟校准完成标志位
<b>CTC_FLAG_CKWA RN</b>	时钟校准警告中断标志位
<b>CTC_FLAG_ERR</b>	错误中断标志位
<b>CTC_FLAG_EREF</b>	期望参考信号中断标志位
<b>CTC_FLAG_CKER R</b>	时钟校准错误位
<b>CTC_FLAG_REFMI SS</b>	参考同步脉冲信号丢失
<b>CTC_FLAG_TRIME RR</b>	校准值错误位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	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 3-114. 函数 **ctc\_flag\_clear**

函数名称	ctc_flag_clear
函数原形	void ctc_flag_clear (uint32_t flag);
功能描述	清除CTC状态标志位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>flag</b>	CTC状态标志
<b>CTC_FLAG_CKOK</b>	时钟校准完成标志位

<i>CTC_FLAG_CKWA RN</i>	时钟校准警告中断标志位
<i>CTC_FLAG_ERR</i>	错误中断标志位
<i>CTC_FLAG_EREF</i>	期望参考信号中断标志位
<i>CTC_FLAG_CKER R</i>	时钟校准错误位
<i>CTC_FLAG_REFMI SS</i>	参考同步脉冲信号丢失
<i>CTC_FLAG_TRIME RR</i>	校准值错误位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* clear CTC flag status */

ctc_flag_clear (CTC_FLAG_CKOK);
```

## 3.6. 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[3.6.1](#)描述了DAC的寄存器列表，章节[3.6.2](#)对DAC库函数进行说明。

### 3.6.1. 外设寄存器说明

DAC寄存器列表如下表所示：

**表 3-115. DAC 寄存器**

寄存器名称	寄存器描述
DAC_CTL	DAC控制寄存器
DAC_SWT	DAC软件触发寄存器
DAC0_R12DH	DAC0 12位右对齐数据保持寄存器
DAC0_L12DH	DAC0 12位左对齐数据保持寄存器
DAC0_R8DH	DAC0 8位右对齐数据保持寄存器
DAC1_R12DH	DAC1 12位右对齐数据保持寄存器
DAC1_L12DH	DAC1 12位左对齐数据保持寄存器

寄存器名称	寄存器描述
DAC1_R8DH	DAC1 8位右对齐数据保持寄存器
DACC_R12DH	DAC 并发模式12位右对齐数据保持寄存器
DACC_L12DH	DAC并发模式12位左对齐数据保持寄存器
DACC_R8DH	DAC并发模式8位右对齐数据保持寄存器
DAC0_DO	DAC0 数据输出寄存器
DAC1_DO	DAC1 数据输出寄存器
DAC_STAT	DAC 状态寄存器

### 3.6.2. 外设库函数说明

DAC库函数列表如下表所示：

**表 3-116. 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_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触发源选择
dac_software_trigger_enable	DAC软件触发使能
dac_software_trigger_disable	DAC软件触发禁能
dac_wave_mode_config	DAC噪声波模式选择
dac_wave_bit_width_config	DAC噪声波位宽设置
dac_lfsr_noise_config	DAC LFSR模式设置
dac_triangle_noise_config	DAC三角波模式设置
dac_concurrent_enable	并发DAC模式使能
dac_concurrent_disable	并发DAC模式禁能
dac_concurrent_software_trigger_enable	并发DAC模式软件触发使能
dac_concurrent_software_trigger_disable	并发DAC模式软件触发禁能
dac_concurrent_output_buffer_enable	并发DAC模式输出缓冲功能使能
dac_concurrent_output_buffer_disable	并发DAC模式输出缓冲功能禁能
dac_concurrent_data_set	并发DAC模式输出数据设置
dac_concurrent_interrupt_enable	并发DAC模式中断使能

库函数名称	库函数描述
dac_concurrent_interrupt_disable	并发DAC模式中断禁能
dac_flag_get	获取DAC标志位（DAC DMA下溢标志位）
dac_flag_clear	清除DAC标志位（DAC DMA下溢标志位）
dac_interrupt_enable	DAC中断（DAC DMA下溢中断）使能
dac_interrupt_disable	DAC中断（DAC DMA下溢中断）禁能
dac_interrupt_flag_get	获取DAC中断标志位（DAC DMA下溢中断标志位）
dac_interrupt_flag_clear	清除DAC中断标志位（DAC DMA下溢中断标志位）

### 函数 **dac\_deinit**

函数dac\_deinit描述见下表：

**表 3-117. 函数 dac\_deinit**

函数名称	dac_deinit
函数原型	void dac_deinit(void);
功能描述	DAC外设复位
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：

```
/* deinitialize DAC */

dac_deinit();
```

### 函数 **dac\_enable**

函数dac\_enable描述见下表：

**表 3-118. 函数 dac\_enable**

函数名称	dac_enable
函数原型	void dac_enable(uint32_t dac_periph);
功能描述	DAC使能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx (x = 0,1)	DAC外设选择 (x =0,1)
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable DAC0 */

dac_enable(DAC0);
```

### 函数 **dac\_disable**

函数**dac\_disable**描述见下表：

**表 3-119. 函数 dac\_disable**

函数名称	dac_disable
函数原型	void dac_disable(uint32_t dac_periph);
功能描述	DAC禁能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx ( $x = 0, 1$ )	DAC外设选择 ( $x = 0, 1$ )
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DAC0 */

dac_disable(DAC0);
```

### 函数 **dac\_dma\_enable**

函数**dac\_dma\_enable**描述见下表：

**表 3-120. 函数 dac\_dma\_enable**

函数名称	dac_dma_enable
函数原型	void dac_dma_enable(uint32_t dac_periph);
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}	
dac_periph	DAC外设
DACx ( $x = 0, 1$ )	DAC外设选择 ( $x = 0, 1$ )

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC0 DMA function */

dac_dma_enable(DAC0);
```

### 函数 **dac\_dma\_disable**

函数dac\_dma\_disable描述见下表:

表 3-121. 函数 **dac\_dma\_disable**

函数名称	dac_dma_disable
函数原型	void dac_dma_disable(uint32_t dac_periph);
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}	
dac_periph	DAC外设
DACx (x = 0, 1)	DAC外设选择 (x =0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DAC0 DMA function */

dac_dma_disable(DAC0);
```

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

函数dac\_output\_buffer\_enable描述见下表:

表 3-122. 函数 **dac\_output\_buffer\_enable**

函数名称	dac_output_buffer_enable
函数原型	void dac_output_buffer_enable(uint32_t dac_periph);
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}	
dac_periph	DAC外设

<i>DACx (x = 0, 1)</i>	DAC外设选择 (x =0,1)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* enable DAC0 output buffer */

dac_output_buffer_enable(DAC0);
```

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

函数dac\_output\_buffer\_disable描述见下表:

**表 3-123. 函数 dac\_output\_buffer\_disable**

函数名称	dac_output_buffer_disable
函数原型	void dac_output_buffer_disable(uint32_t dac_periph);
功能描述	DAC输出缓冲区禁能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>dac_periph</b>	DAC外设
<i>DACx (x = 0, 1)</i>	DAC外设选择 (x =0,1)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* disable DAC0 output buffer */

dac_output_buffer_disable(DAC0);
```

### 函数 **dac\_output\_value\_get**

函数dac\_output\_value\_get描述见下表:

**表 3-124. 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);
功能描述	获取DAC输出数据
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	

<b>dac_periph</b>	DAC外设
<i>DACx (x = 0,1)</i>	DAC外设选择 (x =0,1)
<b>输出参数{out}</b>	
-	DAC输出数据
<b>返回值</b>	
<b>uint16_t</b>	外设DACx数据保持寄存器值 (0x000~0x7FF)

例如:

```
/* get the last data output value */
data = dac_output_value_get(DAC0);
```

### 函数 **dac\_data\_set**

函数dac\_data\_set描述见下表:

**表 3-125. 函数 dac\_data\_set**

函数名称	dac_data_set
函数原型	void dac_data_set(uint32_t dac_periph, 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	-
<b>输入参数{in}</b>	
<b>dac_periph</b>	DAC外设
<i>DACx (x = 0,1)</i>	DAC外设选择 (x =0,1)
<b>输入参数{in}</b>	
<b>dac_align</b>	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
<b>输入参数{in}</b>	
<b>data</b>	装载的数据
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* set DAC0 data holding register value */
dac_data_set(DAC0, DAC_ALIGN_8B_R, 0xff);
```

### 函数 **dac\_trigger\_enable**

函数**dac\_trigger\_enable**描述见下表：

**表 3-126. 函数 dac\_trigger\_enable**

函数名称	dac_trigger_enable	
函数原型	void dac_trigger_enable(uint32_t dac_periph);	
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 (x = 0, 1)</i>	DAC外设选择 (x =0,1)	
输出参数{out}		
-	-	
返回值		
-	-	

例如：

```
/* enable DAC0 trigger */

dac_trigger_enable(DAC0);
```

### 函数 **dac\_trigger\_disable**

函数**dac\_trigger\_disable**描述见下表：

**表 3-127. 函数 dac\_trigger\_disable**

函数名称	dac_trigger_disable	
函数原型	void dac_trigger_disable(uint32_t dac_periph);	
功能描述	DAC触发禁能	
先决条件	-	
被调用函数	-	
输入参数{in}		
<b>dac_periph</b>	DAC外设	
<i>DACx (x = 0, 1)</i>	DAC外设选择 (x =0,1)	
输出参数{out}		
-	-	
返回值		
-	-	

例如：

```
/* disable DAC0 trigger */

dac_trigger_disable(DAC0);
```

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

函数**dac\_trigger\_source\_config**描述见下表:

**表 3-128. 函数 **dac\_trigger\_source\_config****

函数名称	<b>dac_trigger_source_config</b>
函数原型	<code>void dac_trigger_source_config(uint32_t dac_periph,uint32_t triggersource);</code>
功能描述	DAC触发源选择
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>dac_periph</b>	DAC外设
<b>DACx (x = 0,1)</b>	DAC外设选择 (x =0,1)
<b>输入参数{in}</b>	
<b>triggersource</b>	DAC触发源
<b>DAC_TRIGGER_T1_TRGO</b>	TIMER1 TRGO
<b>DAC_TRIGGER_T3_TRGO</b>	TIMER3 TRGO
<b>DAC_TRIGGER_T4_TRGO</b>	TIMER4 TRGO
<b>DAC_TRIGGER_T5_TRGO</b>	TIMER5 TRGO
<b>DAC_TRIGGER_T6_TRGO</b>	TIMER6 TRGO
<b>DAC_TRIGGER_T7_TRGO</b>	TIMER7 TRGO
<b>DAC_TRIGGER_EXTI_9</b>	EXTI线 9中断
<b>DAC_TRIGGER_SOFTWARE</b>	软件触发
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

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

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

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

表 3-129. 函数 **dac\_software\_trigger\_enable**

函数名称	dac_software_trigger_enable
函数原型	void dac_software_trigger_enable(uint32_t dac_periph);
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}	
dac_periph	DAC外设
DACx ( $x = 0, 1$ )	DAC外设选择 ( $x = 0, 1$ )
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数dac\_software\_trigger\_disable描述见下表:

 表 3-130. 函数 **dac\_software\_trigger\_disable**

函数名称	dac_software_trigger_disable
函数原型	void dac_software_trigger_disable(uint32_t dac_periph);
功能描述	DAC软件触发禁能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx ( $x = 0, 1$ )	DAC外设选择 ( $x = 0, 1$ )
输出参数{out}	
-	-
返回值	
-	-

例如:

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

### 函数 **dac\_wave\_mode\_config**

函数dac\_wave\_mode\_config描述见下表:

表 3-131. 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, uint32_t wave_mode);</code>
功能描述	DAC噪声波模式选择
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx (x = 0,1)</code>	DAC外设选择 ( $x = 0,1$ )
输入参数{in}	
<code>wave_mode</code>	噪声波模式选择
<code>DAC_WAVE_DISABLE</code>	噪声波模式禁能
<code>DAC_WAVE_MODE_LFSR</code>	LFSR噪声波模式
<code>DAC_WAVE_MODE_TRIANGLE</code>	三角波噪声波模式
输出参数{out}	
-	-
返回值	
-	-

例如:

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

### 函数 `dac_wave_bit_width_config`

函数`dac_wave_bit_width_config`描述见下表:

 表 3-132. 函数 `dac_wave_bit_width_config`

函数名称	<code>dac_wave_bit_width_config</code>
函数原型	<code>void dac_wave_bit_width_config(uint32_t dac_periph, uint32_t bit_width);</code>
功能描述	DAC噪声波位宽设置
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx (x = 0,1)</code>	DAC外设选择 ( $x = 0,1$ )
输入参数{in}	
<code>bit_width</code>	噪声波位宽
<code>DAC_WAVE_BIT_WIDTH_X</code>	噪声波位宽 ( $x = 1..12$ )

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DAC0 wave bit width */
dac_wave_bit_width_config(DAC0, DAC_WAVE_BIT_WIDTH_1);
```

### 函数 **dac\_lfsr\_noise\_config**

函数dac\_lfsr\_noise\_config描述见下表:

表 3-133. 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, uint32_t unmask_bits);
功能描述	DAC LFSR模式设置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx (x = 0, 1)	DAC外设选择 (x =0,1)
输入参数{in}	
unmask_bits	噪声波的非屏蔽位宽
DAC_LFSR_BIT0	LFSR模式位0非屏蔽
DAC_LFSR_BITSx_0	LFSR模式位[x:0]非屏蔽
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DAC0 LFSR noise mode */
dac_lfsr_noise_config(DAC0, DAC_LFSR_BIT0);
```

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

函数dac\_triangle\_noise\_config描述见下表:

表 3-134. 函数 **dac\_triangle\_noise\_config**

函数名称	dac_triangle_noise_config
函数原型	void dac_triangle_noise_config(uint32_t dac_periph, uint32_t amplitude);

功能描述	DAC三角波模式设置
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>dac_periph</b>	DAC外设
<i>DACx (x = 0, 1)</i>	DAC外设选择 (x =0,1)
<b>输入参数{in}</b>	
<b>amplitude</b>	DAC三角波噪声幅值
<i>DAC_TRIANGLE_AMPLITUDE_x</i>	三角波幅值为x = 2 <sup>n</sup> -1 (n = 1..12)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* configure DAC0 triangle noise mode */

dac_triangle_noise_config(DAC0, DAC_TRIANGLE_AMPLITUDE_1);
```

#### 函数 **dac\_concurrent\_enable**

函数**dac\_concurrent\_enable**描述见下表:

**表 3-135. 函数 dac\_concurrent\_enable**

函数名称	dac_concurrent_enable
函数原型	void dac_concurrent_enable (void);
功能描述	并发DAC模式使能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* enable DAC concurrent mode */

dac_concurrent_enable();
```

#### 函数 **dac\_concurrent\_disable**

函数**dac\_concurrent\_disable**描述见下表:

表 3-136. 函数 **dac\_concurrent\_disable**

函数名称	dac_concurrent_disable
函数原型	void dac_concurrent_disable(void);
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}	
	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DAC concurrent mode */

dac_concurrent_disable();
```

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

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

 表 3-137. 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 (void);
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}	
	-
输出参数{out}	
-	-
返回值	
-	-

例如:

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

dac_concurrent_software_trigger_enable();
```

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

函数dac\_concurrent\_software\_trigger\_disable描述见下表:

表 3-138. 函数 **dac\_concurrent\_software\_trigger\_disable**

函数名称	dac_concurrent_software_trigger_disable
函数原型	void dac_concurrent_software_trigger_disable (void);
功能描述	并发DAC模式软件触发禁能
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 DAC concurrent software trigger */

dac_concurrent_software_trigger_disable();
```

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

函数dac\_concurrent\_output\_buffer\_enable描述见下表:

 表 3-139. 函数 **dac\_concurrent\_output\_buffer\_enable**

函数名称	dac_concurrent_output_buffer_enable
函数原型	void dac_concurrent_output_buffer_enable(void);
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}	
	-
输出参数{out}	
-	-
返回值	
-	-

例如:

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

dac_concurrent_output_buffer_enable();
```

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

函数dac\_concurrent\_output\_buffer\_disable描述见下表:

表 3-140. 函数 `dac_concurrent_output_buffer_disable`

函数名称	dac_concurrent_output_buffer_disable
函数原型	void dac_concurrent_output_buffer_disable(void);
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}	
	-
输出参数{out}	
-	-
返回值	
-	-

例如：

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

dac_concurrent_output_buffer_disable();
```

#### 函数 `dac_concurrent_data_set`

函数 `dac_concurrent_data_set` 描述见下表：

 表 3-141. 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_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_align</b>	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DAC0的数据
输入参数{in}	
<b>data1</b>	写入DAC1的数据
输出参数{out}	
-	-
返回值	
-	-

例如：

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

dac_concurrent_data_set(DAC_ALIGN_8B_R, 0xff, 0xff);
```

### **函数 dac\_concurrent\_interrupt\_enable**

函数dac\_concurrent\_interrupt\_enable描述见下表：

**表 3-142. 函数 dac\_concurrent\_interrupt\_enable**

函数名称	dac_concurrent_interrupt_enable
函数原形	void dac_concurrent_interrupt_enable(void);
功能描述	并发DAC模式中断使能
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 DAC concurrent interrupt funcution */

dac_concurrent_interrupt_enable();
```

### **函数 dac\_concurrent\_interrupt\_disable**

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

**表 3-143. 函数 dac\_concurrent\_interrupt\_disable**

函数名称	dac_concurrent_interrupt_disable
函数原形	void dac_concurrent_interrupt_disable(void);
功能描述	并发DAC模式中断禁能
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 DAC concurrent interrupt funcution */
```

```
dac_concurrent_interrupt_disable();
```

### 函数 **dac\_flag\_get**

函数**dac\_flag\_get**描述见下表:

**表 3-144. 函数 dac\_flag\_get**

函数名称	dac_flag_get
函数原形	FlagStatus dac_flag_get(uint32_t dac_periph);
功能描述	获取DAC标志位（DAC DMA下溢标志位）
先决条件	-
被调用函数	-
输入参数{in}	
<b>dac_periph</b>	DAC外设
<i>DACx (x = 0, 1)</i>	DAC外设选择 (x =0, 1)
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或 RESET

例如:

```
/* get the DAC0 flag bit */

FlagStatus flag_value;

flag_value = dac_flag_get(DAC0);
```

### 函数 **dac\_flag\_clear**

函数**dac\_flag\_clear**描述见下表:

**表 3-145. 函数 dac\_flag\_clear**

函数名称	dac_flag_clear
函数原形	void dac_flag_clear(uint32_t dac_periph);
功能描述	清除DAC标志位（DAC DMA下溢标志位）
先决条件	-
被调用函数	-
输入参数{in}	
<b>dac_periph</b>	DAC外设
<i>DACx (x = 0, 1)</i>	DAC外设选择 (x =0, 1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the DAC0 flag bit */

dac_flag_clear(DAC0);
```

### 函数 **dac\_interrupt\_enable**

函数**dac\_interrupt\_enable**描述见下表：

**表 3-146. 函数 dac\_interrupt\_enable**

函数名称	dac_interrupt_enable
函数原形	void dac_interrupt_enable(uint32_t dac_periph);
功能描述	DAC中断（DAC DMA下溢中断）使能
先决条件	-
被调用函数	-
输入参数{in}	
<b>dac_periph</b>	DAC外设
<i>DACx (x = 0,1)</i>	DAC外设选择 (x =0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DAC0 interrupt */

dac_interrupt_enable(DAC0);
```

### 函数 **dac\_interrupt\_disable**

函数**dac\_interrupt\_disable**描述见下表：

**表 3-147. 函数 dac\_interrupt\_disable**

函数名称	dac_interrupt_disable
函数原形	void dac_interrupt_disable(uint32_t dac_periph);
功能描述	DAC中断（DAC DMA下溢中断）禁能
先决条件	-
被调用函数	-
输入参数{in}	
<b>dac_periph</b>	DAC外设
<i>DACx (x = 0,1)</i>	DAC外设选择 (x =0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DAC0 interrupt */

dac_interrupt_disable(DAC0);
```

### 函数 **dac\_interrupt\_flag\_get**

函数dac\_interrupt\_flag\_get描述见下表：

**表 3-148. 函数 dac\_interrupt\_flag\_get**

函数名称	dac_interrupt_flag_get
函数原形	FlagStatus dac_interrupt_flag_get(uint32_t dac_periph);
功能描述	获取DAC中断标志位（DAC DMA下溢中断标志位）
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx ( $x = 0, 1$ )	DAC外设选择 ( $x = 0, 1$ )
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the DAC0 interrupt flag bit */

FlagStatus flag_value;

flag_value = dac_interrupt_flag_get(DAC0);
```

### 函数 **dac\_interrupt\_flag\_clear**

函数dac\_interrupt\_flag\_clear描述见下表：

**表 3-149. 函数 dac\_interrupt\_flag\_clear**

函数名称	dac_interrupt_flag_clear
函数原形	void dac_interrupt_flag_clear(uint32_t dac_periph);
功能描述	清除DAC中断标志位（DAC0DMA下溢中断标志位）
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx ( $x = 0, 1$ )	DAC外设选择 ( $x = 0, 1$ )
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* clear the DAC0 interrupt flag bit */
dac_interrupt_flag_clear(DAC0);
```

## 3.7. **DBG**

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

### 3.7.1. 外设寄存器说明

DBG寄存器列表如下表所示：

**表 3-150. DBG 寄存器**

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

### 3.7.2. 外设库函数说明

DBG库函数列表如下表所示：

**表 3-151. DBG 库函数**

库函数名称	库函数描述
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
dbg_trace_pin_mode_set	配置跟踪引脚分配模式

**枚举类型 `dbg_periph_enum`**
**表 3-152. 枚举类型 `dbg_periph_enum`**

成员名称	功能描述
<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_TIMER0_HOLD</code>	当内核停止时，保持TIMER0计数器计数值不变
<code>DBG_TIMER1_HOLD</code>	当内核停止时，保持TIMER1计数器计数值不变
<code>DBG_TIMER2_HOLD</code>	当内核停止时，保持TIMER2计数器计数值不变
<code>DBG_TIMER3_HOLD</code>	当内核停止时，保持TIMER3计数器计数值不变
<code>DBG_CAN0_HOLD</code>	当内核停止时，CAN0接收寄存器停止接收数据
<code>DBG_I2C0_HOLD</code>	当内核停止时，保持I2C0的SMBUS状态不变，用于调试
<code>DBG_I2C1_HOLD</code>	当内核停止时，保持I2C1的SMBUS状态不变，用于调试
<code>DBG_TIMER4_HOLD</code>	当内核停止时，保持TIMER4计数器计数值不变
<code>DBG_TIMER5_HOLD</code>	当内核停止时，保持TIMER5计数器计数值不变
<code>DBG_TIMER6_HOLD</code>	当内核停止时，保持TIMER6计数器计数值不变
<code>DBG_TIMER7_HOLD</code>	当内核停止时，保持TIMER7计数器计数值不变
<code>DBG_CAN1_HOLD</code>	当内核停止时，CAN1接收寄存器停止接收数据
<code>DBG_TIMER11_HOLD</code>	当内核停止时，保持TIMER11计数器计数值不变
<code>DBG_TIMER12_HOLD</code>	当内核停止时，保持TIMER12计数器计数值不变
<code>DBG_TIMER13_HOLD</code>	当内核停止时，保持TIMER13计数器计数值不变
<code>DBG_TIMER8_HOLD</code>	当内核停止时，保持TIMER8计数器计数值不变
<code>DBG_TIMER9_HOLD</code>	当内核停止时，保持TIMER9计数器计数值不变
<code>DBG_TIMER10_HOLD</code>	当内核停止时，保持TIMER10计数器计数值不变

**函数 `dbg_id_get`**

函数`dbg_id_get`描述见下表：

**表 3-153. 函数 `dbg_id_get`**

函数名称	<code>dbg_id_get</code>
函数原形	<code>uint32_t dbg_id_get(void);</code>

功能描述	Read DBG_ID code register
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 3-154. 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}	
<b>dbg_low_power</b>	低功耗模式调试保持
<b>DBG_LOW_POWER_R_SLEEP</b>	在睡眠模式下，保持调试器连接，可进行调试
<b>DBG_LOW_POWER_R_DEEPSLEEP</b>	在深度睡眠模式下，保持调试器连接，可进行调试
<b>DBG_LOW_POWER_R_STANDBY</b>	在待机模式下，保持调试器连接，可进行调试

输出参数{out}	
-	-
返回值	
-	-

例如：

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

#### 函数 **dbg\_low\_power\_disable**

函数dbg\_low\_power\_disable描述见下表：

表 3-155. 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
<b>DBG_LOW_POWER_R_SLEEP</b>	在睡眠模式下，保持调试器连接，可进行调试
<b>DBG_LOW_POWER_R_DEEPSLEEP</b>	在深度睡眠模式下，保持调试器连接，可进行调试
<b>DBG_LOW_POWER_R_STANDBY</b>	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：

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

```
dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

### 函数 **dbg\_periph\_enable**

函数**dbg\_periph\_enable**描述见下表:

**表 3-156. 函数 **dbg\_periph\_enable****

函数名称	dbg_periph_enable
函数原形	void dbg_periph_enable(dbg_periph_enum dbg_periph);
功能描述	使能外设的MCU调试保持功能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>dbg_periph</b>	Peripheral refer to <a href="#">表 3-152. 枚举类型<b>dbg_periph_enum</b></a>
<i>DBG_FWDGT_HOLD</i>	当内核停止时，保持FWDGT计数器时钟
<i>DBG_WWDGT_HOLD</i>	当内核停止时，保持WWDGT计数器时钟
<i>DBG_CANx_HOLD</i>	当内核停止时，CANx接收寄存器停止接收数据
<i>DBG_I2Cx_HOLD</i>	当内核停止时，保持I2Cx (x=0,1) 的SMBUS状态不变，用于调试
<i>DBG_TIMERx_HOLD</i>	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)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

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

### 函数 **dbg\_periph\_disable**

函数**dbg\_periph\_disable**描述见下表:

表 3-157. 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>	Peripheral refer to <a href="#">表 3-152. 枚举类型<code>dbg_periph_enum</code></a>
<code>DBG_FWDGT_HOLD_D</code>	当内核停止时，保持FWDGT计数器时钟
<code>DBG_WWDGT_HOLD_LD</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接收寄存器停止接收数据（x=0,1）
<code>DBG_I2Cx_HOLD</code>	当内核停止时，保持I2Cx（x=0,1）的SMBUS状态不变，用于调试
<code>DBG_TIMERx_HOLD_D</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)
输出参数{out}	
-	-
返回值	
-	-

例如：

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

dbg_periph_disable(DBG_TIMER0_HOLD);
```

### 函数 `dbg_trace_pin_enable`

函数`dbg_trace_pin_enable`描述见下表：

 表 3-158. 函数 `dbg_trace_pin_enable`

函数名称	<code>dbg_trace_pin_enable</code>
函数原形	<code>void dbg_trace_pin_enable(void);</code>
功能描述	使能跟踪引脚分配

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

例如：

```
/* 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 3-159. 函数 **dbg\_trace\_pin\_disable**

函数名称	dbg_trace_pin_disable
函数原形	void dbg_trace_pin_disable(void);
功能描述	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();
```

### 函数 `dbg_trace_pin_mode_set`

函数`dbg_trace_pin_mode_set`描述见下表：

**表 3-160. 函数 `dbg_trace_pin_mode_set`**

函数名称	dbg_trace_pin_mode_set
函数原形	<code>void dbg_trace_pin_mode_set(uint32_t trace_mode);</code>
功能描述	配置跟踪引脚分配模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>trace_mode</code>	跟踪引脚分配模式选择
<code>TRACE_MODE_ASYNC</code>	跟踪引脚用于异步模式
<code>TRACE_MODE_SYNC_NC_DATASIZE_1</code>	跟踪引脚用于同步模式且数据长度为1
<code>TRACE_MODE_SYNC_NC_DATASIZE_2</code>	跟踪引脚用于同步模式且数据长度为2
<code>TRACE_MODE_SYNC_NC_DATASIZE_4</code>	跟踪引脚用于同步模式且数据长度为4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* trace pin mode selection */

dbg_trace_pin_mode_set(TRACE_MODE_ASYNC);
```

## 3.8. DCI

数字摄像头接口是一个同步并行接口，可以从数字摄像头捕获视频和图像信息。DCI寄存器列举在章节[3.8.1](#)，DCI固件库函数列举在章节[3.8.2](#)。

### 3.8.1. 外设寄存器说明

DCI寄存器列表如下表所示：

**表 3-161. DCI 寄存器**

寄存器名称	寄存器描述
DCI_CTL	DCI控制寄存器
DCI_STAT0	DCI状态寄存器0
DCI_STAT1	DCI状态寄存器1
DCI_INTEN	DCI中断使能寄存器
DCI_INTF	DCI中断标志寄存器
DCI_INTC	DCI中断标志清除寄存器
DCI_SC	DCI同步码寄存器
DCI_SCUMSK	DCI同步码屏蔽寄存器
DCI_CWSPOS	DCI裁剪窗口起始位置寄存器
DCI_CWSZ	DCI裁剪窗口大小寄存器
DCI_DATA	DCI数据寄存器

### 3.8.2. 外设库函数说明

DCI库函数列表如下表所示：

**表 3-162. DCI 库函数**

库函数名称	库函数描述
dci_deinit	复位DCI
dci_init	初始化DCI寄存器
dci_enable	使能DCI功能
dci_disable	除能DCI功能
dci_capture_enable	使能DCI捕获功能
dci_capture_disable	除能DCI捕获功能
dci_jpeg_enable	使能DCI JPEG模式
dci_jpeg_disable	除能DCI JPEG模式

库函数名称	库函数描述
dci_crop_window_enable	使能裁剪窗口功能
dci_crop_window_disable	除能裁剪窗口功能
dci_crop_window_config	配置DCI裁剪窗口功能
dci_embedded_sync_enable	使能内嵌同步模式
dci_embedded_sync_disable	除能内嵌同步模式
dci_sync_codes_config	在内嵌同步模式下配置同步码
dci_sync_codes_unmask_config	在内嵌同步模式下配置非屏蔽同步码
dci_data_read	读取DCI数据寄存器
dci_flag_get	获取指定标志
dci_interrupt_enable	使能指定的DCI中断
dci_interrupt_enable	除能指定的DCI中断
dci_interrupt_flag_get	获取指定的中断标志
dci_interrupt_flag_clear	清除指定的中断标志

### 结构体 **dci\_parameter\_struct**

表 3-163. 结构体 **dci\_parameter\_struct**

成员名称	功能描述
capture_mode	DCI获取模式: DCI_CAPTURE_MODE_CONTINUOUS / DCI_CAPTURE_MODE_SNAPSHOT
clock_polarity	时钟极性选择: DCI_CK_POLARITY_FALLING / DCI_CK_POLARITY_RISING
hsync_polarity	水平极性选择: DCI_HSYNC_POLARITY_LOW / DCI_HSYNC_POLARITY_HIGH
vsync_polarity	垂直极性选择: DCI_VSYNC_POLARITY_LOW / DCI_VSYNC_POLARITY_HIGH
frame_rate	帧获取速率: DCI_FRAME_RATE_ALL / DCI_FRAME_RATE_1_2 / DCI_FRAME_RATE_1_4
interface_format	数码相机接口格式: DCI_INTERFACE_FORMAT_8BITS / DCI_INTERFACE_FORMAT_10BITS / DCI_INTERFACE_FORMAT_12BITS /

	DCI_INTERFACE_FORMAT_14BITS
--	-----------------------------

### 函数 **dci\_deinit**

函数dci\_deinit描述见下表:

**表 3-164. 函数 dci\_deinit**

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

例如:

```
/* DCI deinit */
dci_deinit();
```

### 函数 **dci\_init**

函数dci\_init描述见下表:

**表 3-165. 函数 dci\_init**

函数名称	dci_init
函数原形	void dci_init(dci_parameter_struct* dci_struct);
功能描述	初始化DCI寄存器
先决条件	-
被调用函数	-
输入参数{in}	
dci_struct	DCI参数初始化结构体

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize DCI registers */

dci_parameter_struct  dci_struct;

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

### 函数 **dci\_enable**

函数dci\_enable描述见下表：

表 3-166. 函数 **dci\_enable**

函数名称	dci_enable
函数原形	void dci_enable(void);
功能描述	使能DCI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 DCI function */

dci_enable( );
```

### **函数 dci\_disable**

函数dci\_disable描述见下表：

**表 3-167. 函数 dci\_disable**

函数名称	dci_disable
函数原形	void dci_disable(void);
功能描述	disable DCI function
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 DCI function */

dci_disable( );
```

### **函数 dci\_capture\_enable**

函数dci\_capture\_enable描述见下表：

**表 3-168. 函数 dci\_capture\_enable**

函数名称	dci_capture_enable
函数原形	void dci_capture_enable(void);
功能描述	使能DCI捕获功能
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 DCI capture */

dci_capture_enable();
```

#### 函数 **dci\_capture\_disable**

函数dci\_capture\_disable描述见下表：

**表 3-169. 函数 dci\_capture\_disable**

函数名称	dci_capture_disable
函数原形	void dci_capture_disable(void);
功能描述	除能DCI捕获功能
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 DCI capture */

dci_capture_disable();
```

#### 函数 **dci\_jpeg\_enable**

函数dci\_jpeg\_enable描述见下表：

表 3-170. 函数 dci\_jpeg\_enable

函数名称	dci_jpeg_enable
函数原形	void dci_jpeg_enable(void);
功能描述	使能DCI JPEG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 DCI jpeg mode */
dci_jpeg_enable();
```

#### 函数 dci\_jpeg\_disable

函数dci\_jpeg\_disable描述见下表：

表 3-171. 函数 dci\_jpeg\_disable

函数名称	dci_jpeg_disable
函数原形	void dci_jpeg_disable(void);
功能描述	除能DCI JPEG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 DCI jpeg mode */

dci_jpeg_disable();
```

#### 函数 **dci\_crop\_window\_enable**

函数dci\_crop\_window\_enable描述见下表：

表 3-172. 函数 **dci\_crop\_window\_enable**

函数名称	dci_crop_window_enable
函数原形	void dci_crop_window_enable(void);
功能描述	使能裁剪窗口功能
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 cropping window function */

dci_crop_window_enable();
```

#### 函数 **dci\_crop\_window\_disable**

函数dci\_crop\_window\_disable描述见下表：

表 3-173. 函数 **dci\_crop\_window\_disable**

函数名称	dci_crop_window_disable
函数原形	void dci_crop_window_disable(void);
功能描述	除能裁剪窗口功能
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 cropping window function */
dci_crop_window_disable( );
```

#### 函数 **dci\_crop\_window\_config**

函数dci\_crop\_window\_config描述见下表：

表 3-174. 函数 **dci\_crop\_window\_config**

函数名称	dci_crop_window_config
函数原形	void dci_crop_window_config(uint16_t start_x, uint16_t start_y, uint16_t size_width, uint16_t size_height);
功能描述	配置DCI裁剪窗口功能
先决条件	-
被调用函数	-
输入参数{in}	
start_x	窗口水平起始地址
输入参数{in}	
start_y	窗口垂直起始地址
输入参数{in}	
size_width	窗口水平长度大小
输入参数{in}	
size_height	窗口垂直长度大小
输出参数{out}	

-	-
返回值	
-	-

例如：

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

#### 函数 **dci\_embedded\_sync\_enable**

函数dci\_embedded\_sync\_enable描述见下表：

**表 3-175. 函数 dci\_embedded\_sync\_enable**

函数名称	dci_embedded_sync_enable
函数原形	void dci_embedded_sync_enable(void);
功能描述	使能内嵌同步模式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
返回值	
-	-

例如：

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

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

函数dci\_embedded\_sync\_disable描述见下表：

**表 3-176. 函数 dci\_embedded\_sync\_disable**

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

功能描述	除能内嵌同步模式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

### 函数 **dci\_sync\_codes\_config**

函数dci\_sync\_codes\_config描述见下表：

表 3-177. 函数 **dci\_sync\_codes\_config**

函数名称	dci_sync_codes_config
函数原形	void dci_sync_codes_config(uint8_t frame_start, uint8_t line_start, uint8_t line_end, uint8_t frame_end);
功能描述	在内嵌同步模式下配置同步码
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
frame_start	在内嵌同步模式帧起始码
<b>输入参数{in}</b>	
line_start	在内嵌同步模式行起始码
<b>输入参数{in}</b>	
line_end	在内嵌同步模式行终止码
<b>输入参数{in}</b>	

<b>frame_end</b>	在内嵌同步模式帧终止码
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

### 函数 **dci\_sync\_codes\_unmask\_config**

函数dci\_sync\_codes\_unmask\_config描述见下表：

表 3-178. 函数 **dci\_sync\_codes\_unmask\_config**

函数名称	dci_sync_codes_unmask_config
函数原形	void dci_sync_codes_unmask_config(uint8_t frame_start, uint8_t line_start, uint8_t line_end, uint8_t frame_end);
功能描述	在内嵌同步模式下配置非屏蔽同步码
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>frame_start</b>	在内嵌同步模式帧起始码
<b>输入参数{in}</b>	
<b>line_start</b>	在内嵌同步模式行起始码
<b>输入参数{in}</b>	
<b>line_end</b>	在内嵌同步模式行终止码
<b>输入参数{in}</b>	
<b>frame_end</b>	在内嵌同步模式帧终止码
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	

-	-
---	---

例如：

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

### 函数 **dci\_data\_read**

函数dci\_data\_read描述见下表：

**表 3-179. 函数 dci\_data\_read**

函数名称	dci_data_read
函数原形	uint32_t dci_data_read(void);
功能描述	读取DCI数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	0x00 – 0xFFFFFFFF

例如：

```
/* read DCI data register */

uint32_t data = dci_data_read( );
```

### 函数 **dci\_flag\_get**

函数dci\_flag\_get描述见下表：

**表 3-180. 函数 dci\_flag\_get**

函数名称	dci_flag_get
函数原形	FlagStatus dci_flag_get(uint32_t flag);
功能描述	获取指定标志
先决条件	-

<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>flag</b>	DCI标志
DCI_FLAG_HS	水平同步行状态
DCI_FLAG_VS	水平同步行状态
DCI_FLAG_FV	FIFO有效
DCI_FLAG_EF	帧结束
DCI_FLAG_OVR	FIFO溢出
DCI_FLAG_ESE	内嵌同步错误
DCI_FLAG_VSYNC	垂直同步
DCI_FLAG_EL	行结束
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
FlagStatus	SET或RESET

例如：

```
/* get specified flag */
FlagStatus status = dci_flag_get (DCI_FLAG_HS);
```

### 函数 **dci\_interrupt\_enable**

函数dci\_interrupt\_enable描述见下表：

**表 3-181. 函数 dci\_interrupt\_enable**

<b>函数名称</b>	dci_interrupt_enable
<b>函数原形</b>	void dci_interrupt_enable(uint32_t interrupt);
<b>功能描述</b>	使能指定的DCI中断
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>interrupt</b>	DCI中断

<i>DCI_INT_EF</i>	帧结束中断标志
<i>DCI_INT_OVR</i>	FIFO溢出中断标志
<i>DCI_INT_ESE</i>	内嵌码同步错误中断标志
<i>DCI_INT_VSYNC</i>	垂直同步中断标志
<i>DCI_INT_EL</i>	行结束中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable specified DCI interrupt */

dci_interrupt_enable (DCI_INT_EF);
```

#### 函数 **dci\_interrupt\_disable**

函数dci\_interrupt\_disable描述见下表：

表 3-182. 函数 **dci\_interrupt\_disable**

函数名称	dci_interrupt_disable
函数原形	void dci_interrupt_disable(uint32_t interrupt);
功能描述	除能指定的DCI中断
先决条件	-
被调用函数	-
输入参数{in}	
<b>interrupt</b>	DCI interrupt
<i>DCI_INT_EF</i>	帧结束中断标志
<i>DCI_INT_OVR</i>	FIFO溢出中断标志
<i>DCI_INT_ESE</i>	内嵌码同步错误中断标志
<i>DCI_INT_VSYNC</i>	垂直同步中断标志
<i>DCI_INT_EL</i>	行结束中断标志
输出参数{out}	

-	-
返回值	
-	-

例如：

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

#### 函数 **dci\_interrupt\_flag\_get**

函数dci\_interrupt\_flag\_get描述见下表：

**表 3-183. 函数 dci\_interrupt\_flag\_get**

函数名称	dci_interrupt_flag_get
函数原形	FlagStatus dci_interrupt_flag_get(uint32_t interrupt);
功能描述	获取指定的中断标志
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
interrupt	DCI中断
DCI_INT_FLAG_EF	帧结束中断标志
DCI_INT_FLAG_OVR	FIFO溢出中断标志
DCI_INT_FLAG_ESE	内嵌码同步错误中断标志
DCI_INT_FLAG_VSYN C	垂直同步中断标志
DCI_INT_FLAG_EL	行结束中断标志
<b>输出参数{out}</b>	
-	-
返回值	
FlagStatus	SET或RESET

例如：

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

### 函数 `dci_interrupt_flag_clear`

函数`dci_interrupt_flag_clear`描述见下表：

**表 3-184. 函数 `dci_interrupt_flag_clear`**

函数名称	<code>dci_interrupt_flag_clear</code>
函数原形	<code>void dci_interrupt_flag_clear(uint32_t interrupt);</code>
功能描述	清除指定的中断标志
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<code>interrupt</code>	DCI中断
<code>DCI_INT_FLAG_EF</code>	帧结束中断标志
<code>DCI_INT_FLAG_OVR</code>	FIFO溢出中断标志
<code>DCI_INT_FLAG_ESE</code>	内嵌码同步错误中断标志
<code>DCI_INT_FLAG_VSYN_C</code>	垂直同步中断标志
<code>DCI_INT_FLAG_EL</code>	行结束中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

## 3.9. 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[3.9.1](#)描述了DMA的寄存器列表，章节[3.9.2](#)对DMA库函数进行说明。

### 3.9.1. 外设寄存器说明

DMA寄存器列表如下表所示：

**表 3-185. 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..7)	通道x控制寄存器
DMA_CHxCNT (x=0..7)	通道x计数寄存器
DMA_CHxPADDR (x=0..7)	通道x外设基址寄存器
DMA_CHxM0ADDR (x=0..7)	通道x存储器0基址寄存器
DMA_CHxM1ADDR (x=0..7)	通道x存储器1基址寄存器
DMA_CHxFCTL (x=0..7)	通道x FIFO控制寄存器

### 3.9.2. 外设库函数说明

DMA库函数列表如下表所示：

**表 3-186. DMA 库函数**

库函数名称	库函数描述
dma_deinit	复位DMA一个通道的所有寄存器
dma_single_data_mode_init	DMA单数据传输模式初始化
dma_multi_data_mode_init	DMA多数据传输模式初始化
dma_periph_address_config	DMA外设基址配置
dma_memory_address_config	DMA存储器基址配置
dma_transfer_number_config	DMA需要传输数据数配置

<code>dma_transfer_number_get</code>	DMA需要传输数据数获取
<code>dma_priority_config</code>	DMA通道传输优先级配置
<code>dma_memory_burst_beats_config</code>	存储器增量突发传输节拍数配置
<code>dma_periph_burst_beats_config</code>	外设增量突发传输节拍数配置
<code>dma_memory_width_config</code>	存储器传输数据宽度配置
<code>dma_periph_width_config</code>	外设传输数据宽度配置
<code>dma_memory_address_generation_config</code>	存储器下地址算法配置
<code>dma_peripheral_address_generation_config</code>	外设地址算法配置
<code>dma_circulation_enable</code>	DMA循环传输模式使能
<code>dma_circulation_disable</code>	DMA循环传输模式失能
<code>dma_channel_enable</code>	DMA通道使能
<code>dma_channel_disable</code>	DMA通道失能
<code>dma_transfer_direction_config</code>	DMA通道数据传输方向配置
<code>dma_switch_buffer_mode_config</code>	DMA存储器传输缓冲区配置
<code>dma_using_memory_get</code>	DMA存储器传输缓冲区获取
<code>dma_channel_subperipheral_select</code>	DMA通道外设使能选择
<code>dma_flow_controller_config</code>	DMA传输控制器配置
<code>dma_switch_buffer_mode_enable</code>	DMA存储切换模式使能
<code>dma_fifo_status_get</code>	DMA FIFO状态获取
<code>dma_flag_get</code>	DMA通道传输标志位获取
<code>dma_flag_clear</code>	DMA通道传输标志位清除
<code>dma_interrupt_flag_get</code>	DMA通道传输中断标志位获取
<code>dma_interrupt_flag_clear</code>	DMA 通道传输中断标志位清除
<code>dma_interrupt_enable</code>	DMA中断使能
<code>dma_interrupt_disable</code>	DMA中断失能

### 结构体 `dma_multi_data_parameter_struct`

**表 3-187. 结构体 `dma_multi_data_parameter_struct`**

成员名称	功能描述
<code>periph_addr</code>	外设基地址
<code>periph_width</code>	外设数据传输宽度
<code>periph_inc</code>	外设地址生成算法模式
<code>memory0_addr</code>	存储器0基地址
<code>memory_width</code>	存储器数据传输宽度
<code>memory_inc</code>	存储器地址生成算法模式
<code>memory_burst_width</code>	存储器增量突发类型
<code>periph_burst_width</code>	外设增量突发类型
<code>critical_value</code>	DMA FIFO寄存器数
<code>circular_mode</code>	DMA循环模式
<code>direction</code>	DMA通道数据传输方向
<code>number</code>	DMA通道数据传输数
<code>priority</code>	DMA通道优先级

### 结构体 `dma_single_data_parameter_struct`

**表 3-188. 结构体 `dma_single_data_parameter_struct`**

成员名称	功能描述
<code>periph_addr</code>	外设基地址
<code>periph_inc</code>	外设地址生成算法模式
<code>memory0_addr</code>	存储器0基地址
<code>memory_inc</code>	存储器地址生成算法模式
<code>periph_memory_width</code>	外设数据传输宽度
<code>circular_mode</code>	DMA循环模式
<code>direction</code>	DMA通道数据传输方向
<code>number</code>	DMA通道数据传输数

priority	DMA通道优先级
----------	----------

### 函数 **dma\_deinit**

函数dma\_deinit描述见下表：

**表 3-189. 函数 dma\_deinit**

函数名称	dma_deinit
函数原型	void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	复位DMA一个通道的所有寄存器
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..7)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

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

### 函数 **dma\_single\_data\_mode\_init**

函数dma\_single\_data\_mode\_init描述见下表：

**表 3-190. 函数 dma\_single\_data\_mode\_init**

函数名称	dma_single_data_mode_init
函数原形	void dma_single_data_mode_init(uint32_t dma_periph,dma_channel_enum channelx,dma_single_data_parameter_struct* init_struct);

功能描述	DMA单数据传输模式初始化
先决条件	-
被调用函数	-
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	指定被初始化的DMA通道
DMA_CHx(x=0..7)	DMA通道选择
输出参数{ in }	
init_struct	初始化结构体, 结构体成员参考 <a href="#">表 3-188. 结构体 dma_single_data_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize DMA single data mode */
dma_single_data_parameter_struct dma_single_data_struct;
dma_single_data_mode_init(DMAx0, DMA_CH0, &dma_single_data_struct);
```

#### 函数 **dma\_multi\_data\_mode\_init**

函数dma\_multi\_data\_mode\_init描述见下表：

表 3-191. 函数 **dma\_multi\_data\_mode\_init**

函数名称	dma_multi_data_mode_init
函数原型	void dma_multi_data_mode_init(uint32_t dma_periph,dma_channel_enum channelx,dma_multi_data_parameter_struct* init_struct);
功能描述	DMA多数据传输模式初始化
先决条件	-
被调用函数	-

输入参数{in}	
dma_periph	DMA外设
DMAx( $x=0, 1$ )	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx( $x=0..7$ )	DMA通道选择
输入参数{in}	
init_struct	初始化结构体, 结构体成员参考 <a href="#">表 3-187. 结构体 dma_multi_data_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize DMA multi data mode */
dma_multi_data_parameter_struct dma_multi_data_struct;
dma_init(DMA0, DMA_CH0, &dma_multi_data_struct);
```

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

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

**表 3-192. 函数 **dma\_periph\_address\_config****

函数名称	dma_periph_address_config
函数原型	void dma_periph_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
功能描述	DMA外设基地址配置
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx( $x=0, 1$ )	DMA外设选择

输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx(x=0..7)</i>	DMA通道选择
输入参数{in}	
<b>address</b>	外设基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

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

函数**dma\_memory\_address\_config**描述见下表：

表 3-193. 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,uint8_t memory_flag,uint32_t address);
功能描述	DMA存储器基地址配置
先决条件	-
被调用函数	-
输入参数{in}	
<b>dma_periph</b>	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx(x=0..7)</i>	DMA通道选择

输入参数{in}	
<b>memory_flag</b>	DMA存储器
<i>DMA_MEMORY_x(x=0, 1)</i>	DMA存储器选择
输入参数{in}	
<b>address</b>	存储器地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
uint8_t g_destbuf[TRANSFER_NUM];
dma_memory_address_config(DMA0, DMA_CH0, DMA_MEMORY_0, (uint32_t)g_destbuf);
```

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

函数dma\_transfer\_number\_config描述见下表：

表 3-194. 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	DMA需要传输数据数配置
先决条件	-
被调用函数	-
输入参数{in}	
<b>dma_periph</b>	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx(x=0..7)</i>	DMA通道选择

输入参数{in}	
<b>number</b>	DMA需要传输数据数
0-0xffff	数据数
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 3-195. 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	DMA需要传输数据数获取
先决条件	-
被调用函数	-
输入参数{in}	
<b>dma_periph</b>	DMA外设
<b>DMAx(x=0..1)</b>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<b>DMA_CHx(x=0..7)</b>	DMA通道选择
输出参数{out}	
-	-

返回值	
<b>uint32_t(0-0xffff)</b>	DMA需要传输数据数(0-0xffff)

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 3-196. 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	configure priority level of DMA channel
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0, 1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..7)	DMA通道选择
输入参数{in}	
priority	通道的优先等级
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\_burst\_beats\_config**

函数dma\_memory\_burst\_beats\_config描述见下表：

表 3-197. 函数 **dma\_memory\_burst\_beats\_config**

函数名称	dma_memory_burst_beats_config
函数原型	void dma_memory_burst_beats_config (uint32_t dma_periph,dma_channel_enum channelx,uint32_t mbeat);
功能描述	存储器增量突发传输节拍数配置
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0, 1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..7)	DMA通道选择
输入参数{in}	
mbeat	存储器增量突发类型
DMA_MEMORY_BURO T_SINGLE	存储器单一传输
DMA_MEMORY_BURO T_4_BEAT	存储器4拍增量突发传输
DMA_MEMORY_BURO T_8_BEAT	存储器8拍增量突发传输
DMA_MEMORY_BURO T_16_BEAT	存储器16拍增量突发传输

输出参数{out}	
-	-
返回值	
-	-

例如：

```
dma_memory_burst_beats_config (DMA0, DMA_CH0, DMA_MEMORY_BURST_8_BEAT);
```

#### 函数 **dma\_periph\_burst\_beats\_config**

函数dma\_periph\_burst\_beats\_config描述见下表：

表 3-198. 函数 **dma\_periph\_burst\_beats\_config**

函数名称	dma_periph_burst_beats_config
函数原型	void dma_periph_burst_beats_config (uint32_t dma_periph,dma_channel_enum channelx,uint32_t pbeat);
功能描述	外设增量突发传输节拍数配置
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..7)	DMA通道选择
输入参数{in}	
pbeat	外设增量突发类型
DMA_PERIPH_BURST_SINGLE	外设单一传输
DMA_PERIPH_BURST_4_BEAT	外设4拍增量突发传输
DMA_PERIPH_BURST_8_BEAT	外设8拍增量突发传输

<i>DMA_PERIPH_BURST_16_BEAT</i>	外设16拍增量突发传输
输出参数{out}	
-	-
返回值	
-	-

例如：

```
dma_periph_burst_beats_config(DMA0, DMA_CH0, DMA_PERIPH_BURST_8_BEAT);
```

### 函数 **dma\_memory\_width\_config**

函数**dma\_memory\_width\_config**描述见下表：

**表 3-199. 函数**dma\_memory\_width\_config****

函数名称	<b>dma_memory_width_config</b>
函数原型	void dma_memory_width_config(uint32_t dma_periph,dma_channel_enum channelx,uint32_t msize);
功能描述	存储器传输数据宽度配置
先决条件	-
被调用函数	-
输入参数{in}	
<b>dma_periph</b>	DMA外设
<i>DMAx(x=0, 1)</i>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx(x=0..7)</i>	DMA通道选择
输入参数{in}	
<b>msize</b>	存储器传输数据宽度
<i>DMA_MEMORY_WIDTH_8BIT</i>	存储器传输数据8-bit宽度
<i>DMA_MEMORY_WIDTH_16BIT</i>	存储器传输数据16-bit宽度

<i>DMA_MEMORY_WIDTH_32BIT</i>	存储器传输数据32-bit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 3-200. 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	外设传输数据宽度配置
先决条件	-
被调用函数	-
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外设
<i>DMA_CHx(x=0..7)</i>	DMA外设选择
输入参数{in}	
<b>psize</b>	外设传输数据宽度
<i>DMA_PERIPHERAL_WIDTH_8BIT</i>	外设传输数据8-bit宽度
<i>DMA_PERIPHERAL_WIDTH_16BIT</i>	外设传输数据16-bit宽度

<i>DMA_PERIPHERAL_WIDTH_32BIT</i>	外设传输数据32-bit宽度
输出参数{out}	
-	-
返回值	
-	-

例如：

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

### 函数 **dma\_memory\_address\_generation\_config**

函数dma\_memory\_address\_generation\_config描述见下表：

**表 3-201. 函数dma\_memory\_address\_generation\_config**

函数名称	dma_memory_address_generation_config
函数原型	void dma_memory_address_generation_config(uint32_t dma_periph,dma_channel_enum channelx,uint8_t generation_algorithm);
功能描述	存储器下地址算法配置
先决条件	-
被调用函数	-
输入参数{in}	
<b>dma_periph</b>	DMA外设
<b>DMAx(x=0, 1)</b>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<b>DMA_CHx(x=0..7)</b>	DMA通道选择
输入参数{in}	
<b>generation_algorithm</b>	存储器地址生成算法
<b>DMA_MEMORY_INCR_EASE_ENABLE</b>	存储器增量地址模式
<b>DMA_MEMORY_INCR_EASE_DISABLE</b>	存储器固定地址模式

输出参数{out}	
-	-
返回值	
-	-

例如：

```
dma_memory_address_generation_config (DMA0, DMA_CH0,
DMA_MEMORY_INCREASE_ENABLE);
```

### 函数 **dma\_peripheral\_address\_generation\_config**

函数**dma\_peripheral\_address\_generation\_config**描述见下表：

**表 3-202. 函数 **dma\_peripheral\_address\_generation\_config****

函数名称	dma_peripheral_address_generation_config
函数原型	void dma_peripheral_address_generation_config(uint32_t dma_periph,dma_channel_enum channelx,uint8_t generation_algorithm);
功能描述	外设寻址算法配置
先决条件	-
被调用函数	-
输入参数{in}	
<b>dma_periph</b>	DMA外设
<b>DMAx(x=0,1)</b>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<b>DMA_CHx(x=0..7)</b>	DMA通道选择
输入参数{in}	
<b>generation_algorithm</b>	外设地址生成算法
<b>DMA_PERIPH_INCRE ASE_ENABLE</b>	外设增量地址模式
<b>DMA_PERIPH_INCRE ASE_DISABLE</b>	外设固定地址模式
<b>DMA_PERIPH_INCRE ASE_FIX</b>	外设地址增量固定为4（非PWIDTH位决定）

输出参数{out}	
-	-
返回值	
-	-

例如：

```
dma_peripheral_address_generation_config (DMA0, DMA_CH0, DMA_PERIPH_INCREASE_ENABLE);
```

### 函数 **dma\_circulation\_enable**

函数**dma\_circulation\_enable**描述见下表：

**表 3-203. 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循环传输模式使能
先决条件	-
被调用函数	-
输入参数{in}	
<b>dma_periph</b>	DMA外设
<b>DMAx(x=0,1)</b>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<b>DMA_CHx(x=0..7)</b>	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
dma_circulation_enable (DMA0, DMA_CH0);
```

### 函数 **dma\_circulation\_disable**

函数**dma\_circulation\_disable**描述见下表:

**表 3-204. 函数 **dma\_circulation\_disable****

函数名称	dma_circulation_disable
函数原型	vvoid dma_circulation_disable(uint32_t dma_periph,dma_channel_enum channelx);
功能描述	DMA循环传输模式失能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>dma_periph</b>	DMA外设
<b>DMAx(x=0, 1)</b>	DMA外设选择
<b>输入参数{in}</b>	
<b>channelx</b>	DMA通道
<b>DMA_CHx(x=0..7)</b>	DMA通道选择
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

例如:

```
dma_circulation_disable (DMA0, DMA_CH0);
```

### 函数 **dma\_channel\_enable**

函数**dma\_channel\_enable**描述见下表:

**表 3-205. 函数 **dma\_channel\_enable****

函数名称	dma_channel_enable
函数原型	void dma_channel_enable(uint32_t dma_periph,dma_channel_enum channelx);
功能描述	DMA通道使能
先决条件	-

<b>被调用函数</b>	-
<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(x=0..7)</i>	DMA通道选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
dma_channel_enable (DMA0, DMA_CH0);
```

#### 函数**dma\_channel\_disable**

函数**dma\_channel\_disable**描述见下表：

**表 3-206. 函数 `dma_channel_disable`**

<b>函数名称</b>	<code>dma_channel_disable</code>
<b>函数原型</b>	<code>void dma_channel_disable(uint32_t dma_periph,dma_channel_enum channelx);</code>
<b>功能描述</b>	DMA通道失能
<b>先决条件</b>	-
<b>被调用函数</b>	-
<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(x=0..7)</i>	DMA通道选择
<b>输出参数{out}</b>	

-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如：

```
dma_channel_disable (DMA0, DMA_CH0);
```

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

函数**dma\_transfer\_direction\_config**描述见下表：

**表 3-207. 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,uint8_t direction);
功能描述	DMA通道数据传输方向配置
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>dma_periph</b>	DMA外设
<b>DMAx(x=0, 1)</b>	DMA外设选择
<b>输入参数{in}</b>	
<b>channelx</b>	DMA通道
<b>DMA_CHx(x=0..7)</b>	DMA通道选择
<b>输出参数{out}</b>	
<b>direction</b>	指定数据传输的方向
<b>DMA_PERIPH_TO_MEMORY</b>	从外设读取并写入存储器
<b>DMA_MEMORY_TO_PERIPH</b>	从存储器读取并写入外设
<b>DMA_MEMORY_TO_MEMORY</b>	从存储器读取并写入存储器
<b>输出参数{out}</b>	

-	
返回值	
-	-

例如：

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

### 函数 **dma\_switch\_buffer\_mode\_config**

函数**dma\_switch\_buffer\_mode\_config**描述见下表：

表 3-208. 函数 **dma\_switch\_buffer\_mode\_config**

函数名称	dma_switch_buffer_mode_config
函数原型	void dma_switch_buffer_mode_config(uint32_t dma_periph,dma_channel_enum channelx,uint32_t memory1_addr,uint32_t memory_select);
功能描述	DMA存储器传输缓冲区配置
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0, 1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..7)	DMA通道选择
输入参数{in}	
memory1_addr	存储器1基地址
输入参数{in}	
memory_select	DMA存储器0或DMA存储器1
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
dma_switch_buffer_mode_config(DMA0, DMA_CH0,0xaabbccdd,DMA_MEMORY_0);
```

### 函数 **dma\_using\_memory\_get**

函数dma\_using\_memory\_get描述见下表：

**表 3-209. 函数 dma\_using\_memory\_get**

函数名称	dma_using_memory_get
函数原型	uint32_t dma_using_memory_get(uint32_t dma_periph,dma_channel_enum channelx);
功能描述	DMA存储器传输缓冲区获取
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0, 1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..7)	DMA通道选择
输出参数{out}	
-	-
返回值	
uint32_t	使用的存储器

例如：

```
Uint32_t using_memory = dma_using_memory_get(DMA0, DMA_CH0);
```

### 函数 **dma\_channel\_subperipheral\_select**

函数dma\_channel\_subperipheral\_select描述见下表：

**表 3-210. 函数 dma\_channel\_subperipheral\_select**

函数名称	dma_channel_subperipheral_select
------	----------------------------------

函数原型	void dma_channel_subperipheral_select(uint32_t dma_periph,dma_channel_enum channelx,dma_subperipheral_enum sub_periph);
功能描述	DMA通道外设使能选择
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>dma_periph</b>	DMA外设
<b>DMAx(x=0,1)</b>	DMA外设选择
<b>输入参数{in}</b>	
<b>channelx</b>	DMA通道
<b>DMA_CHx(x=0..7)</b>	DMA通道选择
<b>输入参数{in}</b>	
<b>sub_periph</b>	指定DMA通道的外设
<b>DMA_SUBPERIx(x=0..7)</b>	DMA通道外设选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
dma_channel_subperipheral_select (DMA0, DMA_CH0, DMA_SUBPERI0);
```

### 函数 **dma\_flow\_controller\_config**

函数**dma\_flow\_controller\_config**描述见下表：

**表 3-211. 函数 **dma\_flow\_controller\_config****

函数名称	<b>dma_flow_controller_config</b>
函数原型	void dma_flow_controller_config(uint32_t dma_periph,dma_channel_enum channelx,uint32_t controller);
功能描述	DMA传输控制器配置

先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>dma_periph</b>	DMA外设
<b>DMAx(x=0,1)</b>	DMA外设选择
<b>输入参数{in}</b>	
<b>channelx</b>	DMA通道
<b>DMA_CHx(x=0..7)</b>	DMA通道选择
<b>输入参数{in}</b>	
<b>controller</b>	指定DMA传输控制器
<b>DMA_FLOW_CONTROLLER_DMA</b>	DMA作为传输控制器
<b>DMA_FLOW_CONTROLLER_PERI</b>	外设作为传输控制器
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
dma_flow_controller_config (DMA0, DMA_CH0, DMA_FLOW_CONTROLLER_DMA);
```

#### 函数 **dma\_switch\_buffer\_mode\_enable**

函数**dma\_switch\_buffer\_mode\_enable**描述见下表：

**表 3-212. 函数 **dma\_switch\_buffer\_mode\_enable****

函数名称	<b>dma_switch_buffer_mode_enable</b>
函数原型	void dma_switch_buffer_mode_enable(uint32_t dma_periph,dma_channel_enum channelx,ControlStatus newvalue);
功能描述	DMA存储切换模式使能
先决条件	-
被调用函数	-

输入参数{in}	
<b>dma_periph</b>	DMA外设
<i>DMAx(x=0, 1)</i>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx(x=0..7)</i>	DMA通道选择
输入参数{in}	
<b>newvalue</b>	ENABLE或DISABLE
输出参数{out}	
-	-
返回值	
-	-

例如：

```
dma_switch_buffer_mode_enable(DMA0, DMA_CH0, ENABLE);
```

#### 函数 **dma\_fifo\_status\_get**

函数dma\_fifo\_status\_get描述见下表：

表 3-213. 函数 **dma\_fifo\_status\_get**

函数名称	dma_fifo_status_get
函数原型	uint32_t dma_fifo_status_get(uint32_t dma_periph,dma_channel_enum channelx);
功能描述	DMA FIFO状态获取
先决条件	-
被调用函数	-
输入参数{in}	
<b>dma_periph</b>	DMA外设
<i>DMAx(x=0, 1)</i>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道

<i>DMA_CHx(x=0..7)</i>	DMA通道选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<i>uint32_t</i>	使用的存储器

例如：

```
uint32_t using_memory dma_fifo_status_get (DMA0, DMA_CH0);
```

### 函数 **dma\_flag\_get**

函数dma\_flag\_get描述见下表：

**表 3-214. 函数 **dma\_flag\_get****

函数名称	<b>dma_flag_get</b>
函数原型	FlagStatus dma_flag_get(uint32_t dma_periph,dma_channel_enum channelx,uint32_t flag);
功能描述	DMA通道传输标志位获取
先决条件	-
被调用函数	-
<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(x=0..7)</i>	DMA通道选择
<b>输入参数{in}</b>	
<b>flag</b>	指定获取的标志位
<b>DMA_FLAG_FEE</b>	FIFO错误与异常标志
<b>DMA_FLAG_SDE</b>	单数据传输模式异常标志
<b>DMA_FLAG_TAE</b>	传输错误标志
<b>DMA_FLAG_HTF</b>	半传输完成标志

<i>DMA_FLAG_FTF</i>	传输完成标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET或RESET

例如：

```
FlagStatus flag_status = dma_flag_get (DMA0, DMA_CH0, DMA_FLAG_FEE);
```

### 函数 **dma\_flag\_clear**

函数dma\_flag\_clear描述见下表：

**表 3-215. 函数 **dma\_flag\_clear****

函数名称	dma_flag_clear
函数原型	void dma_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	DMA通道传输标志位清除
先决条件	-
被调用函数	-
<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(x=0..7)</i>	DMA通道选择
<b>输入参数{in}</b>	
<b>flag</b>	指定获取的标志位
<i>DMA_FLAG_FEE</i>	FIFO错误与异常标志
<i>DMA_FLAG_SDE</i>	单数据传输模式异常标志
<i>DMA_FLAG_TAE</i>	传输错误标志
<i>DMA_FLAGHTF</i>	半传输完成标志

<i>DMA_FLAG_FTF</i>	传输完成标志
<b>输出参数{out}</b>	
-	-
返回值	
-	-

例如：

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

#### 函数 **dma\_interrupt\_flag\_get**

函数**dma\_interrupt\_flag\_get**描述见下表：

**表 3-216. 函数 *dma\_interrupt\_flag\_get***

函数名称	<i>dma_interrupt_flag_get</i>
函数原型	FlagStatus <i>dma_interrupt_flag_get</i> (uint32_t <i>dma_periph</i> , <i>dma_channel_enum</i> <i>channelx</i> , uint32_t <i>interrupt</i> );
功能描述	DMA通道传输中断标志位获取
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<i>dma_periph</i>	DMA外设
<i>DMAx(x=0, 1)</i>	DMA外设选择
<b>输入参数{in}</b>	
<i>channelx</i>	DMA通道
<i>DMA_CHx(x=0..7)</i>	DMA通道选择
<b>输入参数{in}</b>	
<i>interrupt</i>	指定获取的标志位
<i>DMA_INT_FLAG_FEE</i>	FIFO错误与异常标志
<i>DMA_INT_FLAG_SDE</i>	单数据传输模式异常标志
<i>DMA_INT_FLAG_TAE</i>	传输错误标志
<i>DMA_INT_FLAG_HTF</i>	半传输完成标志

<i>DMA_INT_FLAG_FTF</i>	传输完成标志
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 3-217. 函数 **dma\_interrupt\_flag\_clear****

函数名称	<b>dma_interrupt_flag_clear</b>
函数原型	void dma_interrupt_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t interrupt);
功能描述	DMA 通道传输中断标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
<b>dma_periph</b>	DMA外设
<i>DMAx(x=0, 1)</i>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx(x=0..7)</i>	DMA通道选择
输入参数{in}	
<b>interrupt</b>	指定获取的标志位
<i>DMA_INT_FLAG_FEE</i>	FIFO错误与异常标志
<i>DMA_INT_FLAG_SDE</i>	单数据传输模式异常标志
<i>DMA_INT_FLAG_TAE</i>	传输错误标志

<i>DMA_INT_FLAG_HTF</i>	半传输完成标志
<i>DMA_INT_FLAG_FTF</i>	传输完成标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

### 函数 **dma\_interrupt\_enable**

函数**dma\_interrupt\_enable**描述见下表：

**表 3-218. 函数 **dma\_interrupt\_enable****

函数名称	<b>dma_interrupt_enable</b>
函数原型	void dma_interrupt_enable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
功能描述	DMA中断使能
先决条件	-
被调用函数	-
<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外设
<i>DMA_CHx(x=0..7)</i>	DMA外设选择
<b>输入参数{in}</b>	
<b>source</b>	DMA中断源
<i>DMA_CHXCTL_SDEIE</i>	单数据传输模式异常中断使能位
<i>DMA_CHXCTL_TAEIE</i>	传输错误中断使能位

<i>DMA_CHXCTL_HTFIE</i>	半传输完成中断使能位
<i>DMA_CHXCTL_FTFIE</i>	传输完成中断使能位
<i>DMA_CHXFCTL_FEEIE</i>	FIFO异常中断使能位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* DMA0 channel0 interrupt configuration */

dma_interrupt_enable(DMA0, DMA_CH0, DMA_CHXCTL_SDEIE);
```

#### 函数 **dma\_interrupt\_disable**

函数dma\_interrupt\_disable描述见下表：

**表 3-219. 函数dma\_interrupt\_disable**

函数名称	dma_interrupt_disable
函数原型	void dma_interrupt_disable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
功能描述	DMA中断失能
先决条件	-
被调用函数	-
输入参数{in}	
<b>dma_periph</b>	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx(x=0..7)</i>	DMA通道选择
输入参数{in}	
<b>source</b>	DMA中断源

<i>DMA_CHXCTL_SDEIE</i>	单数据传输模式异常中断使能位
<i>DMA_CHXCTL_TAEIE</i>	传输错误中断使能位
<i>DMA_CHXCTL_HTFIE</i>	半传输完成中断使能位
<i>DMA_CHXCTL_FTFIE</i>	传输完成中断使能位
<i>DMA_CHXFCTL_FEEIE</i>	FIFO异常中断使能位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

## 3.10. 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[3.10.1](#)描述了ENET的寄存器列表，章节[3.10.2](#)对ENET库函数进行说明。

### 3.10.1. 外设寄存器描述

ENET寄存器列表如下表所示：

**表 3-220. ENET 寄存器**

寄存器名称	寄存器描述
<i>ENET_MAC_CFG</i>	MAC配置寄存器
<i>ENET_MAC_FRMF</i>	MAC帧过滤器寄存器
<i>ENET_MAC_HLH</i>	MAC hash列表高寄存器
<i>ENET_MAC_HLL</i>	MAC hash列表低寄存器
<i>ENET_MAC_PHY_CTL</i>	MAC PHY控制寄存器

寄存器名称	寄存器描述
ENET_MAC_PHY_DATA	MAC MII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_INTMSK	MAC中断屏蔽寄存器
ENET_MAC_ADDR0H	MAC地址0高寄存器
ENET_MAC_ADDR0L	MAC地址0低寄存器
ENET_MAC_ADDR1H	MAC地址1高寄存器
ENET_MAC_ADDR1L	MAC地址1低寄存器
ENET_MAC_ADDT2H	MAC地址2高寄存器
ENET_MAC_ADDR2L	MAC地址2低寄存器
ENET_MAC_ADDR3H	MAC地址3高寄存器
ENET_MAC_ADDR3L	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	MSC接收中断屏蔽寄存器

寄存器名称	寄存器描述
MSK	
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
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_TSFS	PTP时间戳标志寄存器
ENET_PTP_PPSCT L	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

寄存器名称	寄存器描述
ENET_DMA_RDTA DDR	DMA接收描述符列表地址寄存器
ENET_DMA_TDTA 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

### 3.10.2. 外设库函数说明

ENET库函数列表如下表所示：

**表 3-221. 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
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

库函数名称	库函数描述
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接收功能禁能（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

库函数名称	库函数描述
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
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

库函数名称	库函数描述
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

库函数名称	库函数描述
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_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
其它	
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 3-222. 结构体 **enet\_initpara\_struct**

成员名称	功能描述

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

### 结构体 **enet\_descriptors\_struct**

表 3-223. 结构体 **enet\_descriptors\_struct**

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

### 结构体 `enet_ptp_systime_struct`

**表 3-224. 结构体 `enet_ptp_systime_struct`**

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

### 函数 `enet_deinit`

函数`enet_deinit`描述见下表：

**表 3-225. 函数 `enet_deinit`**

函数名称	enet_deinit
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()</code> / <code>rcu_periph_reset_disable()</code> / <code>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 3-226. 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>

<b>功能描述</b>	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
<b>先决条件</b>	enet_initpara_reset(void)
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>option</b>	ENET模块功能选项，根据选择需要使用不同参数进行配置，下列参数仅可选择一个
<i>FORWARD_OPTION_N</i>	选择配置帧通过功能相关参数
<i>DMABUS_OPTION</i>	选择配置DMA总线模式相关参数
<i>DMA_MAXBURST_OPTION</i>	选择配置DMA最大突发传输相关参数
<i>DMA_ARBITRATION_OPTION</i>	选择配置DMA仲裁相关参数
<i>STORE_OPTION</i>	选择配置存储转发模式相关参数
<i>DMA_OPTION</i>	选择配置DMA相关参数
<i>VLAN_OPTION</i>	选择配置VLAN相关参数
<i>FLOWCTL_OPTION</i>	选择配置流控相关参数
<i>HASHH_OPTION</i>	选择配置HASH_H相关参数
<i>HASHL_OPTION</i>	选择配置HASH_L相关参数
<i>FILTER_OPTION</i>	选择配置帧过滤器相关参数
<i>HALFDUPLEX_OPTION</i>	选择配置半双工模式相关参数
<i>TIMER_OPTION</i>	选择配置计数器相关参数
<i>INTERFRAMEGAP_OPTION</i>	选择配置帧间隔相关参数
<b>输入参数{in}</b>	
<b>para</b> (该参数值需根据 <b>option</b> 参数对应值进 行选取)	下列参数值均可以被配置 例如： <b>para = (value1   value2   value3...)</b>
当 <b>option</b> 参数值为 <i>FORWARD_OPTION</i> 时	

value1	<i>ENET_AUTO_PADCRC_DROP_ENABLE / ENET_AUTO_PADCRC_DROP_DISABLE</i>
value2	<i>ENET_TYPEFRAME_CRC_DROP_ENABLE / ENET_TYPEFRAME_CRC_DROP_DISABLE</i>
value3	<i>ENET_FORWARD_ERRFRAMES_ENABLE / ENET_FORWARD_ERRFRAMES_DISABLE</i>
value4	<i>ENET_FORWARD_UNDERSZ_GOODFRAMES_ENABLE / ENET_FORWARD_UNDERSZ_GOODFRAMES_DISABLE</i>
当 <b>option</b> 参数值为 <i>DMA_BUS_OPTION</i> 时	
value1	<i>ENET_ADDRESS_ALIGN_ENABLE / ENET_ADDRESS_ALIGN_DISABLE</i>
value2	<i>ENET_FIXED_BURST_ENABLE / ENET_FIXED_BURST_DISABLE</i>
value3	<i>ENET_MIXED_BURST_ENABLE / ENET_MIXED_BURST_DISABLE</i>
当 <b>option</b> 参数值为 <i>DMA_MAXBURST_OPTION</i> 时	
value1	<i>ENET_RXDP_1BEAT / ENET_RXDP_2BEAT / ENET_RXDP_4BEAT / ENET_RXDP_8BEAT / ENET_RXDP_16BEAT / ENET_RXDP_32BEAT / ENET_RXDP_4xPGBL_4BEAT / ENET_RXDP_4xPGBL_8BEAT / ENET_RXDP_4xPGBL_16BEAT / ENET_RXDP_4xPGBL_32BEAT / ENET_RXDP_4xPGBL_64BEAT / ENET_RXDP_4xPGBL_128BEAT</i>
value2	<i>ENET_PGBL_1BEAT / ENET_PGBL_2BEAT / ENET_PGBL_4BEAT / ENET_PGBL_8BEAT / ENET_PGBL_16BEAT / ENET_PGBL_32BEAT / ENET_PGBL_4xPGBL_4BEAT / ENET_PGBL_4xPGBL_8BEAT / ENET_PGBL_4xPGBL_16BEAT / ENET_PGBL_4xPGBL_32BEAT / ENET_PGBL_4xPGBL_64BEAT / ENET_PGBL_4xPGBL_128BEAT</i>
value3	<i>ENET_RXTX_DIFFERENT_PGBL / ENET_RXTX_SAME_PGBL</i>
当 <b>option</b> 参数值为 <i>DMA_ARBITRATION_OPTION</i> 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 <i>STORE_OPTION</i> 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 <b>DMA_OPTION</b> 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 <b>VLAN_OPTION</b> 时	
value1	<i>ENET_VLANTAGCOMPARISON_12BIT / ENET_VLANTAGCOMPARISON_16BIT</i>
value2	<i>MAC_VLT_VLT(<i>regval</i>)</i>
当 <b>option</b> 参数值为 <b>FLOWCTL_OPTION</b> 时	
value1	<i>MAC_FCTL_PTM(<i>regval</i>)</i>
value2	<i>ENET_ZERO_QUANTA_PAUSE_ENABLE / ENET_ZERO_QUANTA_PAUSE_DISABLE</i>
value3	<i>ENET_PAUSETIME_MINUS4 / ENET_PAUSETIME_MINUS28 / ENET_PAUSETIME_MINUS144 / ENET_PAUSETIME_MINUS256</i>
value4	<i>ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDTECT / ENET_UNIQUE_PAUSEDTECT</i>
value5	<i>ENET_RX_FLOWCONTROL_ENABLE / ENET_RX_FLOWCONTROL_DISABLE</i>
value6	<i>ENET_TX_FLOWCONTROL_ENABLE / ENET_TX_FLOWCONTROL_DISABLE</i>
当 <b>option</b> 参数值为 <b>HASHH_OPTION</b> 时	
value1	<i>0x0~0xFFFF FFFFU</i>
当 <b>option</b> 参数值为 <b>HASHL_OPTION</b> 时	
value1	<i>0x0~0xFFFF FFFFU</i>
当 <b>option</b> 参数值为 <b>FILTER_OPTION</b> 时	
value1	<i>ENET_SRC_FILTER_NORMAL_ENABLE / ENET_SRC_FILTER_INVERSE_ENABLE / ENET_SRC_FILTER_DISABLE</i>

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

例如：

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

```

enet_initpara_reset();

enet_initpara_config(DMA_OPTION, ENET_FLUSH_RXFRAME_ENABLE | ENET_SECO
NDFRAME_OPT_ENABLE | ENET_NORMAL_DESCRIPTOR);

```

### 函数 **enet\_init**

函数**enet\_init**描述见下表：

**表 3-227. 函数 enet\_init**

函数名称	enet_init
函数原型	ErrStatus enet_init(enet_mediemode_enum mediemode, enetc_checksumconf_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}	
mediemode	ENET通讯方式配置，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_HALF DUPLEX	100Mbit/s, 半双工
ENET_10M_FULLDUPLEX	10Mbit/s, 全双工
ENET_10M_HALFDUPLEX	10Mbit/s, 半双工
ENET_LOOPBACK MODE	MII模式下的回环模式
输入参数{in}	
checksum	IP帧数据校验和功能，仅可选择唯一参数
ENET_NO_AUTOCHECKSUM	关闭IP帧校验和功能
ENET_AUTOCHECKSUM_DROP_FAILIF	使能IP帧校验和功能

<i>RAMES</i>	
<i>ENET_AUTOCHECKSUM_DR</i> <i>SUM_ACCEPT_FAIL_FRAMES</i>	使能IP帧校验和功能，不丢弃仅有载荷错误的帧
<b>输入参数{in}</b>	
<b>receipt</b>	帧过滤功能，仅可选择唯一参数
<i>ENET_PROMISCIOUS_MODE</i>	使能混杂模式
<i>ENET_RECEIVEALL</i>	接收所有帧
<i>ENET_BROADCAST_FRAMES_PASS</i>	接收广播帧
<i>ENET_BROADCAST_FRAMES_DROP</i>	禁止接收广播帧
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

例如：

```
/* initialize ENET peripheral */

ErrStatus enet_init_status;

enet_init_status = enet_init(ENET_AUTO_NEGOTIATION, ENET_AUTOCHECKSUM_DR
OP_FAILFRAMES, ENET_BROADCAST_FRAMES_PASS);
```

### 函数 **enet\_software\_reset**

函数**enet\_software\_reset**描述见下表：

**表 3-228. 函数 **enet\_software\_reset****

函数名称	<b>enet_software_reset</b>
函数原型	ErrStatus enet_software_reset(void);
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 or 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enetc\_rxframe\_size\_get描述见下表：

表 3-229. 函数 **enet\_rxframe\_size\_get**

函数名称	enet_rxframe_size_get
函数原型	uint32_t enet_rxframe_size_get(void);
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	
<b>uint32_t</b>	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 3-230. 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，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 3-231. 函数 enet\_descriptors\_ring\_init

函数名称	enet_descriptors_ring_init
函数原型	void enet_descriptors_ring_init(enet_dmadirection_enum direction);
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，下列参数仅可选择一个
<i>ENET_DMA_TX</i>	DMA Tx描述符
<i>ENET_DMA_RX</i>	DMA Rx描述符
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-232. 函数 **enet\_frame\_receive****

<b>函数名称</b>	enet_frame_receive
<b>函数原型</b>	ErrStatus enet_frame_receive(uint8_t *buffer, uint32_t bufsize);
<b>功能描述</b>	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
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>bufsize</b>	缓冲区长度，范围(0~1524)
<b>输出参数{out}</b>	
<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
<b>返回值</b>	
<b>ErrStatus</b>	ERROR or 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 3-233. 函数 enet\_frame\_transmit**

函数名称	enet_frame_transmit
函数原型	ErrStatus enet_frame_transmit(uint8_t *buffer, uint32_t length);
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	-
<b>输入参数{in}</b>	
<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
<b>输入参数{in}</b>	
<b>length</b>	待发送数据长度，范围(0~1524)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>ErrStatus</b>	ERROR or 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 3-234. 函数 enet\_transmit\_checksum\_config**

函数名称	enet_transmit_checksum_config
------	-------------------------------

函数原型	void enet_transmit_checksum_config(enet_descriptors_struct *desc, uint32_t checksum);
功能描述	配置发送帧校验和模式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<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="#">结构体 <i>enet_descriptors_struct</i></a>
<b>输入参数{in}</b>	
<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_TCUPDPICMP_SEGM</i>	TCP/UDP/ICMP校验和（除去伪报头）计算和插入
<i>ENET_CHECKSUM_TCUPDPICMP_FUL</i>	TCP/UDP/ICMP校验和计算和插入
<b>输出参数{out}</b>	
<b>返回值</b>	

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_TCUPDPICMP_FULL);
```

### 函数 **enet\_enable**

函数**enet\_enable**描述见下表：

表 3-235. 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 3-236. 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 3-237. 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	-
<b>输入参数{in}</b>	
<b>mac_addr</b>	选择何组MAC地址将被配置，下列参数仅可选择一个
<i>ENET_MAC_ADDRESS0</i>	配置MAC address 0过滤器
<i>ENET_MAC_ADDRESS1</i>	配置MAC address 1过滤器
<i>ENET_MAC_ADDRESS2</i>	配置MAC address 2过滤器
<i>ENET_MAC_ADDRESS3</i>	配置MAC address 3过滤器
<b>输入参数{in}</b>	
<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>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-238. 函数 **enet\_mac\_address\_get****

函数名称	enet_mac_address_get
函数原型	void enet_mac_address_get(enet_macaddress_enum mac_addr, uint8_t paddr[])
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，下列参数仅可选择一个
<b>ENET_MAC_ADDRESS0</b>	配置MAC address 0过滤器
<b>ENET_MAC_ADDRESS1</b>	配置MAC address 1过滤器
<b>ENET_MAC_ADDRESS2</b>	配置MAC address 2过滤器
<b>ENET_MAC_ADDRESS3</b>	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}
返回值	
-	-

例如：

```
/* get mac address */
enet_mac_address_get (ENET_MAC_ADDRESS0, netif->hwaddr);
```

### 函数 **enet\_flag\_get**

函数 **enet\_flag\_get** 描述见下表：

**表 3-239. 函数 **enet\_flag\_get****

函数名称	enet_flag_get
函数原型	FlagStatus enet_flag_get(enet_flag_enum enet_flag);
功能描述	获取ENET模块MAC/MSC/PTP/DMA状态标志位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>enet_flag</b>	ENET状态标志位，下列参数仅可选择一个
<i>ENET_MAC_FLAG_MPKR</i>	接收到魔术帧标志位
<i>ENET_MAC_FLAG_WUFR</i>	接收到唤醒帧标志位
<i>ENET_MAC_FLAG_FLOWCONTROL</i>	流控状态标志位
<i>ENET_MAC_FLAG_WUM</i>	WUM状态标志位
<i>ENET_MAC_FLAG_MSC</i>	MSC状态标志位
<i>ENET_MAC_FLAG_MSCR</i>	MSC接收状态标志位
<i>ENET_MAC_FLAG_MSCT</i>	MSC发送状态标志位

<i>ENET_MAC_FLAG_TMST</i>	时间戳触发状态标志位
<i>ENET_PTP_FLAG_TSSCO</i>	时间戳秒计数溢出标志位
<i>ENET_PTP_FLAG_TTM</i>	目标时间匹配标志位
<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
<i>ENET_DMA_FLAG_EB_DMA_ERROR</i>	DMA错误标志位
<i>ENET_DMA_FLAG_EB_TRANSFER_ER_ROR</i>	发送错误标志位
<i>ENET_DMA_FLAG_EB_ACCESS_ERRO_R</i>	DMA访问错误标志位
<i>ENET_DMA_FLAG_MSC</i>	MSC状态标志位
<i>ENET_DMA_FLAG_WUM</i>	WUM状态标志位
<i>ENET_DMA_FLAG_TST</i>	时间戳触发状态标志位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	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 3-240. 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}	
<b>enet_flag</b>	ENET模块DMA标志位清除，下列参数仅可选择一个
<i>ENET_DMA_FLAG_TS_CLR</i>	发送状态标志位清除
<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_RS_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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-241. 函数 **enet\_interrupt\_enable****

函数名称	<b>enet_interrupt_enable</b>
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	-
<b>输入参数{in}</b>	
<b>enet_int</b>	ENET中断，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_T</i>	时间戳触发中断屏蔽

<i>MSTIM</i>	
<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_TE</i>	发送中断使能
<i>ENET_DMA_INT_TPSIE</i>	发送流程停止中断使能
<i>ENET_DMA_INT_TBUIE</i>	发送缓冲区不可用中断使能
<i>ENET_DMA_INT_TJTI</i>	发送jabber超时中断使能
<i>ENET_DMA_INT_ROIE</i>	接收溢出中断使能
<i>ENET_DMA_INT_TUIE</i>	发送下溢中断使能
<i>ENET_DMA_INT_RI</i> <i>E</i>	接收中断使能
<i>ENET_DMA_INT_RBUIE</i>	接收缓冲区不可用中断使能
<i>ENET_DMA_INT_RPSIE</i>	接收流程停止中断使能
<i>ENET_DMA_INT_RWTIE</i>	接收看门狗超时中断使能
<i>ENET_DMA_INT_ET</i>	早发送中断使能

<i>IE</i>	
<i>ENET_DMA_INT_FB_EIE</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_NIE</i>	正常中断汇总使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-242. 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	-
<b>输入参数{in}</b>	
<b>enet_int</b>	ENET中断，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_TIMSTIM</i>	时间戳触发中断屏蔽

<i>ENET_MSC_INT_RF_CEIM</i>	接收帧CRC错误中断屏蔽
<i>ENET_MSC_INT_RF_AEIM</i>	接收帧对齐错误中断屏蔽
<i>ENET_MSC_INT_R_GUFIM</i>	接收“好”单播帧中断屏蔽
<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_TI_E</i>	发送中断使能
<i>ENET_DMA_INT_TP_SIE</i>	发送流程停止中断使能
<i>ENET_DMA_INT_TB_UIE</i>	发送缓冲区不可用中断使能
<i>ENET_DMA_INT_TJ_TIE</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_TU_IE</i>	发送下溢中断使能
<i>ENET_DMA_INT_RI_E</i>	接收中断使能
<i>ENET_DMA_INT_RBUIE</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_ET_IE</i>	早发送中断使能

<i>ENET_DMA_INT_FB_EIE</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_NIE</i>	正常中断汇总使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-243. 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	-
<b>输入参数{in}</b>	
<b>int_flag</b>	ENET中断标志位，下列参数仅可选择一个
<i>ENET_MAC_INT_FL_AG_WUM</i>	WUM中断标志位
<i>ENET_MAC_INT_FL_AG_MSC</i>	MSC中断标志位
<i>ENET_MAC_INT_FL</i>	MSC接收中断标志位

<i>AG_MSCR</i>	
<i>ENET_MAC_INT_FL</i> <i>AG_MSCT</i>	MSC发送中断标志位
<i>ENET_MAC_INT_FL</i> <i>AG_TMST</i>	时间戳触发中断标志位
<i>ENET_MSC_INT_FL</i> <i>AG_RFCE</i>	接收帧CRC错误中断标志位
<i>ENET_MSC_INT_FL</i> <i>AG_RFAE</i>	接收帧对齐错误中断标志位
<i>ENET_MSC_INT_FL</i> <i>AG_RGUF</i>	接收“好”单播帧中断标志位
<i>ENET_MSC_INT_FL</i> <i>AG_TGFSC</i>	仅遇到1个冲突后发送“好”帧中断标志位
<i>ENET_MSC_INT_FL</i> <i>AG_TGFMSC</i>	遇到1个以上冲突后发送“好”帧中断标志位
<i>ENET_MSC_INT_FL</i> <i>AG_TGF</i>	发送“好”的帧的中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_TS</i>	发送中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_TPS</i>	发送流程停止中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_TBU</i>	发送缓冲区不可用中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_TJT</i>	发送jabber超时中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_RO</i>	接收溢出中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_TU</i>	发送下溢中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_RS</i>	接收中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_RBU</i>	接收缓冲区不可用中断标志位
<i>ENET_DMA_INT_FL</i>	接收流程停止中断标志位

<i>AG_RPS</i>	
<i>ENET_DMA_INT_FL</i> <i>AG_RWT</i>	接收看门狗超时中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_ET</i>	早发送中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_FBE</i>	致命总线错误中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_ER</i>	早接收中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_AI</i>	异常中断汇总中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_NI</i>	正常中断汇总中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_MSC</i>	MSC中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_WUM</i>	WUM中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_TST</i>	时间戳触发中断标志位
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	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 3-244. 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);

<b>功能描述</b>	禁能ENET中断标志位
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>int_flag_clear</b>	ENET中断标志位清除，下列参数仅可选择一个
<i>ENET_DMA_INT_FL</i> <i>AG_TS_CLR</i>	发送中断标志位清除
<i>ENET_DMA_INT_FL</i> <i>AG_TPS_CLR</i>	发送流程停止中断标志位清除
<i>ENET_DMA_INT_FL</i> <i>AG_TBU_CLR</i>	发送缓冲区不可用中断标志位清除
<i>ENET_DMA_INT_FL</i> <i>AG_TJT_CLR</i>	发送jabber超时中断标志位清除
<i>ENET_DMA_INT_FL</i> <i>AG_RO_CLR</i>	接收溢出中断标志位清除
<i>ENET_DMA_INT_FL</i> <i>AG_TU_CLR</i>	发送下溢中断标志位清除
<i>ENET_DMA_INT_FL</i> <i>AG_RS_CLR</i>	接收中断标志位清除
<i>ENET_DMA_INT_FL</i> <i>AG_RBU_CLR</i>	接收缓冲区不可用中断标志位清除
<i>ENET_DMA_INT_FL</i> <i>AG_RPS_CLR</i>	接收流程停止中断标志位清除
<i>ENET_DMA_INT_FL</i> <i>AG_RWT_CLR</i>	接收看门狗超时中断标志位清除
<i>ENET_DMA_INT_FL</i> <i>AG_ET_CLR</i>	早发送中断标志位清除
<i>ENET_DMA_INT_FL</i> <i>AG_FBE_CLR</i>	致命总线错误中断标志位清除
<i>ENET_DMA_INT_FL</i> <i>AG_ER_CLR</i>	早接收中断标志位清除
<i>ENET_DMA_INT_FL</i> <i>AG_AI_CLR</i>	异常中断汇总中断标志位清除

<i>ENET_DMA_INT_FL</i>	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 3-245. 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 3-246. 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 3-247. 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 3-248. 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	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-249. 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

<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>type</b>	寄存器类型，下列参数仅可选择一个
<i>ALL_MAC_REG</i>	寄存器范围从ENET_MAC_CFG到ENET_MAC_FCTH
<i>ALL_MSC_REG</i>	寄存器范围从ENET_MSC_CTL到ENET_MSC_RGUFHCNT
<i>ALL_PTP_REG</i>	寄存器范围从ENET_PTP_TSCTL到ENET_PTP_PPSCTL
<i>ALL_DMA_REG</i>	寄存器范围从ENET_DMA_BCTL到ENET_DMA_CRBADDR
<b>输入参数{in}</b>	
<b>num</b>	想要获取的寄存器个数，范围(0~54)
<b>输出参数{out}</b>	
<b>preg</b>	存储寄存器值的应用缓冲区指针
<b>返回值</b>	
-	-

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 3-250. 函数 **enet\_debug\_status\_get****

<b>函数名称</b>	enet_debug_status_get
<b>函数原型</b>	uint32_t enet_debug_status_get(uint32_t mac_debug);
<b>功能描述</b>	获取ENET调试状态信息
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>mac_debug</b>	ENET调试信息选项，下列参数仅可选择一个

<i>ENET_MAC_RECEIVER_NOT_IDLE</i>	MAC接收器非空
<i>ENET_RX_ASYNCHRONOUS_FIFO_STATUS</i>	异步接收FIFO状态
<i>ENET_RXFIFO_WRITING</i>	接收FIFO正在执行写操作
<i>ENET_RXFIFO_READ_STATUS</i>	读取接收FIFO操作状态
<i>ENET_RXFIFO_STATUS</i>	接收FIFO状态
<i>ENET_MAC_TRANSMITTER_NOT_IDLE</i>	MAC发送器非空
<i>ENET_MAC_TRANSMITTER_STATUS</i>	MAC发送器状态
<i>ENET_PAUSE_CONDITION_STATUS</i>	暂停条件状态
<i>ENET_TXFIFO_READ_STATUS</i>	读取发送FIFO操作状态
<i>ENET_TXFIFO_WRITING</i>	发送FIFO正在执行写操作
<i>ENET_TXFIFO_NOT_EMPTY</i>	发送FIFO非空
<i>ENET_TXFIFO_FULL</i>	发送FIFO已满
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint32_t</b>	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 3-251. 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	--
<b>输入参数{in}</b>	
mac_addr	选择被使能的MAC地址组，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_ADDRESS3	使能MAC地址组3过滤器
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-252. 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	-
<b>输入参数{in}</b>	
<b>mac_addr</b>	选择被使能的MAC地址组，下列参数仅可选择一个
<i>ENET_MAC_ADDRESS1</i>	使能MAC地址组1过滤器
<i>ENET_MAC_ADDRESS2</i>	使能MAC地址组2过滤器
<i>ENET_MAC_ADDRESS3</i>	使能MAC地址组3过滤器
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-253. 函数 **enet\_address\_filter\_config****

函数名称	<b>enet_address_filter_config</b>
函数原型	void enet_address_filter_config(enet_macaddress_enum mac_addr, uint32_t addr_mask, uint32_t filter_type);
功能描述	配置MAC地址过滤器模式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>mac_addr</b>	选择被使能的MAC地址组，下列参数仅可选择一个
<i>ENET_MAC_ADDRESS1</i>	使能MAC地址组1过滤器

<i>ENET_MAC_ADDRESS2</i>	使能MAC地址组2过滤器
<i>ENET_MAC_ADDRESS3</i>	使能MAC地址组3过滤器
<b>输入参数{in}</b>	
<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
<i>ENET_ADDRESS_MASK_BYTE0</i>	屏蔽ENET_MAC_ADDR1L[7:0] bits
<i>ENET_ADDRESS_MASK_BYTE1</i>	屏蔽ENET_MAC_ADDR1L[15:8] bits
<i>ENET_ADDRESS_MASK_BYTE2</i>	屏蔽ENET_MAC_ADDR1L[23:16] bits
<i>ENET_ADDRESS_MASK_BYTE3</i>	屏蔽ENET_MAC_ADDR1L [31:24] bits
<i>ENET_ADDRESS_MASK_BYTE4</i>	屏蔽ENET_MAC_ADDR1H [7:0] bits
<i>ENET_ADDRESS_MASK_BYTE5</i>	屏蔽ENET_MAC_ADDR1H [15:8] bits
<b>输入参数{in}</b>	
<b>filter_type</b>	选择MAC地址过滤器类型, 下列参数仅可选择一个
<i>ENET_ADDRESS_FILTER_SA</i>	过滤器比对接收帧MAC地址的源地址域
<i>ENET_ADDRESS_FILTER_DA</i>	过滤器比对接收帧MAC地址的目的地址域
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-254. 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SMII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 or 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 3-255. 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}	

<b>direction</b>	下列参数仅可选择一个
<i>ENET_PHY_WRITE</i>	向PHY寄存器写数据
<i>ENET_PHY_READ</i>	从PHY寄存器读数据
<b>输入参数{in}</b>	
<b>phy_address</b>	0x0 - 0x1F
<b>输入参数{in}</b>	
<b>phy_reg</b>	0x0 - 0x1F
<b>输入参数{in}</b>	
<b>pvalue</b>	当 <b>direction</b> 选择 <i>ENET_PHY_WRITE</i> 时，表示将写入PHY寄存器的值
<b>输出参数{out}</b>	
<b>pvalue-</b>	当 <b>direction</b> 选择 <i>ENET_PHY_READ</i> 时，表示将存储PHY寄存器读出的值
<b>返回值</b>	
<b>ErrStatus</b>	ERROR or 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 3-256. 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()
<b>输入参数{in}</b>	
-	-

输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR or 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 3-257. 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	
<b>ErrStatus</b>	ERROR or 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 3-258. 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
<i>ENET_AUTO_PADCRC_DROP</i>	自动去除接收帧填充字节和FCS域
<i>ENET_TYPEFRAME_CRC_DROP</i>	帧通过前自动去除FCS域最后四个字节的CRC校验和
<i>ENET_FORWARD_ERRFRAMES</i>	除了过短帧外的其他错误帧都会转发给应用
<i>ENET_FORWARD_UNDERSZ_GOODFRAMES</i>	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_UNDERSZ_GOODFRAMES);
```

#### 函数 enet\_forward\_feature\_disable

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

表 3-259. 函数 enet\_forward\_feature\_disable

函数名称	enet_forward_feature_disable
函数原型	void enet_forward_feature_disable(uint32_t feature);
功能描述	禁能ENET帧通过相关功能

先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>feature</b>	ENET帧通过功能，下列参数可以选择多个
<i>ENET_AUTO_PADC_RC_DROP</i>	自动去除接收帧填充字节和FCS域
<i>ENET_TYPEFRAME_CRC_DROP</i>	帧通过前自动去除FCS域最后四个字节的CRC校验和
<i>ENET_FORWARD_ERRFRAMES</i>	除了过短帧外的其他错误帧都会转发给应用
<i>ENET_FORWARD_UNDERSZ_GOODFRAMES</i>	帧长小于64字节但没有错误的帧将转发给应用
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable the function that forwarding undersized good frames */
enet_forward_feature_enable(ENET_FORWARD_UNDERSZ_GOODFRAMES);
```

#### 函数 **enet\_filter\_feature\_enable**

函数**enet\_filter\_feature\_enable**描述见下表：

**表 3-260. 函数 **enet\_filter\_feature\_enable****

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

<i>ENET_SRC_FILTER</i>	源地址过滤器
<i>ENET_SRC_FILTER_INVERSE</i>	源地址过滤结果逆转
<i>ENET_DEST_FILTER_INVERSE</i>	目的地址过滤结果逆转
<i>ENET_MULTICAST_FILTER_PASS</i>	接收多播帧
<i>ENET_MULTICAST_FILTER_HASH_MODE</i>	HASH多播过滤器
<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_filter_feature_enable(ENET_SRC_FILTER);
```

#### 函数 **enet\_filter\_feature\_disable**

函数 **enet\_filter\_feature\_disable** 描述见下表：

表 3-261. 函数 **enet\_filter\_feature\_disable**

函数名称	enet_filter_feature_disable
函数原型	void enet_filter_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}	

<b>feature</b>	ENET过滤器功能，下列参数可以选择多个
<i>ENET_SRC_FILTER</i>	源地址过滤器
<i>ENET_SRC_FILTER_INVERSE</i>	源地址过滤结果逆转
<i>ENET_DEST_FILTER_INVERSE</i>	目的地址过滤结果逆转
<i>ENET_MULTICAST_FILTER_PASS</i>	接收多播帧
<i>ENET_MULTICAST_FILTER_HASH_MODE</i>	HASH多播过滤器
<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_filter_feature_enable(ENET_SRC_FILTER);
```

#### 函数 **enet\_pauseframe\_generate**

函数**enet\_pauseframe\_generate**描述见下表：

表 3-262. 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	
<b>ErrStatus</b>	ERROR or SUCCESS

例如：

```
/* generate the pause frame */

ErrStatus rval;

rval = enet_pauseframe_generate();
```

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

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

表 3-263. 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}	
<b>detect</b>	暂停帧检测，下列参数仅可选择一个
<i>ENET_MAC0_AND_UNIQE_ADDRESS_PAUSEDTECT</i>	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
<i>ENET_UNIQUE_PAUSEDETECT</i>	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_PAUSEDTECT */

enet_pauseframe_detect_config(ENET_UNIQUE_PAUSEDTECT);
```

### 函数 **enet\_pauseframe\_config**

函数enet\_pauseframe\_config描述见下表：

**表 3-264. 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}	
<b>pausetime</b>	暂停控制帧时间域，范围(0~0xFFFF)
输入参数{in}	
<b>pause_threshold</b>	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
<i>ENET_PAUSETIME_MINUS4</i>	暂停时间 - 4个时间间隙
<i>ENET_PAUSETIME_MINUS28</i>	暂停时间 - 28个时间间隙
<i>ENET_PAUSETIME_MINUS144</i>	暂停时间 - 144个时间间隙
<i>ENET_PAUSETIME_MINUS256</i>	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 3-265. 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}	
<b>deactive</b>	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Rx FIFO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
<i>ENET_DEACTIVE_T_HRESHOLD_256BYTES</i>	256字节
<i>ENET_DEACTIVE_T_HRESHOLD_512BYTES</i>	512字节
<i>ENET_DEACTIVE_T_HRESHOLD_768BYTES</i>	768字节
<i>ENET_DEACTIVE_T_HRESHOLD_1024BYTES</i>	1024字节
<i>ENET_DEACTIVE_T_HRESHOLD_1280BYTES</i>	1280字节
<i>ENET_DEACTIVE_T_HRESHOLD_1536BYTES</i>	1536字节
<i>ENET_DEACTIVE_T_HRESHOLD_1792BYTES</i>	1792字节

<i>HRESHOLD_1792BYTES</i>	
<b>输入参数{in}</b>	
<b>active</b>	流控激活的阈值。若使能了流控功能，当Rx FIFO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
<i>ENET_ACTIVE_THRESHOLD_256BYTES</i>	256字节
<i>ENET_ACTIVE_THRESHOLD_512BYTES</i>	512字节
<i>ENET_ACTIVE_THRESHOLD_768BYTES</i>	768字节
<i>ENET_ACTIVE_THRESHOLD_1024BYTES</i>	1024字节
<i>ENET_ACTIVE_THRESHOLD_1280BYTES</i>	1280字节
<i>ENET_ACTIVE_THRESHOLD_1536BYTES</i>	1536字节
<i>ENET_ACTIVE_THRESHOLD_1792BYTES</i>	1792字节
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-266. 函数 `enet_flowcontrol_feature_enable`**

函数名称	<code>enet_flowcontrol_feature_enable</code>
函数原型	<code>void enet_flowcontrol_feature_enable(uint32_t feature);</code>
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}	
<code>feature</code>	ENET流控功能模式，下列参数可以选择多个
<code>ENET_ZERO_QUANTA_PAUSE</code>	零时间片暂停控制帧自动生成
<code>ENET_TX_FLOWCONTROL</code>	发送流控功能
<code>ENET_RX_FLOWCONTROL</code>	接收流控功能
<code>ENET_BACK_PRESSURE</code>	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 3-267. 函数 `enet_flowcontrol_feature_disable`**

函数名称	<code>enet_flowcontrol_feature_disable</code>
------	---

<b>函数原型</b>	void enet_flowcontrol_feature_disable(uint32_t feature);
<b>功能描述</b>	禁能ENET流控相关功能
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>feature</b>	ENET流控功能模式, 下列参数可以选择多个
<i>ENET_ZERO_QUAN TA_PAUSE</i>	零时间片暂停控制帧自动生成
<i>ENET_TX_FLOWCO NTROL</i>	发送流控功能
<i>ENET_RX_FLOWC ONTROL</i>	发送流控功能
<i>ENET_BACK_PRES SURE</i>	背压功能 (仅在半双工模式下)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-268. 函数 enet\_dmaprocess\_state\_get**

<b>函数名称</b>	enet_dmaprocess_state_get
<b>函数原型</b>	uint32_t enet_dmaprocess_state_get(enet_dmadirection_enum direction);
<b>功能描述</b>	获取DMA发送/接收流程状态
<b>先决条件</b>	-
<b>被调用函数</b>	-

输入参数{in}	
<b>direction</b>	DMA传输方向
<i>ENET_DMA_TX</i>	DMA发送进程
<i>ENET_DMA_RX</i>	DMA接收进程
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	DMA流程状态，可取值如下： <i>ENET_RX_STATE_STOPPED / ENET_RX_STATE_FETCHING /</i> <i>ENET_RX_STATE_WAITING / ENET_RX_STATE_SUSPENDED /</i> <i>ENET_RX_STATE_CLOSING / ENET_RX_STATE_QUEUEING /</i> <i>ENET_TX_STATE_STOPPED / ENET_TX_STATE_FETCHING /</i> <i>ENET_TX_STATE_WAITING / ENET_TX_STATE_READING /</i> <i>ENET_TX_STATE_SUSPENDED / ENET_TX_STATE_CLOSING</i>

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 3-269. 函数 **enet\_dmaprocess\_resume****

函数名称	<b>enet_dmaprocess_resume</b>
函数原型	<b>void enet_dmaprocess_resume(enet_dmadirection_enum direction);</b>
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
<i>ENET_DMA_TX</i>	DMA发送进程
<i>ENET_DMA_RX</i>	DMA接收进程
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-270. 函数 enet\_rxprocess\_check\_recovery**

<b>函数名称</b>	enet_rxprocess_check_recovery
<b>函数原型</b>	void enet_rxprocess_check_recovery(void);
<b>功能描述</b>	检测并恢复接收流程
<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：

```
/* 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 3-271. 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	
<b>ErrStatus</b>	ERROR or 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 3-272. 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}	
<b>addr_get</b>	可获取的描述符地址类型，下列参数仅可选择一个

<i>ENET_RX_DESC_TABLE</i>	接收描述符列表首地址
<i>ENET_RX_CURRENT_DESC</i>	当前DMA控制器使用的接收描述符地址
<i>ENET_RX_CURRENT_BUFFER</i>	当前DMA控制器使用的接收描述符缓冲区地址
<i>ENET_TX_DESC_TABLE</i>	发送描述符列表首地址
<i>ENET_TX_CURRENT_DESC</i>	当前DMA控制器使用的发送描述符地址
<i>ENET_TX_CURRENT_BUFFER</i>	当前DMA控制器使用的发送描述符缓冲区地址
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint32_t</b>	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 3-273. 函数 *enet\_desc\_information\_get***

函数名称	<i>enet_desc_information_get</i>
函数原型	<i>uint32_t enet_desc_information_get(enet_descriptors_struct *desc, enet_descstate_enum info_get);</i>
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	-
<b>输入参数{in}</b>	

<b>desc</b>	描述符指针，结构体成员介绍请参考 <a href="#">结构体enet_descriptors_struct</a>
<b>输入参数{in}</b>	
<b>info_get</b>	可选择的描述符信息类型，下列参数仅可选择一个
<i>RXDESC_BUFFER_1_SIZE</i>	接收缓冲区1大小
<i>RXDESC_BUFFER_2_SIZE</i>	接收缓冲区2大小
<i>RXDESC_FRAME_LENGTH</i>	接收帧长度
<i>TXDESC_COLLISION_N_COUNT</i>	帧发送出去前出现的冲突次数
<i>RXDESC_BUFFER_1_ADDR</i>	接收帧的缓冲区地址
<i>TXDESC_BUFFER_1_ADDR</i>	发送帧的缓冲区地址
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint32_t</b>	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 raval;

raval = 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 3-274. 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 *rx fifo_drop, uint32_t *rx dma_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}	
<b>rxfifo_drop</b>	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}	
<b>rxdma_drop</b>	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 3-275. 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}	
<b>desc</b>	描述符指针，结构体成员介绍请参考 <a href="#">结构体enet_descriptors_struct</a>
输入参数{in}	
<b>desc_flag</b> (the value according to the parameter)	描述符标志位，下列参数仅可选择一个

<b>desc)</b>	
当 <b>desc</b> 参数为发送描述符时	
<i>ENET_TDES0_DB</i>	顺延位
<i>ENET_TDES0_UFE</i>	数据下溢错误位
<i>ENET_TDES0_EXD</i>	过度顺延位
<i>ENET_TDES0_VFR_M</i>	VLAN帧位
<i>ENET_TDES0_ECO</i>	过度冲突位
<i>ENET_TDES0_LCO</i>	延迟冲突位
<i>ENET_TDES0_NCA</i>	无载波位
<i>ENET_TDES0_LCA</i>	载波丢失位
<i>ENET_TDES0_IPPE</i>	IP数据错误位
<i>ENET_TDES0_FRM_F</i>	帧清空位
<i>ENET_TDES0_JT</i>	Jabber超时位
<i>ENET_TDES0_ES</i>	错误汇总
<i>ENET_TDES0_IPHE</i>	IP报头错误位
<i>ENET_TDES0_TTM_SS</i>	发送时间戳状态位
<i>ENET_TDES0_TCH_M</i>	第二地址链表模式位
<i>ENET_TDES0_TER_M</i>	环形发送结束模式位
<i>ENET_TDES0_TTS_EN</i>	使能发送时间戳位
<i>ENET_TDES0_DPA_D</i>	不填充位
<i>ENET_TDES0_DCR_C</i>	不计算CRC位
<i>ENET_TDES0_FSG</i>	第一分块位
<i>ENET_TDES0_LSG</i>	最后分块位

<i>ENET_TDES0_INTC</i>	完成时中断位
<i>ENET_TDES0_DAV</i>	DAV位
当 <b>desc</b> 参数为接收描述符时	
<i>ENET_RDES0_PCE_RR</i>	数据校验和错误
<i>ENET_RDES0_CER_R</i>	CRC错误
<i>ENET_RDES0_DBEE_RR</i>	Dribble位错误
<i>ENET_RDES0_RER_R</i>	接收错误
<i>ENET_RDES0_RWD_T</i>	接收看门狗超时
<i>ENET_RDES0_FRM_T</i>	帧类型
<i>ENET_RDES0_LCO</i>	延迟冲突位
<i>ENET_RDES0_IPHE_RR</i>	IP帧报头校验和错误
<i>ENET_RDES0_LDE_S</i>	最后一个描述符
<i>ENET_RDES0_FDE_S</i>	第一个描述符
<i>ENET_RDES0_VTAG</i>	VLAN标签位
<i>ENET_RDES0_OER_R</i>	溢出错误位
<i>ENET_RDES0_LER_R</i>	长度错误位
<i>ENET_RDES0_SAF_F</i>	未通过源地址过滤器位
<i>ENET_RDES0_DER_R</i>	描述符错误位
<i>ENET_RDES0_ERR</i>	错误汇总位

S	
<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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	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 3-276. 函数 *enet\_desc\_flag\_set***

函数名称	<i>enet_desc_flag_set</i>
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	-
<b>输入参数{in}</b>	
<b>desc</b>	描述符指针，结构体成员介绍请参考 <a href="#">结构体<i>enet_descriptors_struct</i></a>
<b>输入参数{in}</b>	
<b>desc_flag</b> (the value according to the parameter <b>desc</b> )	描述符标志位，下列参数仅可选择一个
当 <b>desc</b> 参数为发送描述符时	
<i>ENET_TDES0_VFR</i> <i>M</i>	VLAN帧位

<i>ENET_TDES0_FRM F</i>	帧清空位
<i>ENET_TDES0_TCH M</i>	第二地址链表模式位
<i>ENET_TDES0_TER M</i>	环形发送结束模式位
<i>ENET_TDES0_TTS EN</i>	使能发送时间戳位
<i>ENET_TDES0_DPA D</i>	不填充位
<i>ENET_TDES0_DCR C</i>	不计算CRC位
<i>ENET_TDES0_FSG</i>	第一分块位
<i>ENET_TDES0_LSG</i>	最后分块位
<i>ENET_TDES0_INTC</i>	完成时中断位
<i>ENET_TDES0_DAV</i>	DAV位
当 <i>desc</i> 参数为接收描述符时	
<i>ENET_RDES0_DAV</i>	描述符可用位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-277. 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);

<b>功能描述</b>	清除ENET模块DMA描述符标志位
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>desc</b>	描述符指针, 结构体成员介绍请参考 <a href="#"><u>结构体enet_descriptors_struct</u></a>
<b>输入参数{in}</b>	
<b>desc_flag</b>  (the value according to the parameter <b>desc</b> )	描述符标志位, 下列参数仅可选择一个
当 <b>desc</b> 参数为发送描述符时	
<i>ENET_TDES0_VFR</i> <i>M</i>	VLAN帧位
<i>ENET_TDES0_FRM</i> <i>F</i>	帧清空位
<i>ENET_TDES0_TCH</i> <i>M</i>	第二地址链表模式位
<i>ENET_TDES0_TER</i> <i>M</i>	环形发送结束模式位
<i>ENET_TDES0_TTS</i> <i>EN</i>	使能发送时间戳位
<i>ENET_TDES0_DPA</i> <i>D</i>	不填充位
<i>ENET_TDES0_DCR</i> <i>C</i>	不计算CRC位
<i>ENET_TDES0_FSG</i>	第一分块位
<i>ENET_TDES0_LSG</i>	最后分块位
<i>ENET_TDES0_INTC</i>	完成时中断位
<i>ENET_TDES0_DAV</i>	DAV位
当 <b>desc</b> 参数为接收描述符时	
<i>ENET_RDES0_DAV</i>	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 3-278. 函数 **enet\_rx\_desc\_immediate\_receive\_complete\_interrupt****

函数名称	enet_rx_desc_immediate_receive_complete_interrupt
函数原型	void enet_rx_desc_immediate_receive_complete_interrupt(enet_descriptors_struct *desc);
功能描述	当接收完成时，立即置位ENET_DMA_STAT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="#">结构体enet_descriptors_struct</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 3-279. 函数 enet\_rx\_desc\_delay\_receive\_complete\_interrupt

函数名称	enet_rx_desc_delay_receive_complete_interrupt
函数原型	void enet_rx_desc_delay_receive_complete_interrupt(enet_descriptors_struct *desc, uint32_t delay_time);
功能描述	当接收完成时，延迟指定时间再置位ENET_DMA_STAT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="#">结构体enet_descriptors_struct</a>
输入参数{in}	
delay_time	延迟时间，实际延迟时间为256*delay_time个HCLK (0~0xFF)
输出参数{out}	
-	-
返回值	
-	-

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 3-280. 函数 enet\_rxframe\_drop

函数名称	enet_rxframe_drop
函数原型	void enet_rxframe_drop(void);
功能描述	丢弃当前接收到的帧
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
-	-

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 3-281. 函数 **enet\_dma\_feature\_enable****

函数名称	enet_dma_feature_enable
函数原型	void enet_dma_feature_enable(uint32_t feature);
功能描述	使能ENET模块DMA相关功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>feature</b>	DMA功能，下列参数可以选择多个
<i>ENET_NO_FLUSH_RXFRAME</i>	描述符不可用时，RxDMA控制器清空接收帧功能
<i>ENET_SECONDFRAMEOPT</i>	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 3-282. 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 3-283. 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	-

<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>desc</b>	描述符指针, 结构体成员介绍请参考 <a href="#">结构体enet_descriptors_struct</a>
<b>输入参数{in}</b>	
<b>desc_status</b>	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_PTOEF	PTP网络帧
ENET_RDES4_PTPVF	PTP版本格式
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint32_t</b>	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 3-284. 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 3-285. 函数 enet\_ptp\_enhanced\_descriptors\_chain\_init**

函数名称	enet_ptp_enhanced_descriptors_chain_init
函数原型	void enet_ptp_enhanced_descriptors_chain_init(enet_dmadirection_enum direction);
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}	
direction	描述符类型，下列参数仅可选择一个

<i>ENET_DMA_TX</i>	DMA Tx描述符
<i>ENET_DMA_RX</i>	DMA Rx描述符
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-286. 函数 **enet\_ptp\_enhanced\_descriptors\_ring\_init**

函数名称	enet_ptp_enhanced_descriptors_ring_init
函数原型	void enet_ptp_enhanced_descriptors_ring_init(enet_dmadirection_enum 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	-
<b>输入参数{in}</b>	
<b>direction</b>	描述符类型，下列参数仅可选择一个
<i>ENET_DMA_TX</i>	DMA Tx描述符
<i>ENET_DMA_RX</i>	DMA Rx描述符
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-287. 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	-
<b>输入参数{in}</b>	
<b>bufsize</b>	缓冲区大小
<b>输出参数{out}</b>	
<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
<b>输出参数{out}</b>	
<b>timestamp</b>	存放时间戳指针
<b>返回值</b>	
<b>ErrStatus</b>	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 3-288. 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	-
<b>输入参数{in}</b>	
<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指定的位置
<b>输入参数{in}</b>	
<b>length</b>	发送数据大小
<b>输出参数{out}</b>	
<b>timestamp</b>	存放时间戳的指针，如果输入为NULL，则忽略时间戳
<b>返回值</b>	
<b>ErrStatus</b>	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 3-289. 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	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-290. 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	-
<b>输入参数{in}</b>	
<b>direction</b>	描述符类型，下列参数仅可选择一个
<i>ENET_DMA_TX</i>	DMA Tx描述符
<i>ENET_DMA_RX</i>	DMA Rx描述符
<b>输入参数{in}</b>	
<b>desc_ptptab</b>	描述符指针，结构体成员介绍请参考 <a href="#">结构体enet_descriptors_struct</a>
<b>输出参数{out}</b>	
-	-

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 3-291. 函数 `enet_ptp_normal_descriptors_ring_init`

函数名称	enet_ptp_normal_descriptors_ring_init
函数原型	void enet_ptp_normal_descriptors_ring_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
<i>ENET_DMA_TX</i>	DMA Tx描述符
<i>ENET_DMA_RX</i>	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="#">结构体 <code>enet_descriptors_struct</code></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 3-292. 函数 **enet\_ptpframe\_receive\_normal\_mode****

函数名称	enet_ptpframe_receive_normal_mode
函数原型	ErrStatus enet_ptpframe_receive_normal_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	-
<b>输入参数{in}</b>	
<b>bufsize</b>	缓冲区大小
<b>输出参数{out}</b>	
<b>timestamp</b>	存放时间戳指针
<b>输出参数{out}</b>	
<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
<b>返回值</b>	
<b>ErrStatus</b>	ERROR or 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 3-293. 函数 enet\_ptpframe\_transmit\_normal\_mode**

<b>函数名称</b>	enet_ptpframe_transmit_normal_mode
<b>函数原型</b>	ErrStatus enet_ptpframe_transmit_normal_mode(uint8_t *buffer, uint32_t length, uint32_t timestamp[]);
<b>功能描述</b>	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，并同时间戳一起发送
<b>先决条件</b>	-
<b>被调用函数</b>	--
<b>输入参数{in}</b>	
<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指定的位置
<b>输入参数{in}</b>	
<b>length</b>	发送数据大小
<b>输出参数{out}</b>	
<b>timestamp</b>	存放时间戳的指针，如果输入为NULL，则忽略时间戳
<b>返回值</b>	
<b>ErrStatus</b>	ERROR or 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 3-294. 函数 enet\_wum\_filter\_register\_pointer\_reset**

<b>函数名称</b>	enet_wum_filter_register_pointer_reset
<b>函数原型</b>	void enet_wum_filter_register_pointer_reset(void);
<b>功能描述</b>	远程唤醒帧过滤器寄存器指针复位

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

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 3-295. 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	-
<b>输入参数{in}</b>	
<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）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-296. 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
<i>ENET_WUM_POWER_DOWN</i>	掉电模式
<i>ENET_WUM_MAGIC_PACKET_FRAME</i>	使能接收到魔法帧的唤醒事件
<i>ENET_WUM_WAKE_UP_FRAME</i>	使能接收到唤醒帧的唤醒事件
<i>ENET_WUM_GLOBAL_UNICAST</i>	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 3-297. 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>	
feature	下列参数可以选择多个
<i>ENET_WUM_MAGI C_PACKET_FRAME</i>	使能接收到魔法帧的唤醒事件
<i>ENET_WUM_WAKE _UP_FRAME</i>	使能接收到唤醒帧的唤醒事件
<i>ENET_WUM_GLOB AL_UNICAST</i>	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：

```
/* 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 3-298. 函数 enet\_msc\_counters\_reset**

函数名称	enet_msc_counters_reset
函数原型	void enet_msc_counters_reset(void)
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 3-299. 函数 enet\_msc\_feature\_enable

函数名称	enet_msc_feature_enable
函数原型	void enet_msc_feature_enable(uint32_t feature);
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
<i>ENET_MSC_COUN TER_STOP_ROLLO VER</i>	计数器停止回转
<i>ENET_MSC_RESET _ON_READ</i>	读时复位
<i>ENET_MSC_COUN TERS_FREEZE</i>	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 3-300. 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	-
<b>输入参数{in}</b>	
<b>feature</b>	下列参数可以选择多个
<i>ENET_MSC_COUN TER_STOP_ROLLO VER</i>	计数器停止回转
<i>ENET_MSC_RESET _ON_READ</i>	读时复位
<i>ENET_MSC_COUN TERS_FREEZE</i>	MSC计数器冻结位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-301. 函数 enet\_msc\_counters\_preset\_config

函数名称	enet_msc_counters_preset_config
函数原型	void enet_msc_counters_preset_config(enet_msc_preset_enum mode);
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	下列参数可以选择多个
ENET_MSC_PRESET_NONE	关闭MSC计数器预设功能
ENET_MSC_PRESET_HALF	预设为最大值一半
ENET_MSC_PRESET_FULL	预设为最大值一半
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 3-302. 函数 enet\_msc\_counters\_get

函数名称	enet_msc_counters_get
函数原型	uint32_t enet_msc_counters_get(enet_msc_counter_enum counter);
功能描述	获取MAC相关统计计数器值

先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>counter</b>	MSC计数器, 下列参数仅可选择一个
<i>ENET_MSC_TX_SC CNT</i>	MSC 1次冲突后发送“好”帧的计数器
<i>ENET_MSC_TX_MS CCNT</i>	MSC 1次以上冲突后发送“好”帧的计数器
<i>ENET_MSC_TX_TG FCNT</i>	MSC发送“好”帧计数器
<i>ENET_MSC_RX_RF CECNT</i>	MSC CRC错误接收帧计数器
<i>ENET_MSC_RX_RF AECNT</i>	MSC对齐错误接收帧计数器
<i>ENET_MSC_RX_RG UFCNT</i>	MSC“好”单播帧接收帧计数器
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint32_t</b>	MSC计数器值

例如:

```
/* get MSC transmitted good frames after a single collision counter value*/
uint32_t raval;
raval = enet_msc_counters_get(ENET_MSC_TX_SCCNT);
```

#### 函数 **enet\_ptp\_feature\_enable**

函数**enet\_ptp\_feature\_enable**描述见下表:

表 3-303. 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	-
<b>输入参数{in}</b>	
<b>feature</b>	ENET模块PTP功能，下列参数可以选择多个
<i>ENET_RXTX_TIMES_TAMP</i>	发送/接收帧时间戳
<i>ENET_PTP_TIMESTAMP_INT</i>	时间戳中断触发
<i>ENET_ALL_RX_TIMESTAMP</i>	所有接收帧时间戳快照使能
<i>ENET_NONTYPE_FRAME_SNAPSHOT</i>	接收以太网帧时时间戳使能
<i>ENET_IPV6_FRAME_SNAPSHOT</i>	接收Ipv6帧时时间戳使能
<i>ENET_IPV4_FRAME_SNAPSHOT</i>	接收Ipv4帧时时间戳使能
<i>ENET_PTP_FRAME_USE_MACADDRESS_FILTER</i>	PTP帧MAC地址过滤使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-304. 函数 enet\_ptp\_feature\_disable**

函数名称	enet_ptp_feature_disable
函数原型	void enet_ptp_feature_disable(uint32_t feature);

<b>功能描述</b>	禁能PTP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>	ENET模块PTP功能, 下列参数可以选择多个
<i>ENET_RXTX_TIMESAMP</i>	发送/接收帧时间戳
<i>ENET_PTP_TIMESTAMP_INT</i>	时间戳中断触发
<i>ENET_ALL_RX_TIMESTAMP</i>	所有接收帧时间戳快照使能
<i>ENET_NONTYPE_FRAME_SNAPSHOT</i>	接收以太网帧时时间戳使能
<i>ENET_IPV6_FRAME_SNAPSHOT</i>	接收Ipv6帧时时间戳使能
<i>ENET_IPV4_FRAME_SNAPSHOT</i>	接收Ipv4帧时时间戳使能
<i>ENET_PTP_FRAME_USE_MACADDRESS_FILTER</i>	PTP帧MAC地址过滤使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-305. 函数 enet\_ptp\_timestamp\_function\_config**

<b>函数名称</b>	enet_ptp_timestamp_function_config
-------------	------------------------------------

<b>函数原型</b>	ErrStatus enet_ptp_timestamp_function_config(enet_ptp_function_enum func);
<b>功能描述</b>	配置PTP时间戳相关功能
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>func</b>	下列参数仅可选择一个
<i>ENET_CKNT_ORDINARY</i>	时间戳时钟节点类型为普通时钟
<i>ENET_CKNT_BOUNDARY</i>	时间戳时钟节点类型为边界时钟
<i>ENET_CKNT_END_TO_END</i>	时间戳时钟节点类型为端对端透明时钟
<i>ENET_CKNT_PEER_TO_PEER</i>	时间戳时钟节点类型为点对点透明时钟
<i>ENET_PTP_ADDEND_UPDATE</i>	加数寄存器更新
<i>ENET_PTP_SYSTIME_UPDATE</i>	时间戳更新
<i>ENET_PTP_SYSTIME_INIT</i>	时间戳初始化
<i>ENET_PTP_FINEMODE</i>	精调模式更新系统时间戳
<i>ENET_PTP_COARSMODE</i>	粗调模式更新系统时间戳
<i>ENET_SUBSECOND_DIGITAL_ROLLOVER</i>	十进制回转模式
<i>ENET_SUBSECOND_BINARY_ROLLOVER</i>	二进制回转模式
<i>ENET_SNOOPING_PTP_VERSION_2</i>	监听PTP帧版本2
<i>ENET_SNOOPING_</i>	监听PTP帧版本1

<i>PTP_VERSION_1</i>	
<i>ENET_EVENT_TYPE_MESSAGES_SNA_PSHOT</i>	只接收事件类型消息使能时间戳
<i>ENET_ALL_TYPE_MESSAGES_SNAP_SHOT</i>	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
<i>ENET_MASTER_NODE_MESSAGE_SNA_PSHOT</i>	主节点消息时间戳快照使能
<i>ENET_SLAVE_NODE_MESSAGE_SNAP_SHOT</i>	从节点消息时间戳快照使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

例如：

```
/* config addend register update function */

ErrStatus rval = ERROR;

rval = 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 3-306. 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	-
<b>输入参数{in}</b>	
<b>subsecond</b>	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 3-307. 函数 **enet\_ptp\_timestamp\_addend\_config**

函数名称	enet_ptp_timestamp_addend_config
函数原型	void enet_ptp_timestamp_addend_config(uint32_t add);
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}	
add	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 3-308. 函数 **enet\_ptp\_timestamp\_update\_config**

函数名称	enet_ptp_timestamp_update_config
------	----------------------------------

<b>函数原型</b>	void enet_ptp_timestamp_update_config(uint32_t sign, uint32_t second, uint32_t subsecond);
<b>功能描述</b>	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
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>sign</b>	时间戳更新正或负符号位，下列参数仅可选择一个
<i>ENET_PTP_ADD_TO_TIME</i>	更新值加到系统时间
<i>ENET_PTP_SUBSTRACT_FROM_TIME</i>	将系统时间减去更新值
<b>输入参数{in}</b>	
<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(0~0x7FFF 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 3-309. 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>	-

被调用函数	-
输入参数{in}	
second	目标时间秒值, 范围(0~0xFFFF FFFF)
输入参数{in}	
nanosecond	目标时间纳秒值, 范围(0~0xFFFF FFFF)
输出参数{out}	
-	-
返回值	
-	-

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 3-310. 函数 enet\_ptp\_system\_time\_get

函数名称	enet_ptp_system_time_get
函数原型	void enet_ptp_system_time_get(enet_ptp_systime_struct *systime_struct);
功能描述	获取PTP当前系统时间
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
systime_struct	PTP系统时间结构体指针, 结构体成员介绍请参考 <a href="#">结构体 enet_ptp_systime_struct</a>
返回值	
-	-

例如：

```

/* get the current system time */

enet_ptp_systime_struct systime;

enet_ptp_system_time_get(&systime);

```

### **函数 enet\_ptp\_pps\_output\_frequency\_config**

函数enet\_ptp\_pps\_output\_frequency\_config描述见下表：

**表 3-311. 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
<i>ENET_PPSOFC_1HZ</i>	PPS输出1Hz
<i>ENET_PPSOFC_2HZ</i>	PPS输出2Hz
<i>ENET_PPSOFC_4HZ</i>	PPS输出4Hz
<i>ENET_PPSOFC_8HZ</i>	PPS输出8Hz
<i>ENET_PPSOFC_16HZ</i>	PPS输出16Hz
<i>ENET_PPSOFC_32HZ</i>	PPS输出32Hz
<i>ENET_PPSOFC_64HZ</i>	PPS输出64Hz
<i>ENET_PPSOFC_128HZ</i>	PPS输出128Hz
<i>ENET_PPSOFC_256HZ</i>	PPS输出256Hz

<i>ENET_PPSOFC_512HZ</i>	PPS输出512Hz
<i>ENET_PPSOFC_1024HZ</i>	PPS输出1024Hz
<i>ENET_PPSOFC_2048HZ</i>	PPS输出2048Hz
<i>ENET_PPSOFC_4096HZ</i>	PPS输出4096Hz
<i>ENET_PPSOFC_8192HZ</i>	PPS输出8192Hz
<i>ENET_PPSOFC_16384HZ</i>	PPS输出16384Hz
<i>ENET_PPSOFC_32768HZ</i>	PPS输出32768Hz
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the PPS output frequency as 1Hz */
enet_ptp_pps_output_frequency_config(ENET_PPSOFC_1HZ);
```

#### 函数 **enet\_initpara\_reset**

函数enet\_initpara\_reset描述见下表：

**表 3-312. 函数 enet\_initpara\_reset**

函数名称	enet_initpara_reset
函数原型	void enet_initpara_reset(void);
功能描述	复位 ENET initpara struct, 需在enet_initpara_config()函数前调用
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	

-	
输出参数{out}	
-	
返回值	
-	

例如：

```
/* reset the ENET initpara struct */

enet_initpara_reset();
```

## 3.11. EXMC

外部存储器控制器EXMC，用来访问各种片外存储器。章节[3.11.1](#)描述了EXMC的寄存器列表，章节[3.11.2](#)对EXMC库函数进行说明。

### 3.11.1. 外设寄存器说明

EXMC寄存器列表如下表所示：

表 3-313. 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
EXMC_SDCTL	SDRAM控制寄存器
EXMC_SDTCFG	SDRAM时序寄存器
EXMC_SDCMD	SDRAM命令寄存器
EXMC_SDARI	SDRAM自动刷新间隔寄存器
EXMC_SDSTAT	SDRAM状态寄存器
EXMC_SDRSCTL	SDRAM读采样控制寄存器
EXMC_SINIT	SPI初始化寄存器
EXMC_SRCMD	SPI读命令寄存器
EXMC_SWCMD	SPI写命令寄存器

寄存器名称	寄存器描述
EXMC_SIDL	SPI ID低位寄存器
EXMC_SIDH	SPI ID高位寄存器

### 3.11.2. 外设库函数说明

EXMC库函数列表如下表所示：

**表 3-314. EXMC 库函数**

库函数名称	库函数描述
exmc_norsram_deinit	复位EXMC NOR/SRAM regionx
exmc_norsram_struct_para_init	初始化结构体exmc_norsram_parameter_struct
exmc_norsram_init	初始化EXMC NOR/SRAM regionx
exmc_norsram_enable	使能EXMC NOR/PSRAM regionx
exmc_norsram_disable	禁用EXMC NOR/PSRAM regionx
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 bankx
exmc_nand_enable	使能EXMC NAND bankx
exmc_nand_disable	禁用EXMC NAND bankx
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
exmc_pccard_enable	使能EXMC PC card bank
exmc_pccard_disable	禁用EXMC PC card bank
exmc_sdram_deinit	复位EXMC SDRAM devicex
exmc_sdram_struct_para_init	初始化结构体exmc_sdram_parameter_struct
exmc_sdram_init	初始化EXMC SDRAM device
exmc_sqipssram_deinit	复位EXMC SQPIPSRAM
exmc_sqipssram_struct_para_init	初始化结构体exmc_sqipssram_parameter_struct
exmc_sqipssram_init	初始化EXMC SQPIPSRAM
exmc_norsram_consecutive_clock_config	配置连续时钟产生条件
exmc_norsram_page_size_config	配置CRAM页大小
exmc_nand_ecc_config	配置EXMC NAND ECC功能
exmc_ecc_get	获取EXMC ECC值
exmc_sdram_readsampEnable	配置读采样功能
exmc_sdram_readsampConfig	配置读采样的延迟单元及周期
exmc_sdram_command_config	配置SDRAM存储器命令及相关参数
exmc_sdram_refresh_count_set	设置自动刷新间隔
exmc_sdram_autorefresh_number_set	设置连续自动刷新的个数
exmc_sdram_write_protection_config	设置写保护功能

库函数名称	库函数描述
exmc_sdram_bankstatus_get	获取SDRAM device0或device1状态
exmc_sqipsram_read_command_set	设置读命令码、读命令模式及读数据结束后的等待周期数
exmc_sqipsram_write_command_set	设置写命令码、写命令模式及写数据结束后的等待周期数
exmc_sqipsram_read_id_command_send	发送读SPI PSRAM ID命令
exmc_sqipsram_write_cmd_send	发送SPI PSRAM没有地址和数据阶段的特殊命令
exmc_sqipsram_low_id_get	获取SPI ID低位数据
exmc_sqipsram_high_id_get	获取SPI ID高位数据
exmc_sqipsram_send_command_state_get	获取EXMC发送写命令或读ID命令的状态
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 3-315. 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
syn_clk_division	同步时钟分频比
bus_latency	总线延迟
asyn_data_setuptime	数据建立时间
asyn_address_holdtime	地址保持时间
asyn_address_setuptime	地址建立时间

### 结构体 `exmc_norsram_parameter_struct`

表 3-316. 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

成员名称	功能描述
memory_write	使能或者禁用写操作
nwait_config	配置NWAIT信号
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
write_timing	未用扩展模式时，写时序参数

### 结构体 **exmc\_nand\_pccard\_timing\_parameter\_struct**

表 3-317. 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（保持命令的最长时间）
setuptime	地址信号的建立时间

### 结构体 **exmc\_nand\_parameter\_struct**

表 3-318. 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
attribute_space_timing	NAND flash属性空间时序配置

### 结构体 **exmc\_pccard\_parameter\_struct**

表 3-319. 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

成员名称	功能描述
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
attribute_space_timing	PC card属性空间时序配置
io_space_timing	PC card I/O空间时序配置

### 结构体 **exmc\_sdrdram\_timing\_parameter\_struct**

表 3-320. 结构体 **exmc\_sdrdram\_timing\_parameter\_struct**

成员名称	功能描述
row_to_column_delay	行至列的延迟
row_preadge_delay	行预充电延迟
write_recovery_delay	写恢复延迟
auto_refresh_delay	自动刷新延迟
row_address_select_delay	行地址选择延迟
exit_selfrefresh_delay	退出自刷新的延迟
load_mode_register_delay	加载模式寄存器延迟

### 结构体 **exmc\_sdrdram\_parameter\_struct**

表 3-321. 结构体 **exmc\_sdrdram\_parameter\_struct**

成员名称	功能描述
sdrdram_device	选择EXMC SDRAM device
pipeline_read_delay	流水线读数据延迟
burst_read_switch	突发读开关
sdclock_config	SDRAM时钟配置
write_protection	配置写保护功能
cas_latency	配置CAS延迟
internal_bank_number	内部Bank的个数
data_width	SDRAM数据总线宽度
row_address_width	行地址位宽
column_address_width	列地址位宽
timing	读写时序参数

### 结构体 exmc\_sdram\_command\_parameter\_struct

表 3-322. 结构体 exmc\_sdram\_command\_parameter\_struct

成员名称	功能描述
mode_register_content	指定SDRAM模式寄存器内容
auto_refresh_number	连续的自动刷新个数
bank_select	选择SDRAM device
command	指定发送到SDRAM上的命令

### 结构体 exmc\_sqipsram\_parameter\_struct

表 3-323. 结构体 exmc\_sqipsram\_parameter\_struct

成员名称	功能描述
sample_polarity	读数据时的采样极性
id_length	SPI PSRAM ID长度
address_bits	SPI PSRAM地址位数
command_bits	SPI PSRAM命令位数

### 函数 exmc\_norsram\_deinit

函数exmc\_norsram\_deinit描述见下表：

表 3-324. 函数 exmc\_norsram\_deinit

函数名称	exmc_norsram_deinit
函数原型	void exmc_norsram_deinit(uint32_t exmc_norsram_region);
功能描述	复位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：

```
/* 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 3-325. 函数 exmc\_norsram\_struct\_para\_init

函数名称	exmc_norsram_struct_para_init
函数原型	void exmc_norsram_struct_para_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);
功能描述	初始化结构体exmc_norsram_parameter_struct
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 3-316. 结构体 exmc_norsram_parameter_struct</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 3-326. 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 3-316. 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_DATALAT_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_setuptime = 5;

lcd_timing_init_struct.asyn_address_holdtime = 2;

lcd_timing_init_struct.asyn_address_setuptime = 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 3-327. 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_re gion	EXMC NOR/SRAM region
EXMC_BANK0_NO RSRAM_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 3-328. 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_re gion	EXMC NOR/SRAM region
EXMC_BANK0_NO RSRAM_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\_nand\_deinit**

函数**exmc\_nand\_deinit**描述见下表：

**表 3-329. 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}	
<b>exmc_nand_bank</b>	EXMC NAND bank
<b>EXMC_BANKx_NA ND</b>	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 3-330. 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}	
<b>exmc_nand_init_st ruct</b>	初始化结构体，结构体成员参考 <a href="#">错误!未找到引用源。表 3-318. 结构体 <u>exmc_nand_parameter_struct</u></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 3-331. 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_st ruct	初始化结构体，结构体成员参考 <a href="#">表 3-318. 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 3-332. 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}	
<b>exmc_nand_bank</b>	EXMC NAND bank
<b>EXMC_BANKx_NA ND</b>	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 3-333. 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_NA ND</i>	x=1,2
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable EXMC NAND bank1 */

exmc_nand_disable(EXMC_BANK1_NAND);
```

#### 函数 **exmc\_pccard\_deinit**

函数**exmc\_pccard\_deinit**描述见下表：

**表 3-334. 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	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-335. 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}	
<b>exmc_pccard_init_struct</b>	初始化结构体, 结构体成员参考 <a href="#">表 3-319. 结构体 exmc_pccard_parameter_struct</a>
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 3-336. 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}	
<b>exmc_pccard_init_struct</b>	初始化结构体, 结构体成员参考 <a href="#">表 3-319. 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 3-337. 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 3-338. 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\_sdrdram\_deinit

函数exmc\_sdrdram\_deinit描述见下表：

表 3-339. 函数 exmc\_sdrdram\_deinit

函数名称	exmc_sdrdram_deinit
函数原型	void exmc_sdrdram_deinit(uint32_t exmc_sdrdram_device);
功能描述	复位EXMC SDRAM device
先决条件	-
被调用函数	-
输入参数{in}	
exmc_sdrdram_devi ce	EXMC SDRAM device
EXMC_SDRAM_DE VICEx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize EXMC SDRAM device1 */

exmc_sdrdram_deinit(EXMC_SDRAM_DEVICE1);
```

### 函数 exmc\_sdrdram\_struct\_para\_init

函数exmc\_sdrdram\_struct\_para\_init描述见下表：

表 3-340. 函数 exmc\_sdrdram\_struct\_para\_init

函数名称	exmc_sdrdram_struct_para_init
函数原型	exmc_sdrdram_struct_para_init(exmc_sdrdram_parameter_struct* exmc_sdrdram_init_struct);
功能描述	初始化结构体exmc_sdrdram_parameter_struct

先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>exmc_sdrdram_init_struct</b>	初始化结构体, 结构体成员参考 <a href="#">表 3-321. 结构体 exmc_sdrdram_parameter_struct</a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* initialize the struct sdrdram_init_struct */

exmc_sdrdram_parameter_struct sdrdram_init_struct;
exmc_sdrdram_struct_para_init (&sdrdram_init_struct);
```

### 函数 **exmc\_sdrdram\_init**

函数exmc\_sdrdram\_init描述见下表:

**表 3-341. 函数 exmc\_sdrdram\_init**

函数名称	<b>exmc_sdrdram_init</b>
函数原型	void exmc_sdrdram_init(exmc_sdrdram_parameter_struct* exmc_sdrdram_init_struct);
功能描述	初始化EXMC SDRAM device
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>exmc_sdrdram_init_struct</b>	初始化结构体, 结构体成员参考 <a href="#">表 3-321. 结构体 exmc_sdrdram_parameter_struct</a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
exmc_sdrdram_parameter_struct          sdrdram_init_struct;
exmc_sdrdram_timing_parameter_struct   sdrdram_timing_init_struct;
/* EXMC configuration */
sdrdram_timing_init_struct.load_mode_register_delay = 2;
/* XSRD: min = 67ns */
```

```

sdram_timing_init_struct.exit_selfrefresh_delay = 7;

/* RASD: min=42ns , max=120k (ns) */

sdram_timing_init_struct.row_address_select_delay = 5;

/* ARFD: min=60ns */

sdram_timing_init_struct.auto_refresh_delay = 6;

/* WRD: min=1 Clock cycles +6ns */

sdram_timing_init_struct.write_recovery_delay = 2;

/* RPD: min=18ns */

sdram_timing_init_struct.row_precharge_delay = 2;

/* RCD: min=18ns */

sdram_timing_init_struct.row_to_column_delay = 2;

sdram_init_struct.sdram_device = sdram_device;

sdram_init_struct.column_address_width = EXMC_SDRAM_COW_ADDRESS_9;

sdram_init_struct.row_address_width = EXMC_SDRAM_ROW_ADDRESS_13;

sdram_init_struct.data_width = EXMC_SDRAM_DATABUS_WIDTH_16B;

sdram_init_struct.internal_bank_number = EXMC_SDRAM_4_INTER_BANK;

sdram_init_struct.cas_latency = EXMC_CAS_LATENCY_3_SDCLK;

sdram_init_struct.write_protection = DISABLE;

sdram_init_struct.sdclock_config = EXMC_SDCLK_PERIODS_2_HCLK;

sdram_init_struct.brust_read_switch = ENABLE;

sdram_init_struct.pipeline_read_delay = EXMC_PIPELINE_DELAY_1_HCLK;

sdram_init_struct.timing = &sdram_timing_init_struct;

/* EXMC SDRAM bank initialization */

exmc_sdram_init(&sdram_init_struct);

```

### 函数 **exmc\_sqipsram\_deinit**

函数**exmc\_sqipsram\_deinit**描述见下表：

**表 3-342. 函数 exmc\_sqipsram\_deinit**

函数名称	<b>exmc_sqipsram_deinit</b>
函数原型	void exmc_sqipsram_deinit(void);
功能描述	复位EXMC SQPIPSRAM

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 SQPIPSRAM */

exmc_sqpipsram_deinit(void);
```

### 函数 **exmc\_sqpipsram\_struct\_para\_init**

函数exmc\_sqpipsram\_struct\_para\_init描述见下表：

**表 3-343. 函数 exmc\_sqpipsram\_struct\_para\_init**

函数名称	exmc_sqpipsram_struct_para_init
函数原型	void exmc_sqpipsram_struct_para_init(exmc_sqpipsram_parameter_struct* exmc_sqpipsram_init_struct);
功能描述	初始化结构体exmc_sqpipsram_parameter_struct
先决条件	-
被调用函数	-
输入参数{in}	
exmc_sqpipsram_i nit_struct	初始化结构体，结构体成员参考 <a href="#">表 3-323. 结构体 exmc_sqpipsram_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the struct sqpipsram_init_struct */

exmc_sqpipsram_parameter_struct sqpipsram_init_struct;
exmc_sqpipsram_struct_para_init (&sqpipsram_init_struct);
```

### 函数 **exmc\_sqpipsram\_init**

函数exmc\_sqpipsram\_init描述见下表：

**表 3-344. 函数 exmc\_sqpipsram\_init**

函数名称	exmc_sqpipsram_init
------	---------------------

函数原型	void exmc_sqipsram_init(exmc_sqipsram_parameter_struct* exmc_sqipsram_init_struct);
功能描述	初始化EXMC SQPIPSRAM
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>exmc_sqipsram_i nit_struct</b>	初始化结构体, 结构体成员参考 <a href="#">错误!未找到引用源。表 3-323. 结构体 <u>exmc_sqipsram_parameter_struct</u></a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
exmc_sqipsram_parameter_struct sqipsram_init_struct;
/* EXMC configuration */
sqipsram_init_struct.sample_polarity = EXMC_SQPIPSRAM_SAMPLE_RISING_EDGE;
sqipsram_init_struct.id_length = EXMC_SQPIPSRAM_ID_LENGTH_64B;
sqipsram_init_struct.address_bits = EXMC_SQPIPSRAM_ADDR_LENGTH_24B;
sqipsram_init_struct.command_bits = EXMC_SQPIPSRAM_COMMAND_LENGTH_8B;
/* EXMC SDRAM bank initialization */
exmc_sqipsram_init(&sqipsram_init_struct);
```

### 函数 **exmc\_norsram\_consecutive\_clock\_config**

函数exmc\_norsram\_consecutive\_clock\_config描述见下表:

**表 3-345. 函数 exmc\_norsram\_consecutive\_clock\_config**

函数名称	exmc_norsram_consecutive_clock_config
函数原型	void exmc_norsram_consecutive_clock_config(uint32_t clock_mode);
功能描述	配置连续时钟
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>clock_mode</b>	连续时钟模式
<b>EXMC_CLOCK_SYN N_MODE</b>	EXMC_CLK只在同步模式产生
<b>EXMC_CLOCK_UN CONDITIONALLY</b>	EXMC_CLK无条件产生
<b>输出参数{out}</b>	

-	-
返回值	
-	-

例如：

```
/* configure consecutive clock */

exmc_norsram_consecutive_clock_config(EXMC_CLOCK_SYN_MODE);
```

#### 函数 **exmc\_norsram\_page\_size\_config**

函数exmc\_norsram\_page\_size\_config描述见下表：

**表 3-346. 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_AUTOSPLIT	页边界自动突发分割
EXMC_CRAM_PAGESIZE_128_BYT	页大小128字节
ES	
EXMC_CRAM_PAGESIZE_256_BYT	页大小256字节
S	
EXMC_CRAM_PAGESIZE_512_BYT	页大小512字节
S	
EXMC_CRAM_PAGESIZE_1024_BYT	页大小1024字节
ES	
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\_ecc\_config

函数exmc\_nand\_ecc\_config描述见下表:

**表 3-347. 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}	
exmc_nand_bank	EXMC NAND bank
EXMC_BANKx_NA ND	x=1,2
输入参数{in}	
newvalue	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 3-348. 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_NA ND	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\_sdrdram\_readsampEnable**

函数exmc\_sdrdram\_readsampEnable描述见下表：

**表 3-349. 函数 exmc\_sdrdram\_readsampEnable**

函数名称	exmc_sdrdram_readsampEnable
函数原型	void exmc_sdrdram_readsampEnable(ControlStatus newvalue);
功能描述	使能或禁用读采样功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>newvalue</b>	ENABLE或DISABLE
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable read sample */

exmc_sdrdram_readsampEnable(ENABLE);
```

#### 函数 **exmc\_sdrdram\_readsampConfig**

函数exmc\_sdrdram\_readsampConfig描述见下表：

**表 3-350. 函数 exmc\_sdrdram\_readsampConfig**

函数名称	exmc_sdrdram_readsampConfig
函数原型	void exmc_sdrdram_readsampConfig(uint32_t delay_cell, uint32_t extra_hclk);
功能描述	配置读数据的采样时钟的延迟单元及采样周期
先决条件	-
被调用函数	-
输入参数{in}	
<b>delay_cell</b>	读数据的采样时钟的延迟单元
<b>EXMC_SDRAM_X_DELAY_CELL</b>	x=0...15
输入参数{in}	
<b>extra_hclk</b>	读数据的采样周期

<i>EXMC_SDRAM_RE</i>	
<i>ADSAMPLE_x_EXT</i>	x=0,1
<i>RAHCLK</i>	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the delayed sample clock and sample cycle of read data */

exmc_sdrdram_readsampel_config(EXMC_SDRAM_1_DELAY_CELL,
EXMC_SDRAM_READSAMPLE_1_EXTRAHCLK);
```

### 函数 **exmc\_sdrdram\_command\_config**

函数**exmc\_sdrdram\_command\_config**描述见下表：

表 3-351. 函数 **exmc\_sdrdram\_command\_config**

函数名称	exmc_sdrdram_command_config
函数原型	void exmc_sdrdram_command_config(exmc_sdrdram_command_parameter_struct* exmc_sdrdram_command_init_struct);
功能描述	配置SDRAM存储器命令参数
先决条件	-
被调用函数	-
输入参数{in}	
<b>exmc_sdrdram_com mand_init_struct</b>	初始化结构体，结构体成员参考 <a href="#">表 3-322. 结构体 <b>exmc_sdrdram_command_parameter_struct</b></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the SDRAM memory command */

exmc_sdrdram_command_parameter_struct      sdrdram_command_init_struct;
sdrdram_command_init_struct.command = EXMC_SDRAM_CLOCK_ENABLE;
sdrdram_command_init_struct.bank_select = bank_select;
sdrdram_command_init_struct.auto_refresh_number
EXMC_SDRAM_AUTO_REFLESH_1_SDCLK;
sdrdram_command_init_struct.mode_register_content = 0;
```

```
exmc_sdran_command_config(&sdran_command_init_struct);
```

### 函数 exmc\_sdran\_refresh\_count\_set

函数exmc\_sdran\_refresh\_count\_set描述见下表:

**表 3-352. 函数 exmc\_sdran\_refresh\_count\_set**

函数名称	exmc_sdran_refresh_count_set
函数原型	void exmc_sdran_refresh_count_set(uint32_t exmc_count);
功能描述	配置自刷新间隔
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
exmc_count	两个连续的自动刷新命令之间间隔的存储器时钟周期单元 0x0000~0x1FFF
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* set the SDRAM auto-refresh rate counter */

exmc_sdran_refresh_count_set(761);
```

### 函数 exmc\_sdran\_autorefresh\_number\_set

函数exmc\_sdran\_autorefresh\_number\_set描述见下表:

**表 3-353. 函数 exmc\_sdran\_autorefresh\_number\_set**

函数名称	exmc_sdran_autorefresh_number_set
函数原型	void exmc_sdran_autorefresh_number_set(uint32_t exmc_number);
功能描述	配置连续自刷新个数
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
exmc_number	连续的自动刷新个数
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* set the number of successive auto-refresh command */
```

```
exmc_sdran_ autorefresh_number_set(10);
```

### 函数 exmc\_sdran\_write\_protection\_config

函数exmc\_sdran\_write\_protection\_config描述见下表：

**表 3-354. 函数 exmc\_sdran\_write\_protection\_config**

函数名称	exmc_sdran_write_protection_config
函数原型	void exmc_sdran_write_protection_config(uint32_t exmc_sdran_device, ControlStatus newvalue);
功能描述	使能或禁用写保护功能
先决条件	-
被调用函数	-
输入参数{in}	
exmc_sdran_devi ce	EXMC SDRAM device
EXMC_SDRAM_DE VICEx	x=0,1
输入参数{in}	
newvalue	ENABLE或DISABLE
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the write protection function */
exmc_sdran_write_protection_config(EXMC_SDRAM_DEVICE1, ENABLE);
```

### 函数 exmc\_sdran\_bankstatus\_get

函数exmc\_sdran\_bankstatus\_get描述见下表：

**表 3-355. 函数 exmc\_sdran\_bankstatus\_get**

函数名称	exmc_sdran_bankstatus_get
函数原型	uint32_t exmc_sdran_bankstatus_get(uint32_t exmc_sdran_device);
功能描述	获取SDRAM device0或device1状态
先决条件	-
被调用函数	-
输入参数{in}	
exmc_sdran_devi ce	EXMC SDRAM device
EXMC_SDRAM_DE VICEx	x=0,1

输出参数{out}	
-	-
返回值	
uint32_t	SDRAM deviceX状态

例如：

```
/* get the status of SDRAM device1 */
uint32_t status;
status = exmc_sdram_bankstatus_get (EXMC_SDRAM_DEVICE1);
```

#### 函数 **exmc\_sqipsram\_read\_command\_set**

函数exmc\_sqipsram\_read\_command\_set描述见下表：

表 3-356. 函数 **exmc\_sqipsram\_read\_command\_set**

函数名称	exmc_sqipsram_read_command_set
函数原型	void exmc_sqipsram_read_command_set(uint32_t read_command_mode,uint32_t read_wait_cycle,uint32_t read_command_code);
功能描述	设置读命令
先决条件	-
被调用函数	-
输入参数{in}	
<b>read_command_m ode</b>	SPI PSRAM读命令模式
<b>EXMC_SQPIPSRA M_READ_MODE_D ISABLE</b>	非SPI模式
<b>EXMC_SQPIPSRA M_READ_MODE_S PI</b>	SPI模式
<b>EXMC_SQPIPSRA M_READ_MODE_S QPI</b>	SQPI模式
<b>EXMC_SQPIPSRA M_READ_MODE_Q PI</b>	QPI模式
输入参数{in}	
<b>read_wait_cycle</b>	读数据时地址阶段结束后等待的周期数， 0..15
输入参数{in}	
<b>read_command_co de</b>	SPI读命令的命令码
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* set the SQPIPSRAM read command */
exmc_sqpipsram_read_command_set(EXMC_SQPIPSRAM_READ_MODE_SQPI,1,1);
```

#### 函数 **exmc\_sqpipsram\_write\_command\_set**

函数exmc\_sqpipsram\_write\_command\_set描述见下表：

表 3-357. 函数 **exmc\_sqpipsram\_write\_command\_set**

函数名称	exmc_sqpipsram_write_command_set
函数原型	void exmc_sqpipsram_write_command_set(uint32_t write_command_mode,uint32_t write_wait_cycle,uint32_t write_command_code);
功能描述	设置写命令
先决条件	-
被调用函数	-
输入参数{in}	
<b>write_command_mode</b>	SPI PSRAM写命令模式
<i>EXMC_SQPIPSRA_M_READ_MODE_D_ISABLE</i>	非SPI模式
<i>EXMC_SQPIPSRA_M_READ_MODE_SPI</i>	SPI模式
<i>EXMC_SQPIPSRA_M_READ_MODE_SQPI</i>	SQPI模式
<i>EXMC_SQPIPSRA_M_READ_MODE_QPI</i>	QPI模式
输入参数{in}	
<b>write_wait_cycle</b>	写数据时地址阶段结束后等待的周期数， 0..15
输入参数{in}	
<b>write_command_code</b>	SPI写命令的命令码
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* set the SQPIPSRAM write command */

exmc_sqpipsram_write_command_set(EXMC_SQPIPSRAM_WRITE_MODE_SQPI,1,1);
```

#### 函数 **exmc\_sqpipsram\_read\_id\_command\_send**

函数exmc\_sqpipsram\_read\_id\_command\_send描述见下表：

**表 3-358. 函数 exmc\_sqpipsram\_read\_id\_command\_send**

函数名称	exmc_sqpipsram_read_id_command_send
函数原型	void exmc_sqpipsram_read_id_command_send(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 SPI read ID command */

exmc_sqpipsram_read_id_command_send();
```

#### 函数 **exmc\_sqpipsram\_write\_cmd\_send**

函数exmc\_sqpipsram\_write\_cmd\_send描述见下表：

**表 3-359. 函数 exmc\_sqpipsram\_write\_cmd\_send**

函数名称	exmc_sqpipsram_write_cmd_send
函数原型	void exmc_sqpipsram_write_cmd_send(void);
功能描述	发送SPI PSRAM没有地址和数据阶段的特殊命令
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 SPI special command which does not have address and data phase */

exmc_sqpipsram_write_cmd_send();
```

### 函数 **exmc\_sqpipsram\_low\_id\_get**

函数exmc\_sqpipsram\_low\_id\_get描述见下表：

**表 3-360. 函数 exmc\_sqpipsram\_low\_id\_get**

函数名称	exmc_sqpipsram_low_id_get
函数原型	uint32_t exmc_sqpipsram_low_id_get(void);
功能描述	获取SPI ID低位数据
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	ID低位数据

例如：

```
/* the ID low data */

uint32_t id;

id = exmc_sqpipsram_low_id_get();
```

### 函数 **exmc\_sqpipsram\_high\_id\_get**

函数exmc\_sqpipsram\_high\_id\_get描述见下表：

**表 3-361. 函数 exmc\_sqpipsram\_low\_id\_get**

函数名称	exmc_sqpipsram_high_id_get
函数原型	uint32_t exmc_sqpipsram_high_id_get(void);
功能描述	获取SPI ID高位数据
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	ID高位数据

例如：

```
/* the ID high data */

uint32_t id;

id = exmc_sqipsram_high_id_get();
```

### 函数 **exmc\_sqipsram\_send\_command\_state\_get**

函数exmc\_sqipsram\_send\_command\_state\_get描述见下表：

**表 3-362. 函数 exmc\_sqipsram\_send\_command\_state\_get**

函数名称	exmc_sqipsram_send_command_state_get
函数原型	FlagStatus exmc_sqipsram_send_command_state_get(uint32_t send_command_flag);
功能描述	获取EXMC发送写命令或读ID命令的状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>send_command_flag</b>	发送命令的状态
<b>EXMC_SEND_COMMAND_FLAG_RDID</b>	发送了读SPI PSRAM ID命令的状态
<b>EXMC_SEND_COMMAND_FLAG_SC</b>	发送了读SPI PSRAM没有地址和数据阶段的特殊命令的状态
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如：

```
/* check EXMC_SEND_COMMAND_FLAG_SC is set or not*/

if(RESET != exmc_sqipsram_send_command_state_get(EXMC_SEND_COMMAND_FLAG_SC));
```

### 函数 **exmc\_interrupt\_enable**

函数exmc\_interrupt\_enable描述见下表：

**表 3-363. 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	-
<b>输入参数{in}</b>	
<b>exmc_bank</b>	EXMC外设
<i>EXMC_BANK1_NA ND</i>	NAND bank1
<i>EXMC_BANK2_NA ND</i>	NAND bank2
<i>EXMC_BANK3_PC CARD</i>	PC Card bank
<i>EXMC_SDRAM_DE VICE0</i>	SDRAM device0
<i>EXMC_SDRAM_DE VICE1</i>	SDRAM device1
<b>输入参数{in}</b>	
<b>interrupt</b>	EXMC中断状态
<i>EXMC_NAND_PCC ARD_INT_FLAG_RI SE</i>	上升沿中断
<i>EXMC_NAND_PCC ARD_INT_FLAG_L EVEL</i>	高电平中断
<i>EXMC_NAND_PCC ARD_INT_FLAG_F ALL</i>	下降沿中断
<i>EXMC_SDRAM_IN T_FLAG_REFRESH</i>	刷新错误中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable EXMC interrupt*/

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 3-364. 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

<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>exmc_bank</b>	EXMC外设
<i>EXMC_BANK1_NA ND</i>	NAND bank1
<i>EXMC_BANK2_NA ND</i>	NAND bank2
<i>EXMC_BANK3_PC CARD</i>	PC Card bank
<i>EXMC_SDRAM_DE VICE0</i>	SDRAM device0
<i>EXMC_SDRAM_DE VICE1</i>	SDRAM device1
<b>输入参数{in}</b>	
<b>interrupt</b>	EXMC中断状态
<i>EXMC_NAND_PCC ARD_INT_FLAG_RI SE</i>	上升沿中断
<i>EXMC_NAND_PCC ARD_INT_FLAG_L EVEL</i>	高电平中断
<i>EXMC_NAND_PCC ARD_INT_FLAG_F ALL</i>	下降沿中断
<i>EXMC_SDRAM_IN T_FLAG_REFRESH</i>	刷新错误中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable EXMC interrupt*/
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 3-365. 函数 exmc\_flag\_get**

<b>函数名称</b>	<b>exmc_flag_get</b>
-------------	----------------------

<b>函数原型</b>	FlagStatus exmc_flag_get(uint32_t exmc_bank,uint32_t flag);
<b>功能描述</b>	获取EXMC状态
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>exmc_bank</b>	EXMC外设
<i>EXMC_BANK1_NA ND</i>	NAND bank1
<i>EXMC_BANK2_NA ND</i>	NAND bank2
<i>EXMC_BANK3_PC CARD</i>	PC Card bank
<i>EXMC_SDRAM_DE VICE0</i>	SDRAM device0
<i>EXMC_SDRAM_DE VICE1</i>	SDRAM device1
<b>输入参数{in}</b>	
<b>flag</b>	EXMC标志状态
<i>EXMC_NAND_PCC ARD_FLAG_RISE</i>	上升沿状态
<i>EXMC_NAND_PCC ARD_FLAG_LEVEL</i>	高电平状态
<i>EXMC_NAND_PCC ARD_FLAG_FALL</i>	下降沿状态
<i>EXMC_SDRAM_FL AG_REFRESH</i>	刷新错误状态
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET或RESET

例如：

```
/* check EXMC_NAND_PCCARD_FLAG_RISE 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 3-366. 函数 **exmc\_flag\_clear**

<b>函数名称</b>	exmc_flag_clear
<b>函数原型</b>	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	-
<b>输入参数{in}</b>	
<b>exmc_bank</b>	EXMC外设
<i>EXMC_BANK1_NA ND</i>	NAND bank1
<i>EXMC_BANK2_NA ND</i>	NAND bank2
<i>EXMC_BANK3_PC CARD</i>	PC Card bank
<i>EXMC_SDRAM_DE VICE0</i>	SDRAM device0
<i>EXMC_SDRAM_DE VICE1</i>	SDRAM device1
<b>输入参数{in}</b>	
<b>flag</b>	EXMC标志状态
<i>EXMC_NAND_PCC ARD_FLAG_RISE</i>	上升沿状态
<i>EXMC_NAND_PCC ARD_FLAG_LEVEL</i>	高电平状态
<i>EXMC_NAND_PCC ARD_FLAG_FALL</i>	下降沿状态
<i>EXMC_SDRAM_FL AG_REFRESH</i>	刷新错误状态
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* clear EXMC flag 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 3-367. 函数 **exmc\_interrupt\_flag\_get**

函数名称	<b>exmc_interrupt_flag_get</b>
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	-
<b>输入参数{in}</b>	
<b>exmc_bank</b>	EXMC外设
<i>EXMC_BANK1_NA ND</i>	NAND bank1
<i>EXMC_BANK2_NA ND</i>	NAND bank2
<i>EXMC_BANK3_PC CARD</i>	PC Card bank
<i>EXMC_SDRAM_DE VICE0</i>	SDRAM device0
<i>EXMC_SDRAM_DE VICE1</i>	SDRAM device1
<b>输入参数{in}</b>	
<b>interrupt</b>	EXMC中断状态
<i>EXMC_NAND_PCC ARD_INT_FLAG_RI SE</i>	上升沿中断
<i>EXMC_NAND_PCC ARD_INT_FLAG_L EVEL</i>	高电平中断
<i>EXMC_NAND_PCC ARD_INT_FLAG_F ALL</i>	下降沿中断
<i>EXMC_SDRAM_IN T_FLAG_REFRESH</i>	刷新错误中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET或RESET

例如：

```
/* check EXMC_NAND_PCCARD_INT_FLAG_RISE 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 3-368. 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);

<b>功能描述</b>	清除EXMC中断状态
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>exmc_bank</b>	EXMC外设
<i>EXMC_BANK1_NA ND</i>	NAND bank1
<i>EXMC_BANK2_NA ND</i>	NAND bank2
<i>EXMC_BANK3_PC CARD</i>	PC Card bank
<i>EXMC_SDRAM_DE VICE0</i>	SDRAM device0
<i>EXMC_SDRAM_DE VICE1</i>	SDRAM device1
<b>输入参数{in}</b>	
<b>interrupt</b>	EXMC中断状态
<i>EXMC_NAND_PCC ARD_INT_FLAG_RI SE</i>	上升沿中断
<i>EXMC_NAND_PCC ARD_INT_FLAG_L EVEL</i>	高电平中断
<i>EXMC_NAND_PCC ARD_INT_FLAG_F ALL</i>	下降沿中断
<i>EXMC_SDRAM_IN T_FLAG_REFRESH</i>	刷新错误中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* clear EXMC interrupt flag */

exmc_interrupt_flag_clear(EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

### 3.12. EXTI

EXTI是MCU中的中断/事件控制器，包括23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[3.12.1](#)描述了EXTI的寄存器列表，章节[3.12.2](#)对EXTI库函数进

行说明。

### 3.12.1. 外设寄存器说明

EXTI寄存器列表如下表所示:

**表 3-369. 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

### 3.12.2. 外设库函数说明

EXTI库函数列表如下表所示:

**表 3-370. 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软件中断使能
exti_software_interrupt_disable	EXTI线x软件中断禁能
exti_flag_get	获取EXTI线x标志位
exti_flag_clear	清除EXTI线x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

### 函数 exti\_deinit

函数exti\_deinit描述见下表:

**表 3-371. 函数 exti\_deinit**

函数名称	exti_deinit
函数原形	void exti_deinit(void);
功能描述	复位EXTI，将EXTI的所有寄存器恢复成初始值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

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

### 函数 exti\_init

函数exti\_init描述见下表:

**表 3-372. 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

<i>EXTI_x</i>	x=0,1,2..22
<b>输入参数{in}</b>	
<b>mode</b>	EXTI模式
<i>EXTI_INTERRUPT</i>	中断模式
<i>EXTI_EVENT</i>	事件模式
<b>输入参数{in}</b>	
<b>trig_type</b>	触发类型
<i>EXTI_TRIG_RISING</i>	上升沿触发
<i>EXTI_TRIG_FALLING</i>	下降沿触发
<i>EXTI_TRIG_BOTH</i>	上升沿和下降沿均触发
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-373. 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	-
<b>输入参数{in}</b>	
<b>linex</b>	EXTI线x

<i>EXTI_x</i>	x=0,1,2..22
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 3-374. 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}	
<i>linex</i>	EXTI线x
<i>EXTI_x</i>	x=0,1,2..22
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 3-375. 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
EXTI_x	x=0,1,2..22
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 3-376. 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
EXTI_x	x=0,1,2..22
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 3-377. 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软件中断使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..22
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 3-378. 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软件中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..22
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 3-379. 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标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..22
输出参数{out}	
-	-
返回值	

<b>FlagStatus</b>	SET或RESET
-------------------	-----------

例如：

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

FlagStatus state = exti_flag_get(EXTI_0);
```

### 函数 **exti\_flag\_clear**

函数**exti\_flag\_clear**描述见下表：

**表 3-380. 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标志位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
linex	EXTI线x
EXTI_x	x=0,1,2..22
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-381. 函数 exti\_interrupt\_flag\_get**

函数名称	exti_interrupt_flag_get
函数原形	FlagStatus exti_interrupt_flag_get(exti_line_enum linex);
功能描述	获取EXTI线x中断标志位

先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..22
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

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

FlagStatus state = exti_interrupt_flag_get(EXTI_0);
```

#### 函数 exti\_interrupt\_flag\_clear

函数exti\_interrupt\_flag\_clear描述见下表：

表 3-382. 函数 exti\_interrupt\_flag\_clear

函数名称	exti_interrupt_flag_clear
函数原形	void exti_interrupt_flag_clear(exti_line_enum linex);
功能描述	清除EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..22
输出参数{out}	
-	-
返回值	
-	-

例如：

---

```

/* clear EXTI line 0 interrupt flag status */

exti_interrupt_flag_clear(EXTI_0);

```

## 3.13. 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[3.13.1](#)描述了FMC的寄存器列表，章节[3.13.2](#)对FMC库函数进行说明。

### 3.13.1. 外设寄存器说明

FMC寄存器列表如下：

**表 3-383. 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_OBCTL0	选项字节控制寄存器0
FMC_OBCTL1	选项字节控制寄存器1
FMC_PECFG	页擦除配置寄存器
FMC_PEKEY	页擦除功能解锁寄存器
FMC_WSEN	等待状态使能寄存器
FMC_PID	产品ID寄存器

### 3.13.2. 外设库函数说明

FMC固件库函数列举如下表：

**表 3-384. FMC 固件库函数**

函数名称	函数描述
fmc_wscnt_set	设置FMC等待状态计数值
fmc_unlock	解锁FMC主编程块操作
fmc_lock	锁定FMC主编程块操作
fmc_page_erase	FMC页擦除
fmc_sector_erase	FMC扇区擦除
fmc_mass_erase	FMC全片擦除
fmc_bank0_erase	FMC bank0全片擦除
fmc_bank1_erase	FMC bank1全片擦除
fmc_word_program	在相应地址全字编程
fmc_halfword_program	在相应地址半字编程

函数名称	函数描述
fmc_byte_program	在相应地址字节编程
ob_unlock	解锁选项字节操作
ob_lock	锁定选项字节操作
ob_start	发送更新选项字节指令
ob_erase	擦除选项字节并恢复成出厂值
ob_write_protection_enable	使能写保护
ob_write_protection_disable	除能写保护
ob_drp_enable	使能写保护与D-BUS读保护
ob_drp_disable	除能写保护与D-BUS读保护
ob_security_protection_config	配置安全保护等级
ob_user_write	写用户选项字节
ob_user_bor_threshold	配置选项字节BOR复位阈值
ob_boot_mode_config	配置选项字节启动块值
ob_user_get	获取用户选项字节
ob_write_protection0_get	获取bank0写保护字节
ob_write_protection1_get	获取bank1写保护字节
ob_drp0_get	获取bank0 D-BUS读保护字节
ob_drp1_get	获取bank1 D-BUS读保护字节
ob_spc_get	获取安全保护等级值
ob_user_bor_threshold_get	获取选项字节BOR复位阈值
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	除能FMC中断
fmc_interrupt_flag_get	检查中断标志位是否置位
fmc_interrupt_flag_clear	清除FMC中断标志
fmc_state_get	获取FMC状态
fmc_ready_wait	检查FMC是否准备好

### 枚举 fmc\_state\_enum

表 3-385. 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_RDDERR	D-BUS读错误
FMC_PGSERR	编程顺序错误
FMC_PGMERR	编程类型不匹配错误
FMC_WPERR	擦/写保护错误
FMC_OPERR	闪存操作错误
FMC_TOERR	超时错误

### 函数 fmc\_wscnt\_set

函数fmc\_wscnt\_set描述见下表:

**表 3-386. 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	-
<b>输入参数{in}</b>	
wscnt	等待状态计数值
WS_WSCNT_x	增加X (X=0~15) 个等待状态
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

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

fmc_wscnt_set (WS_WSCNT_1);
```

### 函数 fmc\_unlock

函数fmc\_unlock描述见下表:

**表 3-387. 函数 fmc\_unlock**

函数名称	fmc_unlock
函数原型	void fmc_unlock(void)
功能描述	解锁FMC主编程块操作
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

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

fmc_unlock( );
```

### 函数 fmc\_lock

函数fmc\_lock描述见下表：

**表 3-388. 函数 fmc\_lock**

函数名称	fmc_lock
函数原型	void fmc_lock(void)
功能描述	锁定FMC主编程块操作
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

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

fmc_lock();
```

### 函数 fmc\_page\_erase

函数fmc\_page\_erase描述见下表：

**表 3-389. 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_addr);
功能描述	FMC扇区擦除
先决条件	fmc_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
page_addr	页地址
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，详情参考 <a href="#">枚举fmc_state_enum</a>

例如：

```
/* erase page */

fmc_unlock();

fmc_state_enum state = FMC_READY;
```

```
state = fmc_page_erase(0x08008000);
```

### 函数 **fmc\_sector\_erase**

函数fmc\_sector\_erase描述见下表:

**表 3-390. 函数 fmc\_sector\_erase**

函数名称	fmc_sector_erase
函数原型	fmc_state_enum fmc_sector_erase(uint32_t fmc_sector);
功能描述	FMC扇区擦除
先决条件	fmc_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
fmc_sector	扇区编号
CTL_SECTOR_NUMB ER_x	擦除第 X (X=0~27)号扇区
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 详情参考 <a href="#">枚举fmc_state_enum</a>

例如:

```
/* erase number 15 sector */
fmc_unlock();
fmc_state_enum state = FMC_READY;
state = fmc_sector_erase(CTL_SECTOR_NUMBER_15);
```

### 函数 **fmc\_mass\_erase**

函数fmc\_mass\_erase描述见下表:

**表 3-391. 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	FMC全片擦除
先决条件	fmc_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 详情参考 <a href="#">枚举fmc_state_enum</a>

例如：

```

/* erase the whole chip */

fmc_unlock( );

fmc_state_enum state = FMC_READY;

state = fmc_mass_erase();

```

### **函数 fmc\_bank0\_erase**

函数fmc\_bank0\_erase描述见下表：

**表 3-392. 函数 fmc\_bank0\_erase**

函数名称	fmc_bank0_erase
函数原型	fmc_state_enum fmc_bank0_erase(void);
功能描述	FMC bank0全片擦除
先决条件	fmc_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<b>fmc_state_enum</b>	FMC状态，详情参考 <a href="#">枚举fmc_state_enum</a>

例如：

```

/* erase bank0 */

fmc_unlock( );

fmc_state_enum state = FMC_READY;

state = fmc_bank0_erase( );

```

### **函数 fmc\_bank1\_erase**

函数fmc\_bank1\_erase描述见下表：

**表 3-393. 函数 fmc\_bank1\_erase**

函数名称	fmc_bank1_erase
函数原型	fmc_state_enum fmc_bank1_erase(void);
功能描述	FMC bank1全片擦除
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
<b>fmc_state_enum</b>	FMC状态, 详情参考 <a href="#">枚举fmc_state_enum</a>

例如:

```
/* erase bank1 */

fmc_unlock( );

fmc_state_enum state = FMC_READY;

state = fmc_bank1_erase( );
```

### 函数 **fmc\_word\_program**

函数fmc\_word\_program描述见下表:

**表 3-394. 函数 fmc\_word\_program**

函数名称	fmc_word_program
函数原型	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
功能描述	在相应地址全字编程
先决条件	fmc_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
<b>address</b>	编程地址
<b>data</b>	待编程数据 (0x00000000 – 0xFFFFFFFF)
输出参数{out}	
-	-
返回值	
<b>fmc_state_enum</b>	FMC状态, 详情参考 <a href="#">枚举fmc_state_enum</a>

例如:

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

fmc_unlock( );

fmc_state_enum state = FMC_READY;

state = fmc_word_program(0x08004000, 0x12345678);
```

### 函数 **fmc\_halfword\_program**

函数fmc\_halfword\_program描述见下表:

**表 3-395. 函数 fmc\_halfword\_program**

函数名称	fmc_halfword_program
------	----------------------

<b>函数原型</b>	fmc_state_enum fmc_halfword_program(uint32_t address, uint16_t data);
<b>功能描述</b>	在相应地址半字编程
<b>先决条件</b>	fmc_unlock
<b>被调用函数</b>	fmc_ready_wait
<b>输入参数{in}</b>	
<b>address</b>	编程地址
<b>data</b>	待编程数据 (0x0000 – 0xFFFF)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>fmc_state_enum</b>	FMC状态, 详情参考 <a href="#">枚举fmc_state_enum</a>

例如:

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

fmc_unlock( );

fmc_state_enum state = FMC_READY;

state = fmc_halfword_program (0x08004000, 0x1234);
```

### 函数 fmc\_byte\_program

函数fmc\_byte\_program描述见下表:

**表 3-396. 函数 fmc\_byte\_program**

<b>函数名称</b>	fmc_byte_program
<b>函数原型</b>	fmc_state_enum fmc_byte_program(uint32_t address, uint8_t data);
<b>功能描述</b>	在相应地址字节编程
<b>先决条件</b>	fmc_unlock
<b>被调用函数</b>	fmc_ready_wait
<b>输入参数{in}</b>	
<b>address</b>	编程地址
<b>data</b>	待编程数据 (0x00 – 0xFF)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>fmc_state_enum</b>	FMC状态, 详情参考 <a href="#">枚举fmc_state_enum</a>

例如:

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

fmc_unlock( );

fmc_state_enum state = FMC_READY;

state = fmc_byte_program (0x08004000, 0x12);
```

### 函数 **ob\_unlock**

函数**ob\_unlock**描述见下表:

**表 3-397. 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 option byte operation */

ob_unlock( );
```

### 函数 **ob\_lock**

函数**ob\_lock**描述见下表:

**表 3-398. 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	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* lock the option byte operation */

ob_lock( );
```

### 函数 **ob\_start**

函数**ob\_start**描述见下表:

**表 3-399. 函数 ob\_start**

函数名称	ob_start
函数原型	void ob_start(void);
功能描述	发送更新选项字节指令
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable write protection and then send the start command */

ob_unlock( );

ob_write_protection_enable (OB_WP7);

ob_start();
```

### 函数 **ob\_erase**

函数**ob\_erase**描述见下表:

**表 3-400. 函数 ob\_erase**

函数名称	ob_erase
函数原型	void ob_erase(void);
功能描述	擦除选项字节并恢复成出厂值
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* erase and reset option byte */
```

```

ob_unlock( );
ob_erase( );
ob_start( );
ob_lock( );

```

### 函数 **ob\_write\_protection\_enable**

函数ob\_write\_protection\_enable描述见下表:

**表 3-401. 函数 ob\_write\_protection\_enable**

函数名称	ob_write_protection_enable
函数原型	ErrStatus ob_write_protection_enable(uint32_t ob_wp);
功能描述	使能相应扇区的写保护
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
ob_wp	写保护扇区编号
OB_WPx	编号为 x (x = 0...22) 的扇区
OB_WP_23_27	扇区23、24、25、26、27
OB_WP_ALL	全片写保护
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS 或 ERROR

例如:

```

/* enable write protection of sector 7 */

ErrStatus temp = ERROR;

ob_unlock( );

temp = ob_write_protection_enable (OB_WP7);

ob_start();

ob_lock( );

```

### 函数 **ob\_write\_protection\_disable**

函数ob\_write\_protection\_disable描述见下表:

**表 3-402. 函数 ob\_write\_protection\_disable**

函数名称	ob_write_protection_disable
函数原型	ErrStatus ob_write_protection_disable(uint32_t ob_wp);

<b>功能描述</b>	除能相应扇区的写保护
<b>先决条件</b>	ob_unlock
<b>被调用函数</b>	fmc_ready_wait
<b>输入参数{in}</b>	
<b>ob_wp</b>	写保护扇区编号
<b>OB_WP<sub>x</sub></b>	编号为 $x$ ( $x = 0 \dots 22$ ) 的扇区
<b>OB_WP_23_27</b>	扇区23、24、25、26、27
<b>OB_WP_ALL</b>	全片写保护
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>ErrStatus</b>	SUCCESS 或 ERROR

例如：

```
/* disable write protection of sector 7 */

ErrStatus temp = ERROR;

ob_unlock( );

temp = ob_write_protection_disable (OB_WP7);

ob_start();

ob_lock( );
```

### 函数 **ob\_drp\_enable**

函数**ob\_drp\_enable**描述见下表：

**表 3-403. 函数 ob\_drp\_enable**

<b>函数名称</b>	ob_drp_enable
<b>函数原型</b>	void ob_drp_enable(uint32_t ob_drp);
<b>功能描述</b>	使能对应扇区的写保护与D-BUS读保护
<b>先决条件</b>	ob_unlock
<b>被调用函数</b>	fmc_ready_wait
<b>输入参数{in}</b>	
<b>ob_drp</b>	写保护与D-BUS读保护扇区编号
<b>OB_DRPx</b>	编号为 $x$ ( $x = 0 \dots 22$ ) 的扇区
<b>OB_DRP_23_27</b>	扇区23、24、25、26、27

<b>OB_DRP_ALL</b>	全片写保护与D-BUS读保护
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable write protection and D-BUS read protectio of sector 7 */

ob_unlock( );
ob_drp_enable (OB_DRP7);
ob_start( );
ob_lock( );
```

#### 函数 **ob\_drp\_disable**

函数ob\_drp\_disable描述见下表：

**表 3-404. 函数 ob\_drp\_disable**

<b>函数名称</b>	ob_drp_disable
<b>函数原型</b>	void ob_drp_disable(uint32_t ob_drp);
<b>功能描述</b>	除能对应扇区的写保护与D-BUS读保护
<b>先决条件</b>	ob_unlock
<b>被调用函数</b>	fmc_ready_wait
<b>输入参数{in}</b>	
<b>ob_drp</b>	写保护与D-BUS读保护扇区编号
<b>OB_DRPx</b>	编号为 x (x = 0...22) 的扇区
<b>OB_DRP_23_27</b>	扇区23、24、25、26、27
<b>OB_DRP_ALL</b>	全片写保护与D-BUS读保护
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable write protection and D-BUS read protectio of sector 7 */

ob_unlock( );
ob_drp_disable (OB_DRP7);
ob_start( );
ob_lock( );
```

### 函数 **ob\_security\_protection\_config**

函数**ob\_security\_protection\_config**描述见下表：

**表 3-405. 函数 ob\_security\_protection\_config**

函数名称	ob_security_protection_config	
函数原型	void ob_security_protection_config(uint8_t ob_spc);	
功能描述	配置安全保护等级	
先决条件	ob_unlock	
被调用函数	fmc_ready_wait	
<b>输入参数{in}</b>		
<b>ob_spc</b>	安全保护级别	
<i>FMC_NSPC</i>	无安全保护	
<i>FMC_LSPC</i>	低安全保护	
<i>FMC_HSPC</i>	高安全保护	
<b>输出参数{out}</b>		
-	-	
<b>返回值</b>		
-	-	

例如：

```
/* config FMC as low security protection */

ob_unlock( );

ob_security_protection_config (FMC_LSPC);

ob_start( );

ob_lock( );
```

### 函数 **ob\_user\_write**

函数**ob\_user\_write**描述见下表：

**表 3-406. 函数 ob\_user\_write**

函数名称	ob_user_write	
函数原型	void ob_user_write(uint32_t ob_fwdgt, uint32_t ob_deepsleep, uint32_t ob_standby);	
功能描述	写用户选项字节	
先决条件	ob_unlock	
被调用函数	fmc_ready_wait	
<b>输入参数{in}</b>		
<b>ob_fwdgt</b>	看门狗选项	
<i>OB_FWDGT_SW</i>	软件自由看门狗	
<i>OB_FWDGT_HW</i>	硬件自由看门狗	

输入参数{in}	
<b>ob_deepsleep</b>	深度睡眠模式复位选项
<b>OB_DEEPSLEEP_NRST</b>	进入深度睡眠模式不产生复位
<b>OB_DEEPSLEEP_RST</b>	进入深度睡眠模式产生复位
输入参数{in}	
<b>ob_stby</b>	待机模式复位选项
<b>OB_STDBY_NRST</b>	进入待机模式不产生复位
<b>OB_STDBY_RST</b>	进入待机模式产生复位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config the USER byte of option byte */

ob_unlock();

ob_user_write(OB_FWDGT_SW, OB_DEEPSLEEP_NRST, OB_STDBY_NRST);

ob_start();

ob_lock();
```

### 函数 **ob\_user\_bor\_threshold**

函数**ob\_user\_bor\_threshold**描述见下表：

**表 3-407. 函数 ob\_user\_bor\_threshold**

函数名称	ob_user_bor_threshold
函数原型	void ob_user_bor_threshold(uint32_t ob_bor_th);
功能描述	配置选项字节BOR复位阈值
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
<b>ob_bor_th</b>	选项字节BOR阈值
<b>OB_BOR_TH_VAL UE3</b>	BOR阈值3
<b>OB_BOR_TH_VAL UE2</b>	BOR阈值2
<b>OB_BOR_TH_VAL UE1</b>	BOR阈值1
<b>OB_BOR_TH_OFF</b>	BOR关闭

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config the BOR threshold value as OB_BOR_TH_OFF */

ob_unlock( );

ob_user_bor_threshold(OB_BOR_TH_OFF);

ob_start( );

ob_lock( );
```

### 函数 **ob\_boot\_mode\_config**

函数ob\_boot\_mode\_config描述见下表：

**表 3-408. 函数 ob\_boot\_mode\_config**

函数名称	ob_boot_mode_config
函数原型	void ob_boot_mode_config(uint32_t boot_mode);
功能描述	配置选项字节启动块值
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
<b>boot_mode</b>	启动选项
<i>OB_BB_DISABLE</i>	从bank0启动
<i>OB_BB_ENABLE</i>	从bank1启动，若bank1为空则从bank0启动
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config boot from bank0 */

ob_unlock( );

ob_boot_mode_config(OB_BB_DISABLE);

ob_start( );

ob_lock( );
```

### 函数 **ob\_user\_get**

函数**ob\_user\_get**描述见下表：

**表 3-409. 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	获取选项字节用户字节值	
先决条件	-	
被调用函数	-	
输入参数{in}		
-	-	
输出参数{out}		
-	-	
返回值		
<b>uint8_t</b>	选项字节中用户字节的深度睡眠复位值、待机复位值与看门狗设置值	

例如：

```
/* get the the FMC user option byte values */

uint8_t ob_user_values = 0;

ob_user_values = ob_user_get( );
```

### 函数 **ob\_write\_protection0\_get**

函数**ob\_write\_protection0\_get**描述见下表：

**表 3-410. 函数 ob\_write\_protection0\_get**

函数名称	ob_write_protection0_get	
函数原型	uint16_t ob_write_protection0_get(void);	
功能描述	获取bank0写保护字节	
先决条件	-	
被调用函数	-	
输入参数{in}		
-	-	
输出参数{out}		
-	-	
返回值		
<b>uint16_t</b>	bank0写保护字节值	

例如：

```
/* get the FMC write protection option byte value */

uint16_t bank0_protection_value = 0;
```

```
bank0_protection_value = ob_write_protection0_get( );
```

### 函数 **ob\_write\_protection1\_get**

函数ob\_write\_protection1\_get描述见下表:

**表 3-411. 函数 ob\_write\_protection1\_get**

函数名称	ob_write_protection1_get
函数原型	uint16_t ob_write_protection1_get(void);
功能描述	获取bank1写保护字节
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	bank1写保护字节值

例如:

```
/* get the FMC write protection option byte value */

uint16_t bank1_protection_value = 0;

bank1_protection_value = ob_write_protection1_get( );
```

### 函数 **ob\_drp0\_get**

函数ob\_user\_get描述见下表:

**表 3-412. 函数 ob\_drp0\_get**

函数名称	ob_drp0_get
函数原型	uint16_t ob_drp0_get(void);
功能描述	获取bank0 DBUS读保护字节
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	bank0中开启擦/写保护和D-BUS读保护页的值

例如:

```
/* get the FMC erase/program protection and D-bus read protection option bytes value */
```

```

uint16_t bank0_drp_value = 0;
bank0_drp_value = ob_drp0_get();

```

### 函数 **ob\_drp1\_get**

函数ob\_drp1\_get描述见下表:

**表 3-413. 函数 ob\_drp1\_get**

函数名称	ob_drp1_get
函数原型	uint16_t ob_drp1_get(void);
功能描述	获取bank1 DBUS读保护字节
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<b>uint16_t</b>	bank1中开启擦/写保护和D-BUS读保护页的值

例如:

```

/* get the FMC erase/program protection and D-bus read protection option bytes value */
uint16_t bank1_drp_value = 0;
bank1_drp_value = ob_drp1_get();

```

### 函数 **ob\_spc\_get**

函数ob\_spc\_get描述见下表:

**表 3-414. 函数 ob\_spc\_get**

函数名称	ob_spc_get
函数原型	FlagStatus ob_spc_get(void);
功能描述	获取安全保护等级值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	安全保护是否设立（SET或RESET）

例如:

```
/* get the FMC option byte security protection */
```

```
FlagStatus spc_value = RESET;
```

```
spc_value = ob_spc_get( );
```

### **函数 ob\_user\_bor\_threshold\_get**

函数ob\_user\_bor\_threshold\_get描述见下表：

**表 3-415. 函数 ob\_user\_bor\_threshold\_get**

函数名称	ob_user_bor_threshold_get
函数原型	uint8_t ob_user_bor_threshold_get(void)
功能描述	获取选项字节BOR复位阈值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint8_t	选项字节BOR复位阈值

例如：

```
/* get the FMC option byte BOR threshold value */
```

```
uint8_t bor_value = 0;
```

```
bor_value = ob_user_bor_threshold_get( );
```

### **函数 fmc\_flag\_get**

函数fmc\_flag\_get描述见下表：

**表 3-416. 函数 fmc\_flag\_get**

函数名称	fmc_flag_get
函数原型	FlagStatus fmc_flag_get(uint32_t fmc_flag);
功能描述	检查标志是否置位
先决条件	-
被调用函数	-
输入参数{in}	
fmc_flag	受检查的FMC标志
FMC_FLAG_BUSY	FMC忙碌标志
FMC_FLAG_RDER R	D-BUS读错误标志
FMC_FLAG_PGSE	编程顺序错误标志

<i>RR</i>	
<i>FMC_FLAG_PGME</i> <i>RR</i>	编程类型不匹配错误标志
<i>FMC_FLAG_WPER</i> <i>R</i>	擦/写保护错误标志
<i>FMC_FLAG_OPER</i> <i>R</i>	闪存操作错误标志
<i>FMC_FLAG_END</i>	D-BUS读错误标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET 或 RESET

例如：

```
/* check the FMC_FLAG_END flag set or not*/
```

```
FlagStatus flag = RESET;
```

```
flag = fmc_flag_get(FMC_FLAG_END);
```

#### 函数 **fmc\_flag\_clear**

函数fmc\_flag\_clear描述见下表：

**表 3-417. 函数 fmc\_flag\_clear**

<b>函数名称</b>	fmc_flag_clear
<b>函数原型</b>	void fmc_flag_clear(uint32_t flag);
<b>功能描述</b>	清除FMC标志
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>fmc_flag</b>	待清除的FMC标志
<i>FMC_FLAG_BUSY</i>	FMC忙碌标志
<i>FMC_FLAG_RDER</i> <i>R</i>	D-BUS读错误标志
<i>FMC_FLAG_PGSE</i> <i>RR</i>	编程顺序错误标志
<i>FMC_FLAG_PGME</i> <i>RR</i>	编程类型不匹配错误标志
<i>FMC_FLAG_WPER</i> <i>R</i>	擦/写保护错误标志
<i>FMC_FLAG_OPER</i> <i>R</i>	闪存操作错误标志
<i>FMC_FLAG_END</i>	D-BUS读错误标志

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the FMC_FLAG_END flag */
fmc_flag_clear(FMC_FLAG_END);
```

#### 函数 **fmc\_interrupt\_enable**

函数fmc\_interrupt\_enable描述见下表：

**表 3-418. 函数 fmc\_interrupt\_enable**

函数名称	fmc_interrupt_enable
函数原型	void fmc_interrupt_enable(uint32_t interrupt);
功能描述	使能FMC中断
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
fmc_int	FMC中断使能选项
FMC_INT_END	FMC编程完成中断
FMC_INT_ERR	FMC错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable FMC end of program interrupt */

fmc_unlock( );
fmc_interrupt_enable(FMC_INT_END);
fmc_lock( );
```

#### 函数 **fmc\_interrupt\_disable**

函数fmc\_interrupt\_disable描述见下表：

**表 3-419. 函数 fmc\_interrupt\_disable**

函数名称	fmc_interrupt_disable
函数原型	void fmc_interrupt_disable(uint32_t interrupt);
功能描述	除能FMC中断

<b>先决条件</b>	fmc_unlock
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>fmc_int</b>	FMC中断除能选项
<i>FMC_INT_END</i>	FMC编程完成中断
<i>FMC_INT_ERR</i>	FMC错误中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable FMC end of program interrupt */

fmc_unlock( );

fmc_interrupt_disable(FMC_INT_END);

fmc_lock( );
```

#### 函数 **fmc\_interrupt\_flag\_get**

函数fmc\_interrupt\_flag\_get描述见下表：

**表 3-420. 函数 fmc\_interrupt\_flag\_get**

<b>函数名称</b>	fmc_interrupt_flag_get
<b>函数原型</b>	FlagStatus fmc_interrupt_flag_get(uint32_t fmc_int_flag);
<b>功能描述</b>	检查中断标志是否置位
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>fmc_int_flag</b>	受检查的FMC中断标志
<i>FMC_INT_FLAG_R_DERR</i>	D-BUS读错误中断标志
<i>FMC_INT_FLAG_P_GSERR</i>	编程顺序错误中断标志
<i>FMC_INT_FLAG_P_GMERR</i>	编程类型不匹配错误中断标志
<i>FMC_INT_FLAG_W_PERR</i>	擦/写保护错误中断标志
<i>FMC_INT_FLAG_O_PERR</i>	闪存操作错误中断标志
<i>FMC_INT_FLAG_E_ND</i>	完成中断标志
<b>输出参数{out}</b>	

-	-
返回值	
<b>FlagStatus</b>	SET 或 RESET

例如：

```
/* check the FMC_FLAG_END flag set or not*/
FlagStatus flag = RESET;
flag = fmc_interrupt_flag_get(FMC_INT_FLAG_END);
```

#### 函数 **fmc\_interrupt\_flag\_clear**

函数fmc\_interrupt\_flag\_clear描述见下表：

**表 3-421. 函数 fmc\_interrupt\_flag\_clear**

函数名称	fmc_interrupt_flag_clear
函数原型	void fmc_interrupt_flag_clear(uint32_t flag);
功能描述	清除FMC中断标志
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>fmc_int_flag</b>	待清除的FMC标志
<i>FMC_INT_FLAG_R DERR</i>	D-BUS读错误中断标志
<i>FMC_INT_FLAG_P GSERR</i>	编程顺序错误中断标志
<i>FMC_INT_FLAG_P GMERR</i>	编程类型不匹配错误中断标志
<i>FMC_INT_FLAG_W PERR</i>	擦/写保护错误中断标志
<i>FMC_INT_FLAG_O PERR</i>	闪存操作错误中断标志
<i>FMC_INT_FLAG_E ND</i>	完成中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* clear the FMC_FLAG_END flag */
fmc_interrupt_flag_clear(FMC_FLAG_END);
```

### 函数 **fmc\_state\_get**

函数fmc\_state\_get描述见下表:

**表 3-422. 函数 fmc\_state\_get**

函数名称	fmc_state_get	
函数原型	fmc_state_enum fmc_state_get(void);	
功能描述	获取FMC状态	
先决条件	-	
被调用函数	-	
输入参数{in}		
-	-	
输出参数{out}		
-	-	
返回值		
<b>fmc_state_enum</b>	FMC状态, 详情参考 <a href="#">枚举fmc_state_enum</a>	

例如:

```
/* get the FMC state */

fmc_state_enum state = fmc_state_get( );
```

### 函数 **fmc\_ready\_wait**

函数 fmc\_ready\_wait描述见下表:

**表 3-423. 函数 fmc\_ready\_wait**

函数名称	fmc_ready_wait	
函数原型	fmc_state_enum fmc_ready_wait(void);	
功能描述	检查FMC是否准备好	
先决条件	-	
被调用函数	fmc_state_get	
输入参数{in}		
-	-	
输出参数{out}		
-	-	
返回值		
<b>fmc_state_enum</b>	FMC状态, 详情参考 <a href="#">枚举fmc_state_enum</a>	

例如:

```
/* check whether FMC is ready or not */

fmc_state_enum state = fmc_ready_wait ( );
```

## 3.14. 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[3.14.1](#)描述了FWDGT的寄存器列表，章节[3.14.2](#)对FWDGT库函数进行说明。

### 3.14.1. 外设寄存器说明

FWDGT寄存器列表如下表所示：

表 3-424. 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

### 3.14.2. 外设库函数说明

FWDGT库函数列表如下表所示：

表 3-425. 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_counter_reload	按照FWDGT_RLD寄存器的值重装载IWDG计数器
fwdgt_config	设置FWDGT重装载值、预分频值
fwdgt_flag_get	获取FWDGT标志位状态

#### 函数 `fwdgt_write_enable`

函数fwdgt\_write\_enable描述见下表：

表 3-426. 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 3-427. 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 3-428. 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\_counter\_reload**

函数fwdgt\_counter\_reload描述见下表:

**表 3-429. 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\_config**

函数fwdgt\_config描述见下表：

**表 3-430. 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}	
<b>reload_value</b>	重装载值(0x0000 - 0xFFFF)-
输入参数{in}	
<b>prescaler_div</b>	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 prescaler set to 128

<i>FWDGT_PSC_DIV2</i> 56	FWDGT prescaler set to 256
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR or SUCCESS-

例如：

```
/* confiure FWDGT counter clock: 40KHz(IRC40K) / 64 = 0.625 KHz */
fwdgt_config(2*500, FWDGT_PSC_DIV64);
```

### 函数 **fwdgt\_flag\_get**

函数fwdgt\_flag\_get描述见下表：

**表 3-431. 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}	
<b>flag</b>	需要获取状态的FWDGT标志位
<i>FWDGT_FLAG_PUD</i>	预分频值更新进行中
<i>FWDGT_FLAG_RU</i> <i>D</i>	重装载值更新进行中
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	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

{

...

}

```

## 3.15. GPIO

GPIO用来实现各片上设备的逻辑输入/输出功能。章节[3.15.1](#)描述了GPIO的寄存器列表，章节[错误!未找到引用源。](#)对GPIO库函数进行说明。

### 3.15.1. 外设寄存器说明

GPIO寄存器列表如下表所示：

**表 3-432. GPIO 寄存器**

寄存器名称	寄存器描述
GPIOx_CTL	端口控制寄存器
GPIO_OMODE	端口输出模式寄存器
GPIO_OSPD	端口输出速度寄存器0
GPIO_PUD	端口上拉/下拉寄存器
GPIO_ISTAT	端口输入状态寄存器
GPIO_OCTL	端口输出控制寄存器
GPIO_BOP	端口位操作寄存器
GPIO_LOCK	端口配置锁定寄存器
GPIO_AFSEL0	备用功能选择寄存器0
GPIO_AFSEL1	备用功能选择寄存器1
GPIO_BC	位清除寄存器
GPIO_TG	端口位翻转寄存器

### 3.15.2. 外设库函数说明

GPIO库函数列表如下表所示：

**表 3-433. GPIO 库函数**

库函数名称	库函数描述
gpio_deinit	复位外设GPIOx

库函数名称	库函数描述
gpio_mode_set	设置GPIO模式
gpio_output_options_set	设置GPIO输出模式和速度
gpio_bit_set	置位引脚值
gpio_bit_reset	复位引脚值
gpio_bit_write	将特定的值写入引脚
gpio_port_write	将特定的值写入一组端口
gpio_input_bit_get	获取引脚的输入值
gpio_input_port_get	获取一组端口的输入值
gpio_output_bit_get	获取引脚的输出值
gpio_output_port_get	获取一组端口的输出值
gpio_af_set	设置GPIO复用功能
gpio_pin_lock	相应的引脚配置被锁定
gpio_bit_toggle	翻转GPIO引脚状态
gpio_port_toggle	翻转一组GPIO状态

### 函数 **gpio\_deinit**

函数gpio\_deinit描述见下表：

**表 3-434. 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,H,I)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset GPIOA */

gpio_deinit(GPIOA);
```

### 函数 **gpio\_mode\_set**

函数gpio\_mode\_set描述见下表：

**表 3-435. 函数 gpio\_mode\_set**

函数名称	gpio_mode_set

<b>函数原型</b>	void gpio_mode_set(uint32_t gpio_periph, uint32_t mode, uint32_t pull_up_down, uint32_t pin);
<b>功能描述</b>	设置GPIO模式
<b>先决条件</b>	-
<b>被调用函数</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>输入参数{in}</b>	
<b>gpio_periph</b>	GPIO端口
<b>GPIOx</b>	端口选择 (x=A,B,C,D,E,F,G,H,I)
<b>输入参数{in}</b>	
<b>mode</b>	GPIO引脚模式
<b>GPIO_MODE_INPUT</b>	输入模式
<b>GPIO_MODE_OUTPUT</b>	输出模式
<b>GPIO_MODE_AF</b>	备用功能模式
<b>GPIO_MODE_ANALOG</b>	模拟模式
<b>输入参数{in}</b>	
<b>pull_up_down</b>	GPIO引脚上拉下拉电阻设置
<b>GPIO_PUPD_NONE</b>	悬空模式，无上拉和下拉
<b>GPIO_PUPD_PULLUP</b>	带上拉电阻
<b>GPIO_PUPD_PULLDOWN</b>	带下拉电阻
<b>输入参数{in}</b>	
<b>pin</b>	GPIO引脚
<b>GPIO_PIN_x</b>	引脚选择 (x=0..15)
<b>GPIO_PIN_ALL</b>	所有引脚
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* set PA0 with pull-up input mode */
gpio_mode_set(GPIOA, GPIO_MODE_INPUT, GPIO_PUPD_PULLUP, GPIO_PIN_0);
```

### 函数 **gpio\_output\_options\_set**

函数**gpio\_output\_options\_set**描述见下表：

表 3-436. 函数 gpio\_output\_options\_set

函数名称	gpio_output_options_set
函数原型	void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t pin);
功能描述	设置GPIO输出模式和速度
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x=A,B,C,D,E,F,G,H,I)
输入参数{in}	
otype	GPIO引脚输出模式
GPIO_OTYPE_PP	推挽输出模式
GPIO_OTYPE_OD	开漏输出模式
输入参数{in}	
speed	GPIO引脚输出最大速度
GPIO_OSPEED_2MHZ	最大输出速度为2MHz
GPIO_OSPEED_10MHZ	最大输出速度为25MHz
GPIO_OSPEED_50MHZ	最大输出速度为50MHz
GPIO_OSPEED_M_AX	最大输出速度超过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：

```
/* configure PA0 as push pull mode */
gpio_output_options_set(GPIOA, GPIO_OTYPE_PP, GPIO_OSPEED_2MHZ, GPIO_PIN_0);
```

### 函数 gpio\_bit\_set

函数gpio\_bit\_set描述见下表：

表 3-437. 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	-
<b>输入参数{in}</b>	
gpio_periph	GPIO端口
GPIOx	端口选择 (x=A,B,C,D,E,F,G,H,I)
<b>输入参数{in}</b>	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x=0..15)
GPIO_PIN_ALL	所有引脚
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* set PA0*/
gpio_bit_set(GPIOA, GPIO_PIN_0);
```

### 函数 **gpio\_bit\_reset**

函数gpio\_bit\_reset描述见下表：

表 3-438. 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	-
<b>输入参数{in}</b>	
gpio_periph	GPIO端口
GPIOx	端口选择 (x=A,B,C,D,E,F,G,H,I)
<b>输入参数{in}</b>	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x=0..15)
GPIO_PIN_ALL	所有引脚
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* reset PA0*/
gpio_bit_set(GPIOA, GPIO_PIN_0);
```

### 函数 **gpio\_bit\_write**

函数gpio\_bit\_write描述见下表:

**表 3-439. 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}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x=A,B,C,D,E,F,G,H,I)
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
输入参数{in}	
bit_value	置位或清除
RESET	清除引脚值
SET	置位引脚值
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 3-440. 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GPIO端口
先决条件	-
被调用函数	-

输入参数{in}	
<b>gpio_periph</b>	GPIO端口
<b>GPIOx</b>	端口选择 (x=A,B,C,D,E,F,G,H,I)
输入参数{in}	
<b>data</b>	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 3-441. 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}	
<b>gpio_periph</b>	GPIO端口
<b>GPIOx</b>	端口选择 (x=A,B,C,D,E,F,G,H,I)
输入参数{in}	
<b>pin</b>	GPIO引脚
<b>GPIO_PIN_x</b>	引脚选择 (x=0..15)
<b>GPIO_PIN_ALL</b>	所有引脚
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或 RESET

例如：

```
/* get the 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 3-442. 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,H,I)	
输出参数{out}		
-	-	
返回值		
uint16_t	0x0000-0xFFFF	

例如：

```
/* get the input value of Port A */

uint16_t port_state;

port_state = gpio_input_bit_get (GPIOA);
```

### 函数 **gpio\_output\_bit\_get**

函数**gpio\_output\_bit\_get**描述见下表：

**表 3-443. 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GPIO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,H,I)	
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 the 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 3-444. 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GPIO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,H,I)
输出参数{out}	
-	-
返回值	
uint16_t	0x0000-0xFFFF

例如：

```
/* get the output value of Port A */

uint16_t port_state;

port_state = gpio_output_port_get(GPIOA);
```

### 函数 **gpio\_af\_set**

函数gpio\_af\_set描述见下表：

**表 3-445. 函数 gpio\_af\_set**

函数名称	gpio_af_set
函数原型	void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);
功能描述	设置GPIO的备用功能
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口

<b>GPIOx</b>	端口选择 (x=A,B,C,D,E,F,G,H,I)
<b>输入参数{in}</b>	
<b>alt_func_num</b>	GPIO 引脚备用功能, 请参见特定设备的数据手册
<b>GPIO_AF_0</b>	SYSTEM
<b>GPIO_AF_1</b>	TIMER0, TIMER1
<b>GPIO_AF_2</b>	TIMER2, TIMER3, TIMER4
<b>GPIO_AF_3</b>	TIMER7, TIMER8, TIMER9, TIMER10
<b>GPIO_AF_4</b>	I2C0, I2C1, I2C2
<b>GPIO_AF_5</b>	SPI0, SPI1, SPI2, SPI3, SPI4, SPI5
<b>GPIO_AF_6</b>	SPI2, SPI3, SPI4
<b>GPIO_AF_7</b>	USART0, USART1, USART2, SPI1, SPI2
<b>GPIO_AF_8</b>	UART3, UART4, USART5, UART6, UART7
<b>GPIO_AF_9</b>	CAN0, CAN1, TLI, TIMER11, TIMER12, TIMER13, I2C1, I2C2, CTC
<b>GPIO_AF_10</b>	USB_FS, USB_HS
<b>GPIO_AF_11</b>	ENET
<b>GPIO_AF_12</b>	EXMC, SDIO, USB_HS
<b>GPIO_AF_13</b>	DCI
<b>GPIO_AF_14</b>	TLI
<b>GPIO_AF_15</b>	EVENTOUT
<b>输入参数{in}</b>	
<b>pin</b>	GPIO引脚
<b>GPIO_PIN_x</b>	引脚选择 (x=0..15)
<b>GPIO_PIN_ALL</b>	所有引脚
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/*set PA0 alternate function 0*/
gpio_af_set(GPIOA, GPIO_AF_0, GPIO_PIN_0);
```

### 函数 **gpio\_pin\_lock**

函数gpio\_pin\_lock描述见下表：

**表 3-446. 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	-
<b>输入参数{in}</b>	

<b>gpio_periph</b>	GPIO端口
<b>GPIOx</b>	端口选择 (x=A,B,C,D,E,F,G,H,I)
<b>输入参数{in}</b>	
<b>pin</b>	GPIO引脚
<b>GPIO_PIN_x</b>	引脚选择 (x=0..15)
<b>GPIO_PIN_ALL</b>	所有引脚
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* lock PA0 */

gpio_pin_lock(GPIOA, GPIO_PIN_0);
```

### 函数 **gpio\_bit\_toggle**

函数**gpio\_bit\_toggle**描述见下表：

**表 3-447. 函数 **gpio\_bit\_toggle****

<b>函数名称</b>	gpio_bit_toggle
<b>函数原型</b>	void gpio_bit_toggle(uint32_t gpio_periph, uint32_t pin);
<b>功能描述</b>	翻转GPIO引脚状态
<b>先决条件</b>	
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>gpio_periph</b>	GPIO端口
<b>GPIOx</b>	端口选择 (x=A,B,C,D,E,F,G,H,I)
<b>输入参数{in}</b>	
<b>pin</b>	GPIO引脚
<b>GPIO_PIN_x</b>	引脚选择 (x=0..15)
<b>GPIO_PIN_ALL</b>	所有引脚
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* toggle PA0 */

gpio_bit_toggle(GPIOA, GPIO_PIN_0);
```

### 函数 **gpio\_port\_toggle**

函数**gpio\_port\_toggle**描述见下表：

**表 3-448. 函数 **gpio\_port\_toggle****

函数名称	gpio_port_toggle	
函数原型	void gpio_port_toggle(uint32_t gpio_periph);	
功能描述	翻转一组GPIO状态	
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,H,I)	
输出参数{out}		
-	-	
返回值		
-	-	

例如：

```
/* toggle GPIOA*/
gpio_port_toggle (GPIOA);
```

## 3.16. 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[3.16.1](#)描述了I2C的寄存器列表，章节[3.16.2](#)对I2C库函数进行说明。

### 3.16.1. 外设寄存器说明

I2C寄存器列表如下表所示：

**表 3-449. 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

寄存器名称	寄存器描述
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_FCTL	滤波器控制寄存器
I2C_SAMCS	SAM控制状态寄存器

### 3.16.2. 外设库函数说明

I2C库函数列表如下表所示：

**表 3-450. I2C 库函数**

库函数名称	库函数描述
i2c_deinit	复位外设I2C
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_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_data_transmit	发送数据
i2c_data_receive	接收数据
i2c_dma_enable	I2C DMA模式使能
i2c_dma_last_transfer_config	配置下一个DMA EOT是最后传输
i2c_stretch_scl_low_config	当从机数据没有准备好时是否拉低SCL

库函数名称	库函数描述
i2c_slave_response_to_gcall_config	从机是否响应广播呼叫
i2c_software_reset_config	配置I2C软件复位
i2c_pec_enable	报文错误校验使能
i2c_pec_transfer_enable	传输PEC值使能
i2c_pec_value_get	获取报文错误校验值
i2c_smbus_issue_alert	通过SMBA引脚发送警告
i2c_smbus_arp_enable	SMBus下ARP协议是否开启
i2c_analog_noise_filter_disable	模拟噪声滤波器禁止
i2c_analog_noise_filter_enable	模拟噪声滤波器使能
i2c_digital_noise_filter_config	数字噪声滤波器配置
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_flag_get	标志位获取
i2c_flag_clear	清除标志位
i2c_interrupt_enable	中断使能
i2c_interrupt_disable	中断除能
i2c_interrupt_flag_get	中断标志位获取
i2c_interrupt_flag_clear	中断标志位清除

### 函数 i2c\_deinit

函数i2c\_deinit描述见下表：

**表 3-451. 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\_clock\_config

函数i2c\_clock\_config描述见下表：

表 3-452. 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 clkspeed, 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=0,1,2)
输入参数{in}	
clkspeed	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
I2C_DTCY_16_9	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 3-453. 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}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
输入参数{in}	
i2cmode	模式选择
I2C_I2CMODE_ENABLE	I2C 模式
I2C_SMBUSMODE_ENABLE	SMBus 模式
输入参数{in}	
addformat	7bits 或 10bits
I2C_ADDFORMAT_7BITS	7bits

<i>I2C_ADDFORMAT_10BITS</i>	10bits
<b>输入参数{in}</b>	
<i>addr</i>	I2C地址
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-454. 函数 *i2c\_smbus\_type\_config***

函数名称	<i>i2c_smbus_type_config</i>
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	-
<b>输入参数{in}</b>	
<i>i2c_periph</i>	I2C外设
<i>I2Cx</i>	(x=0,1,2)
<b>输入参数{in}</b>	
<i>type</i>	主机或从机
<i>I2C_SMBUS_DEVICER</i>	从机
<i>I2C_SMBUS_HOST</i>	主机
<b>输出参数{out}</b>	

-	-
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 3-455. 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=0,1,2)
输入参数{in}	
ack	是否发送ACK
I2C_ACK_ENABLE	ACK 会被发送
I2C_ACK_DISABLE	ACK 不会发送
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 will sent ACK */

i2c_ack_config(I2C0, I2C_ACK_ENABLE);
```

### 函数 i2c\_ackpos\_config

函数i2c\_ackpos\_config描述见下表:

**表 3-456. 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}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
输入参数{in}	
pos	ACK位置
I2C_ACKPOS_CUR RENT	目前正在接收的字节是否发送ACK
I2C_ACKPOS_NEX T	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 3-457. 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	-
<b>输入参数{in}</b>	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
<b>输入参数{in}</b>	
addr	从机地址
<b>输入参数{in}</b>	
trandirection	发送或接收
I2C_TRANSMITTER	发送
I2C_RECEIVER	接收
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* send slave address to I2C bus and I2C0 act as receiver */
i2c_master_addressing(I2C0, I2C1_SLAVE_ADDRESS7, I2C_RECEIVER);
```

### 函数 i2c\_dualaddr\_enable

函数i2c\_dualaddr\_enable描述见下表：

表 3-458. 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	-
<b>输入参数{in}</b>	
<i>i2c_periph</i>	I2C外设
<i>I2Cx</i>	(x=0,1,2)
<b>输入参数{in}</b>	
<i>dualaddr</i>	是否使能
<i>I2C_DUADEN_DISABLE</i>	不使能双地址模式
<i>I2C_DUADEN_ENABLE</i>	使能双地址模式
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable I2C0 dual-address */
i2c_dualaddr_enable (I2C0, I2C_DUADEN_ENABLE);
```

### 函数 i2c\_enable

函数*i2c\_enable*描述见下表：

**表 3-459. 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	-
<b>输入参数{in}</b>	
<i>i2c_periph</i>	I2C外设
<i>I2Cx</i>	(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 3-460. 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 3-461. 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 3-462. 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\_data\_transmit

函数i2c\_data\_transmit描述见下表：

表 3-463. 函数 i2c\_data\_transmit

函数名称	i2c_data_transmit
函数原型	void i2c_data_transmit(uint32_t i2c_periph, uint8_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	传输的数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 transmit data */
i2c_data_transmit (I2C0);
```

### 函数 i2c\_data\_receive

函数i2c\_data\_receive描述见下表：

表 3-464. 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	
uint8_t	0x00..0xFF

例如：

```
/* I2C0 receive data */

uint8_t i2c_receiver;

i2c_receiver = i2c_data_receive (I2C0);
```

#### 函数 i2c\_dma\_enable

函数i2c\_dma\_enable描述见下表：

表 3-465. 函数 i2c\_dma\_enable

函数名称	i2c_dma_enable
函数原型	void i2c_dma_enable(uint32_t i2c_periph, uint32_t dmastate);
功能描述	I2C DMA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=0,1,2)

输入参数{in}	
<b>dmastate</b>	开启或关闭
<i>I2C_DMA_ON</i>	DMA模式开启
<i>I2C_DMA_OFF</i>	DMA模式关闭
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 DMA mode enable */
i2c_dma_enable (I2C0, I2C_DMA_ON);
```

#### 函数 **i2c\_dma\_last\_transfer\_enable**

函数*i2c\_dma\_last\_transfer\_enable*描述见下表：

**表 3-466. 函数 *i2c\_dma\_last\_transfer\_enable***

函数名称	i2c_dma_last_transfer_enable
函数原型	void i2c_dma_last_transfer_enable(uint32_t i2c_periph, uint32_t dmalast);
功能描述	使能下一个DMA EOT是最后传输
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=0,1,2)
输入参数{in}	
<b>dmalast</b>	是否使能下一个DMA EOT是最后传输
<i>I2C_DMALST_ON</i>	使能
<i>I2C_DMALST_OFF</i>	不使能
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\_stretch\_scl\_low\_config

函数i2c\_stretch\_scl\_low\_config描述见下表：

**表 3-467. 函数 i2c\_stretch\_scl\_low\_config**

函数名称	i2c_stretch_scl_low_config
函数原型	void i2c_stretch_scl_low_config(uint32_t i2c_periph, uint32_t stretchpara);
功能描述	当从机数据没有准备好时是否拉低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)
输入参数{in}	
stretchpara	是否拉低SCL
I2C_SCLSTRETCH_ENABLE	不拉低SCL
I2C_SCLSTRETCH_DISABLE	不拉低SCL
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_config (I2C0, I2C_SCLSTRETCH_ENABLE);
```

### **函数 i2c\_slave\_response\_to\_gcall\_config**

函数i2c\_slave\_response\_to\_gcall\_config描述见下表:

**表 3-468. 函数 i2c\_slave\_response\_to\_gcall\_config**

函数名称	i2c_slave_response_to_gcall_config
函数原型	void i2c_slave_response_to_gcall_config(uint32_t i2c_periph, uint32_t gcallpara);
功能描述	从机是否响应广播呼叫
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
输入参数{in}	
gcallpara	是否响应广播呼叫
I2C_GCEN_ENABLE	从机响应广播呼叫
I2C_GCEN_DISABLE	从机不响应广播呼叫
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* I2C0 will response to a general call */
i2c_slave_response_to_gcall_config (I2C0, I2C_GCEN_ENABLE);
```

### **函数 i2c\_software\_reset\_config**

函数i2c\_software\_reset\_config描述见下表:

**表 3-469. 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	-
<b>输入参数{in}</b>	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
<b>输入参数{in}</b>	
sreset	是否复位
I2C_SRESET_SET	复位
I2C_SRESET_RES ET	没有复位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* software reset I2C0*/
i2c_software_reset_config (I2C0, I2C_SRESET_SET);
```

#### 函数 i2c\_pec\_enable

函数i2c\_pec\_enable描述见下表：

表 3-470. 函数 i2c\_pec\_enable

函数名称	i2c_pec_enable
函数原型	void i2c_pec_enable(uint32_t i2c_periph, uint32_t pecstate);
功能描述	报文错误校验使能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	

<b>i2c_periph</b>	I2C外设
<i>I2Cx</i>	(x=0,1,2)
<b>输入参数{in}</b>	
<b>pecpara</b>	开启或关闭
<i>I2C_PEC_ENABLE</i>	报文错误校验使能
<i>I2C_PEC_DISABLE</i>	报文错误校验关闭
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable I2C PEC calculation */

i2c_pec_enable (I2C0, I2C_PEC_ENABLE);
```

#### 函数 **i2c\_pec\_transfer\_enable**

函数*i2c\_pec\_transfer\_enable*描述见下表：

**表 3-471. 函数 i2c\_pec\_transfer\_enable**

函数名称	i2c_pec_transfer_enable
函数原型	void i2c_pec_transfer_enable(uint32_t i2c_periph, uint32_t pecpara);
功能描述	传输PEC值使能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>i2c_periph</b>	I2C外设
<i>I2Cx</i>	(x=0,1,2)
<b>输入参数{in}</b>	
<b>pecpara</b>	是否传输PEC
<i>I2C_PECTRANS_ENABLE</i>	传输PEC

<i>I2C_PECTRANS_DISABLE</i>	不传输PEC
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 transfer PEC */
i2c_pec_transfer_enable (I2C0, I2C_PECTRANS_ENABLE);
```

#### 函数 **i2c\_pec\_value\_get**

函数*i2c\_pec\_value\_get*描述见下表：

表 3-472. 函数 **i2c\_pec\_value\_get**

函数名称	i2c_pec_value_get
函数原型	uint8_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}	
<i>i2c_periph</i>	I2C外设
<i>I2Cx</i>	(x=0,1,2)
输出参数{out}	
-	-
返回值	
<b>uint8_t</b>	PEC值

例如：

```
/* I2C0 get packet error checking value */
uint8_t pec_value;
pec_value = i2c_pec_value_get (I2C0);
```

### 函数 i2c\_smbus\_issue\_alert

函数i2c\_smbus\_issue\_alert描述见下表:

**表 3-473. 函数 i2c\_smbus\_issue\_alert**

函数名称	i2c_smbus_issue_alert
函数原型	void i2c_smbus_issue_alert(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}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
输入参数{in}	
smbuspara	是否通过SMBA引脚发送警告
I2C_SALTSEND_ENABLE	通过SMBA引脚发送警告
I2C_SALTSEND_DISABLE	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_issue_alert(I2C0, I2C_SALTSEND_ENABLE);
```

### 函数 i2c\_smbus\_arp\_enable

函数i2c\_smbus\_arp\_enable描述见下表:

**表 3-474. 函数 i2c\_smbus\_arp\_enable**

函数名称	i2c_smbus_arp_enable
函数原型	void i2c_smbus_arp_enable(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=0,1,2)
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_enable (I2C0, I2C_ARP_ENABLE);
```

#### 函数 i2c\_analog\_noise\_filter\_disable

函数i2c\_analog\_noise\_filter\_disable描述见下表：

表 3-475. 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	禁止模拟噪声滤波器
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2)

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C0 analog noise filter */
```

```
i2c_analog_noise_filter_disable (I2C0);
```

#### 函数 **i2c\_analog\_noise\_filter\_enable**

函数*i2c\_analog\_noise\_filter\_enable*描述见下表：

**表 3-476. 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
I2Cx	(x=0,1,2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 analog noise filter */
```

```
i2c_analog_noise_filter_enable (I2C0);
```

#### 函数 **i2c\_digital\_noise\_filter\_config**

函数*i2c\_digital\_noise\_filter\_config*描述见下表：

**表 3-477. 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, i2c_digital_filter_enum dfilterpara);
功能描述	config digital noise filter
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}	
dfilterpara	滤除的噪声尖端峰值
i2c_digital_filter_enum	能够滤除的噪声尖端峰值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config I2C0 digital noise filter as I2C_DF_1PCLK */
i2c_digital_noise_filter_config (I2C0, I2C_DF_1PCLK);
```

#### 函数 i2c\_sam\_enable

函数i2c\_sam\_enable描述见下表：

**表 3-478. 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=0,1,2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 SAM_V interface*/
i2c_sam_enable (I2C0);
```

### 函数 i2c\_sam\_disable

函数i2c\_sam\_disable描述见下表：

表 3-479. 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=0,1,2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C0 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 3-480. 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=0,1,2)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable I2C0 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 3-481. 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

<i>I2Cx</i>	(x=0,1,2)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable I2C0 SAM_V interface timeout detect */

i2c_sam_timeout_disable (I2C0);
```

### 函数 **i2c\_flag\_get**

函数*i2c\_flag\_get*描述见下表：

**表 3-482. 函数 i2c\_flag\_get**

函数名称	<b>i2c_flag_get</b>
函数原型	FlagStatus i2c_flag_get(uint32_t i2c_periph,uint32_t flag );
功能描述	标志位获取
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>i2c_periph</b>	I2C外设
<i>I2Cx</i>	(x=0,1,2)
<b>输入参数{in}</b>	
<b>flag</b>	需要获取的标志位
<i>I2C_FLAG_SBSEN_D</i>	起始位是否发送
<i>I2C_FLAG_ADDSEN_D</i>	主机模式下地址是否发送/从机模式下地址是否匹配
<i>I2C_FLAG_BTC</i>	字节传输完成
<i>I2C_FLAG_ADD10SEND</i>	主机模式下10位地址地址头发送完成
<i>I2C_FLAG_STPDE</i>	从机模式下监测到STOP结束位

<i>T</i>	
<i>I2C_FLAG_RBNE</i>	接收期间I2C_DATA非空
<i>I2C_FLAG_TBE</i>	发送期间I2C_DATA为空
<i>I2C_FLAG_BERR</i>	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
<i>I2C_FLAG_LOSTA RB</i>	主机模式下仲裁丢失
<i>I2C_FLAG_AERR</i>	应答错误
<i>I2C_FLAG_OUERR</i>	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
<i>I2C_FLAG_PECER R</i>	接收数据时PEC错误
<i>I2C_FLAG_SMBTO</i>	SMBus模式下超时信号
<i>I2C_FLAG_SMBAL T</i>	SMBus警报状态
<i>I2C_FLAG_MASTE R</i>	表明I2C时钟在主机模式还是从机模式的标志位
<i>I2C_FLAG_I2CBSY</i>	忙标志
<i>I2C_FLAG_TRS</i>	I2C作发送端还是接收端
<i>I2C_FLAG_RXGC</i>	是否接收到广播地址(00h)
<i>I2C_FLAG_DEFSM B</i>	从机模式下SMBus主机地址头
<i>I2C_FLAG_HSTSM B</i>	从机模式下监测到SMBus主机地址头
<i>I2C_FLAG_DUMOD</i>	从机模式下双标志位表明哪个地址和双地址模式匹配
<i>I2C_FLAG_TFF</i>	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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	

<b>FlagStatus</b>	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 3-483. 函数 *i2c\_flag\_clear***

函数名称	i2c_flag_clear
函数原型	void i2c_flag_clear(uint32_t i2c_periph, uint32_t flag);
功能描述	清除标志位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
输入参数{in}	
flag	标志位类型
I2C_FLAG_SMBAL_T	SMBus警报状态
I2C_FLAG_SMBTO	SMBus模式下超时信号
I2C_FLAG_PECER_R	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_LOSTA_RB	主机模式下仲裁丢失
I2C_FLAG_BERR	总线错误
I2C_FLAG_ADDSE	主机模式下地址是否发送/从机模式下地址是否匹配，通过读I2C_STAT0和

<i>ND</i>	I2C_STAT1来清除
<i>I2C_FLAG_TFF</i>	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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-484. 函数 i2c\_interrupt\_enable

函数名称	i2c_interrupt_enable
函数原型	void i2c_interrupt_enable(uint32_t i2c_periph, uint32_t inttype);
功能描述	中断使能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<i>i2c_periph</i>	I2C外设
<i>I2Cx</i>	(x=0,1,2)
<b>输入参数{in}</b>	
<i>inttype</i>	中断类型
<i>I2C_INT_ERR</i>	错误中断使能
<i>I2C_INT_EV</i>	事件中断使能
<i>I2C_INT_BUF</i>	缓冲区中断使能

<i>I2C_INT_TFF</i>	发送帧下降沿中断使能
<i>I2C_INT_TFR</i>	发送帧上升沿中断使能
<i>I2C_INT_RFF</i>	接收帧下降沿中断使能
<i>I2C_INT_RFR</i>	接收帧上升沿中断使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-485. 函数 i2c\_interrupt\_disable**

函数名称	i2c_interrupt_disable
函数原型	void i2c_interrupt_disable(uint32_t i2c_periph, uint32_t inttype);
功能描述	中断除能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<i>i2c_periph</i>	I2C外设
<i>I2Cx</i>	(x=0,1,2)
<b>输入参数{in}</b>	
<i>inttype</i>	中断类型
<i>I2C_INT_ERR</i>	错误中断使能
<i>I2C_INT_EV</i>	事件中断使能
<i>I2C_INT_BUF</i>	缓冲区中断使能
<i>I2C_INT_TFF</i>	发送帧下降沿中断使能

<i>I2C_INT_TFR</i>	发送帧上升沿中断使能
<i>I2C_INT_RFF</i>	接收帧下降沿中断使能
<i>I2C_INT_RFR</i>	接收帧上升沿中断使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-486. 函数 i2c\_interrupt\_flag\_get**

函数名称	<i>i2c_interrupt_flag_get</i>
函数原型	FlagStatus <i>i2c_interrupt_flag_get(uint32_t i2c_periph, uint32_t intflag);</i>
功能描述	中断标志位获取
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<i>i2c_periph</i>	I2C外设
<i>I2Cx</i>	(x=0,1,2)
<b>输入参数{in}</b>	
<i>int_flag</i>	中断标志
<i>I2C_INT_FLAG_SB_SEND</i>	主机模式下发送START起始位
<i>I2C_INT_FLAG_AD_DSEND</i>	主机模式下成功发送了地址 / 从机模式下接收到地址并且和自身的地址匹配
<i>I2C_INT_FLAG_BT_C</i>	字节发送结束

<i>I2C_INT_FLAG_AD</i>	主机模式下10位地址地址头被发送
<i>I2C_INT_FLAG_ST</i>	从机模式下监测到STOP结束位
<i>I2C_INT_FLAG_RB</i>	接收期间I2C_DATA非空
<i>I2C_INT_FLAG_TB</i>	发送期间I2C_DATA为空
<i>I2C_INT_FLAG_BE</i>	总线错误
<i>I2C_INT_FLAG_LO</i>	主机模式下仲裁丢失
<i>I2C_INT_FLAG_AE</i>	应答错误
<i>I2C_INT_FLAG_OU</i>	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
<i>I2C_INT_FLAG_PE</i>	接收数据时PEC错误
<i>I2C_INT_FLAG_SM</i>	SMBus模式下超时信号
<i>I2C_INT_FLAG_SM</i>	SMBus警报状态
<i>I2C_INT_FLAG_TF</i>	发送帧下降沿中断标志位
<i>I2C_INT_FLAG_TF</i>	发送帧上升沿中断标志位
<i>I2C_INT_FLAG_RF</i>	接收帧下降沿中断标志位
<i>I2C_INT_FLAG_RF</i>	接收帧上升沿中断标志位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	

<b>FlagStatus</b>	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 3-487. 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, uint32_t intflag);
功能描述	中断标志位清除
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}	
int_flag	中断标志
I2C_INT_FLAG_ADDRESSSEND	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_BUSERR	总线错误
I2C_INT_FLAG_LOSTARB	主机模式下仲裁丢失
I2C_INT_FLAG_ACKERR	应答错误
I2C_INT_FLAG_OVERRUNERR	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

<i>I2C_INT_FLAG_SM_BTO</i>	SMBus模式下超时信号
<i>I2C_INT_FLAG_SM_BALT</i>	SMBus警报状态
<i>I2C_INT_FLAG_TF_F</i>	发送帧下降沿中断标志位
<i>I2C_INT_FLAG_TF_R</i>	发送帧上升沿中断标志位
<i>I2C_INT_FLAG_RF_F</i>	接收帧下降沿中断标志位
<i>I2C_INT_FLAG_RF_R</i>	接收帧上升沿中断标志位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* clear the acknowledge error interrupt flag */
i2c_interrupt_flag_clear (I2C0, I2C_INT_FLAG_AERR);
```

## 3.17. IPA

IPA提供从某一个或两个源图像到目标图像的可配置的，灵活的图像处理功能。章节[3.17.1](#)描述了IPA的寄存器列表，章节[3.17.2](#)对IPA库函数进行说明。

### 3.17.1. 外设寄存器说明

IPA 寄存器列表如下表所示：

**表 3-488. IPA 寄存器**

寄存器名称	寄存器描述
IPA_CTL	控制寄存器
IPA_INTF	中断状态寄存器
IPA_INTC	中断标志清除寄存器
IPA_FMADDR	前景层存储区基地址寄存器
IPA_FLOFF	前景层行偏移寄存器

寄存器名称	寄存器描述
IPA_BMADDR	背景层存储区基地址寄存器
IPA_BLOFF	背景层行偏移寄存器
IPA_FPCTL	前景层像素控制寄存器
IPA_FPV	前景层像素值寄存器
IPA_BPCTL	背景层像素控制寄存器
IPA_BPV	背景层像素值寄存器
IPA_FLMADDR	前景层 LUT 存储区基地址寄存器
IPA_BLMADDR	背景层 LUT 存储区基地址寄存器
IPA_DPCTL	目标像素控制寄存器
IPA_DPV	目标像素值寄存器
IPA_DMADDR	目标存储区基地址寄存器
IPA_DLOFF	目标行偏移寄存器
IPA_IMS	图像大小寄存器
IPA_LM	行标记寄存器
IPA_ITCTL	内部定时器控制寄存器

### 3.17.2. 外设库函数说明

IPA 库函数列表如下表所示：

**表 3-489. IPA 库函数**

库函数名称	库函数描述
ipa_deinit	重新初始化IPA
ipa_transfer_enable	使能IPA传输
ipa_transfer_hangup_enable	使能IPA传输挂起
ipa_transfer_hangup_disable	禁能IPA传输挂起
ipa_transfer_stop_enable	使能IPA传输停止
ipa_transfer_stop_disable	禁能IPA传输停止
ipa_foreground_lut_loading_enable	使能前景层LUT加载
ipa_background_lut_loading_enable	使能背景层LUT加载
ipa_pixel_format_convert_mode_set	设置像素格式转换模式，该函数在IPA传输使能的情况下调用无效
ipa_foreground_struct_para_init	用默认值初始化IPA前景层参数结构体，建议在定义一个ipa_foreground_parameter_struct结构体后调用该接口实现对结构体的初始化
ipa_foreground_init	初始化前景层参数
ipa_background_struct_para_init	用默认值初始化IPA背景层参数结构体，建议在定义一个ipa_background_parameter_struct结构体后调用该接口实现对结构体的初始化
ipa_background_init	初始化背景层参数
ipa_destination_struct_para_init	用默认值初始化IPA目标参数结构体，建议在定义一个ipa_destination_parameter_struct结构体后调用该接口实现对

库函数名称	库函数描述
	结构体的初始化
ipa_destination_init	初始化目标参数
ipa_foreground_lut_init	初始化IPA前景层LUT参数
ipa_background_lut_init	初始化IPA背景层LUT参数
ipa_line_mark_config	配置IPA行标记
ipa_inter_timer_config	配置IPA内部定时器使能或禁能
ipa_interval_clock_num_config	配置间隔时钟周期数
ipa_flag_get	从IPA_INTF寄存器获取IPA标志位状态
ipa_flag_clear	清除IPA标志位
ipa_interrupt_enable	使能IPA中断
ipa_interrupt_disable	禁能IPA中断
ipa_interrupt_flag_get	获取IPA中断标志位
ipa_interrupt_flag_clear	清除IPA中断标志位

### 结构体 ipa\_foreground\_parameter\_struct

表 3-490. 结构体 ipa\_foreground\_parameter\_struct

成员名称	功能描述
foreground_memaddr	前景层存储区地址
foreground_lineoff	前景层行偏移
foreground_prealpha	前景层预定义alpha通道值
foreground_alpha_algorithm	前景层alpha值计算算法
foreground_pf	前景层像素格式
foreground_prered	前景层预定义红色值
foreground_pregreen	前景层预定义绿色值
foreground_preblue	前景层预定义蓝色值

### 结构体 ipa\_background\_parameter\_struct

表 3-491. 结构体 ipa\_background\_parameter\_struct

成员名称	功能描述
background_memaddr	背景层存储区地址
background_lineoff	背景层行偏移
background_prealpha	背景层预定义alpha通道值
background_alpha_algorithm	背景层alpha值计算算法
background_pf	背景层像素格式
background_prered	背景层预定义红色值
background_pregreen	背景层预定义绿色值
background_preblue	背景层预定义蓝色值

### 结构体 ipa\_destination\_parameter\_struct

**表 3-492. 结构体 ipa\_destination\_parameter\_struct**

成员名称	功能描述
destination_memaddr	目标存储区地址
destination_lineoff	目标存储区行偏移
destination_prealpha	目标存储区预定义alpha通道值
destination_pf	目标存储区像素格式
destination_prered	目标存储区预定义红色值
destination_pregreen	目标存储区预定义绿色值
destination_preblue	目标存储区预定义蓝色值
image_width	图像宽度
image_height	图像高度

### 函数 ipa\_deinit

函数ipa\_deinit描述见下表:

**表 3-493. 函数 ipa\_deinit**

函数名称	ipa_deinit
函数原型	void ipa_deinit(void);
功能描述	重新初始化IPA
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 IPA */
ipa_deinit();
```

### 函数 ipa\_transfer\_enable

函数ipa\_transfer\_enable描述见下表:

**表 3-494. 函数 ipa\_transfer\_enable**

函数名称	ipa_transfer_enable
函数原型	void ipa_transfer_enable(void);
功能描述	使能IPA传输
先决条件	-

<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 IPA transfer */
ipa_transfer_enable();
```

### 函数 ipa\_transfer\_hangup\_enable

函数ipa\_transfer\_hangup\_enable描述见下表：

**表 3-495. 函数 ipa\_transfer\_hangup\_enable**

<b>函数名称</b>	ipa_transfer_hangup_enable
<b>函数原型</b>	void ipa_transfer_hangup_enable(void);
<b>功能描述</b>	使能IPA传输挂起
<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 IPA transfer hang up */
ipa_transfer_hangup_enable();
```

### 函数 ipa\_transfer\_hangup\_disable

函数ipa\_transfer\_hangup\_disable描述见下表：

**表 3-496. 函数 ipa\_transfer\_hangup\_disable**

<b>函数名称</b>	ipa_transfer_hangup_disable
<b>函数原型</b>	void ipa_transfer_hangup_disable(void);
<b>功能描述</b>	禁能IPA传输挂起
<b>先决条件</b>	-
<b>被调用函数</b>	-

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable IPA transfer hang up */

ipa_transfer_hangup_disable();
```

#### 函数 ipa\_transfer\_stop\_enable

函数ipa\_transfer\_stop\_enable描述见下表：

表 3-497. 函数 ipa\_transfer\_stop\_enable

函数名称	ipa_transfer_stop_enable
函数原型	void ipa_transfer_stop_enable(void);
功能描述	使能IPA传输停止
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 IPA transfer stop */

ipa_transfer_stop_enable();
```

#### 函数 ipa\_transfer\_stop\_disable

函数ipa\_transfer\_stop\_disable描述见下表：

表 3-498. 函数 ipa\_transfer\_stop\_disable

函数名称	ipa_transfer_stop_disable
函数原型	void ipa_transfer_stop_disable(void);
功能描述	禁能IPA传输停止
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 IPA transfer stop */

ipa_transfer_stop_disable();
```

#### 函数 ipa\_foreground\_lut\_loading\_enable

函数ipa\_foreground\_lut\_loading\_enable描述见下表：

**表 3-499. 函数 ipa\_foreground\_lut\_loading\_enable**

函数名称	ipa_foreground_lut_loading_enable
函数原型	void ipa_foreground_lut_loading_enable(void);
功能描述	使能前景层LUT加载
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable IPA foreground LUT loading */

ipa_foreground_lut_loading_enable();
```

#### 函数 ipa\_background\_lut\_loading\_enable

函数ipa\_background\_lut\_loading\_enable描述见下表：

**表 3-500. 函数 ipa\_background\_lut\_loading\_enable**

函数名称	ipa_background_lut_loading_enable
函数原型	void ipa_background_lut_loading_enable(void);
功能描述	使能背景层LUT加载
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable IPA background LUT loading */

ipa_background_lut_loading_enable();
```

#### 函数 ipa\_pixel\_format\_convert\_mode\_set

函数ipa\_pixel\_format\_convert\_mode\_set描述见下表：

表 3-501. 函数 ipa\_pixel\_format\_convert\_mode\_set

函数名称	ipa_pixel_format_convert_mode_set
函数原型	void ipa_pixel_format_convert_mode_set(uint32_t pfcm);
功能描述	设置像素格式转换模式，该函数在IPA传输使能的情况下调用无效
先决条件	-
被调用函数	-
输入参数{in}	
<b>pfcm</b>	像素格式转换模式
<i>IPA_FGTODE</i>	前景层存储区到目标存储区无像素格式转换
<i>IPA_FGTODE_PF_CONVERT</i>	前景层存储区到目标存储区有像素格式转换
<i>IPA_FGBGTODE</i>	混合前景层和背景层存储区到目标存储区
<i>IPA_FILL_UP_DE</i>	用特定的颜色填充目标存储区
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure foreground memory to destination memory with pixel format convert */

ipa_pixel_format_convert_mode_set(IPA_FGTODE_PF_CONVERT);
```

#### 函数 ipa\_foreground\_struct\_para\_init

函数ipa\_foreground\_struct\_para\_init描述见下表：

表 3-502. 函数 ipa\_foreground\_struct\_para\_init

函数名称	ipa_foreground_struct_para_init
函数原型	void ipa_foreground_struct_para_init(ipa_foreground_parameter_struct* foreground_struct);

功能描述	用默认值初始化IPA前景层参数结构体，建议在定义一个ipa_foreground_parameter_struct结构体后调用该接口实现对结构体的初始化
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
*foreground_struct	指向 ipa_foreground_parameter_struct结构体的指针
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
ipa_foreground_parameter_struct fg_struct;
/* initialize the structure of IPA foreground parameter struct with the default values */
ipa_foreground_struct_para_init(&fg_struct);
```

### 函数 ipa\_foreground\_init

函数ipa\_foreground\_init描述见下表：

**表 3-503. 函数 ipa\_foreground\_init**

函数名称	ipa_foreground_init
函数原型	void ipa_foreground_init(ipa_foreground_parameter_struct* foreground_struct);
功能描述	初始化前景层参数
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
*foreground_struct	指向 ipa_foreground_parameter_struct结构体的指针
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
ipa_foreground_parameter_struct ipa_fg_init_struct;
ipa_foreground_struct_para_init(&ipa_fg_init_struct);
/* configure IPA foreground */
ipa_fg_init_struct.foreground_memaddr = (uint32_t)&gBuffer;
ipa_fg_init_struct.foreground_pf = FOREGROUND_PPF_RGB565;
ipa_fg_init_struct.foreground_alpha_algorithm = IPA_FG_ALPHA_MODE_1;
```

```

ipa_fg_init_struct.foreground_prealpha = 0x75;
ipa_fg_init_struct.foreground_lineoff = 0x00;
ipa_fg_init_struct.foreground_preblue = 0x00;
ipa_fg_init_struct.foreground_pregreen = 0x00;
ipa_fg_init_struct.foreground_prered = 0x00;
/* foreground initialization */

ipa_foreground_init(&ipa_fg_init_struct);

```

### **函数 ipa\_background\_struct\_para\_init**

函数ipa\_background\_struct\_para\_init描述见下表:

**表 3-504. 函数 ipa\_background\_struct\_para\_init**

函数名称	ipa_background_struct_para_init
函数原型	void ipa_background_struct_para_init(ipa_background_parameter_struct* background_struct);
功能描述	用默认值初始化IPA背景层参数结构体，建议在定义一个ipa_background_parameter_struct结构体后调用该接口实现对结构体的初始化
先决条件	-
被调用函数	-
输入参数{in}	
*background_struct	指向ipa_background_parameter_struct结构体的指针
输出参数{out}	
-	-
返回值	
-	-

例如:

```

ipa_background_parameter_struct bg_struct;
/* initialize the structure of IPA background parameter struct with the default values */
ipa_background_struct_para_init(&bg_struct);

```

### **函数 ipa\_background\_init**

函数ipa\_background\_init描述见下表:

**表 3-505. 函数 ipa\_background\_init**

函数名称	ipa_background_init
函数原型	void ipa_background_init(ipa_background_parameter_struct* background_struct);
功能描述	初始化背景层参数

先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
*background_struct	指向ipa_background_parameter_struct结构体的指针
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```

ipa_background_parameter_struct    ipa_bg_init_struct;

ipa_background_struct_para_init(&ipa_bg_init_struct);

/* configure IPA background */

ipa_bg_init_struct.background_memaddr = (uint32_t)&gBuffer;

ipa_bg_init_struct.background_pf = BACKGROUND_PPF_RGB565;

ipa_bg_init_struct.background_alpha_algorithm = IPA_BG_ALPHA_MODE_0;

ipa_bg_init_struct.background_prealpha = 255;

ipa_bg_init_struct.background_lineoff = 0x00;

ipa_bg_init_struct.background_preblue = 0x00;

ipa_bg_init_struct.background_pregreen = 0x00;

ipa_bg_init_struct.background_prered = 0x00;

/* background initialization */

ipa_background_init(&ipa_bg_init_struct);

```

### 函数 ipa\_destination\_struct\_para\_init

函数ipa\_destination\_struct\_para\_init描述见下表：

**表 3-506. 函数 ipa\_destination\_struct\_para\_init**

函数名称	ipa_destination_struct_para_init
函数原型	void ipa_destination_struct_para_init(ipa_destination_parameter_struct* destination_struct);
功能描述	用默认值初始化IPA目标参数结构体，建议在定义一个ipa_destination_parameter_struct结构体后调用该接口实现对结构体的初始化
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
*destination_struct	指向ipa_destination_parameter_struct结构体的指针

输出参数{out}	
-	-
返回值	
-	-

例如：

```
ipa_destination_parameter_struct destination_struct;
/* initialize the structure of IPA destination parameter struct with the default values */
ipa_destination_struct_para_init(&destination_struct);
```

### 函数 ipa\_destination\_init

函数ipa\_destination\_init描述见下表：

**表 3-507. 函数 ipa\_destination\_init**

函数名称	ipa_destination_init
函数原型	void ipa_destination_init(ipa_destination_parameter_struct* destination_struct);
功能描述	初始化目标参数
先决条件	-
被调用函数	-
输入参数{in}	
*destination_struct	指向ipa_destination_parameter_struct结构体的指针
输出参数{out}	
-	-
返回值	
-	-

例如：

```
ipa_destination_parameter_struct ipa_destination_init_struct;
ipa_destination_struct_para_init(&ipa_destination_init_struct);
/* configure destination pixel format */
ipa_destination_init_struct.destination_pf = IPA_DPF_RGB565;
/* configure destination memory base address */
ipa_destination_init_struct.destination_memaddr = (uint32_t)&gBuffer;
/* configure destination pre-defined alpha value RGB */
ipa_destination_init_struct.destination_pregreen = 0;
ipa_destination_init_struct.destination_preblue = 0;
ipa_destination_init_struct.destination_prered = 0;
```

```

ipa_destination_init_struct.destination_prealpha = 0;

/* configure destination line offset */

ipa_destination_init_struct.destination_lineoff = 0;

/* configure height of the image to be processed */

ipa_destination_init_struct.image_height = 160;

/* configure width of the image to be processed */

ipa_destination_init_struct.image_width = 229;

/* ipa destination initialization */

ipa_destination_init(&ipa_destination_init_struct);

```

### **函数 ipa\_foreground\_lut\_init**

函数ipa\_foreground\_lut\_init描述见下表：

**表 3-508. 函数 ipa\_foreground\_lut\_init**

函数名称	ipa_foreground_lut_init
函数原型	void ipa_foreground_lut_init(uint8_t fg_lut_num, uint8_t fg_lut_pf, uint32_t fg_lut_addr);
功能描述	初始化IPA前景层LUT参数
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
fg_lut_num	前景层LUT像素数目
<b>输入参数{in}</b>	
fg_lut_pf	前景层LUT像素格式
IPA_LUT_PF_ARG B8888	LUT像素格式是ARGB8888
IPA_LUT_PF_RGB 888	LUT像素格式是RGB888
<b>输入参数{in}</b>	
fg_lut_addr	前景层LUT存储区基地址
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```

/* initialize the LUT foreground LUT parameters */

ipa_foreground_lut_init(1, IPA_LUT_PF_ARGB8888, 0x20002000);

```

### 函数 ipa\_background\_lut\_init

函数ipa\_background\_lut\_init描述见下表：

**表 3-509. 函数 ipa\_background\_lut\_init**

函数名称	ipa_background_lut_init
函数原型	void ipa_background_lut_init(uint8_t bg_lut_num, uint8_t bg_lut_pf, uint32_t bg_lut_addr);
功能描述	初始化IPA背景层LUT参数
先决条件	-
被调用函数	-
输入参数{in}	
bg_lut_num	背景层LUT像素数目
输入参数{in}	
bg_lut_pf	背景层LUT像素格式
IPA_LUT_PF_ARG_B8888	LUT像素格式是ARGB8888
IPA_LUT_PF_RGB888	LUT像素格式是RGB888
输入参数{in}	
bg_lut_addr	背景层LUT存储区基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the LUT background LUT parameters */
ipa_background_lut_init(2, IPA_LUT_PF_RGB888, 0x20001000);
```

### 函数 ipa\_line\_mark\_config

函数ipa\_line\_mark\_config描述见下表：

**表 3-510. 函数 ipa\_line\_mark\_config**

函数名称	ipa_line_mark_config
函数原型	void ipa_line_mark_config(uint16_t line_num);
功能描述	配置IPA行标记
先决条件	-
被调用函数	-
输入参数{in}	
line_num	行数目
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure line mark */
ipa_line_mark_config(10);
```

### 函数 ipa\_inter\_timer\_config

函数ipa\_inter\_timer\_config描述见下表：

**表 3-511. 函数 ipa\_inter\_timer\_config**

函数名称	ipa_inter_timer_config
函数原型	void ipa_inter_timer_config(uint8_t timer_cfg);
功能描述	配置IPA内部定时器使能或禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_cfg	内部定时器配置
IPA_INTER_TIMER_ENABLE	使能内部定时器
IPA_INTER_TIMER_DISABLE	禁能内部定时器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable inter-timer */
ipa_inter_timer_config(IPA_INTER_TIMER_ENABLE);
```

### 函数 ipa\_interval\_clock\_num\_config

函数ipa\_interval\_clock\_num\_config描述见下表：

**表 3-512. 函数 ipa\_interval\_clock\_num\_config**

函数名称	ipa_interval_clock_num_config
函数原型	void ipa_interval_clock_num_config(uint8_t clk_num);
功能描述	配置间隔时钟周期数
先决条件	-
被调用函数	-
输入参数{in}	

<b>clk_num</b>	间隔时钟周期数
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the number of clock cycles interval and it has no meaning if ITEN is '0' */

ipa_interval_clock_num_config(1);
```

### 函数 ipa\_flag\_get

函数ipa\_flag\_get描述见下表：

表 3-513. 函数

<b>函数名称</b>	ipa_flag_get
<b>函数原型</b>	FlagStatus ipa_flag_get(uint32_t flag);
<b>功能描述</b>	从IPA_INTF寄存器获取IPA标志位状态
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>flag</b>	IPA标志
<i>IPA_FLAG_TAE</i>	传输访问错误中断标志
<i>IPA_FLAG_FTF</i>	传输完成中断标志
<i>IPA_FLAG_TLM</i>	传输行标记中断标志
<i>IPA_FLAG_LAC</i>	LUT访问冲突中断标志位
<i>IPA_FLAG_LL</i>	LUT加载完成中断标志LUT
<i>IPA_FLAG_WCF</i>	配置错误中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET或RESET

例如：

```
/* wait for full transfer finish flag set */

while(ipa_flag_get(IPA_FLAG_FTF) == RESET);
```

### 函数 ipa\_flag\_clear

函数ipa\_flag\_clear描述见下表：

表 3-514. 函数

<b>函数名称</b>	ipa_flag_clear
-------------	----------------

函数原型	void ipa_flag_clear(uint32_t flag);
功能描述	清除IPA标志位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>flag</b>	IPA标志
<i>IPA_FLAG_TAE</i>	传输访问错误中断标志
<i>IPA_FLAG_FTF</i>	传输完成中断标志
<i>IPA_FLAG_TLM</i>	传输行标记中断标志
<i>IPA_FLAG_LAC</i>	LUT访问冲突中断标志位
<i>IPA_FLAG_LLFB</i>	LUT加载完成中断标志LUT
<i>IPA_FLAG_WCF</i>	配置错误中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

Example:

```
/* wait for full transfer finish flag set */

while(ipa_flag_get(IPA_FLAG_FTF) == RESET);

/* clear full transfer finish flag */

ipa_flag_clear(IPA_FLAG_FTF);
```

### 函数 ipa\_interrupt\_enable

函数ipa\_interrupt\_enable描述见下表：

表 3-515. 函数 ipa\_interrupt\_enable

函数名称	ipa_interrupt_enable
函数原型	void ipa_interrupt_enable(uint32_t int_flag);
功能描述	使能IPA中断
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>int_flag</b>	IPA中断标志位
<i>IPA_INT_TAE</i>	传输访问错误中断标志
<i>IPA_INT_FTF</i>	传输完成中断标志
<i>IPA_INT_TLM</i>	传输行标记中断标志
<i>IPA_INT_LAC</i>	LUT访问冲突中断标志位
<i>IPA_INT_LLFB</i>	LUT加载完成中断标志LUT

<i>IPA_INT_WCF</i>	配置错误中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable full transfer finish interrupt */

ipa_interrupt_enable(IPA_INT_FTF);
```

### 函数 ipa\_interrupt\_disable

函数ipa\_interrupt\_disable描述见下表：

**表 3-516. 函数 ipa\_interrupt\_disable**

函数名称	ipa_interrupt_disable
函数原型	void ipa_interrupt_disable(uint32_t int_flag);
功能描述	禁能IPA中断
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<i>int_flag</i>	IPA中断标志位
<i>IPA_INT_TAE</i>	传输访问错误中断标志
<i>IPA_INT_FTF</i>	传输完成中断标志
<i>IPA_INT_TLM</i>	传输行标记中断标志
<i>IPA_INT_LAC</i>	LUT访问冲突中断标志位
<i>IPA_INT_LLFI</i>	LUT加载完成中断标志LUT
<i>IPA_INT_WCF</i>	配置错误中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable full transfer finish interrupt */

ipa_interrupt_disable(IPA_INT_FTF);
```

### 函数 ipa\_interrupt\_flag\_get

函数ipa\_interrupt\_flag\_get描述见下表：

**表 3-517. 函数 ipa\_interrupt\_flag\_get**

函数名称	ipa_interrupt_flag_get
------	------------------------

<b>函数原型</b>	FlagStatus ipa_interrupt_flag_get(uint32_t int_flag);
<b>功能描述</b>	获取IPA中断标志位
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>int_flag</b>	IPA中断标志位
<i>IPA_INT_FLAG_TAE</i>	传输访问错误中断标志
<i>IPA_INT_FLAG_FTF</i>	传输完成中断标志
<i>IPA_INT_FLAG_TLM</i>	传输行标记中断标志
<i>IPA_INT_FLAG_LAC</i>	LUT访问冲突中断标志位
<i>IPA_INT_FLAG_LLFI</i>	LUT加载完成中断标志LUT
<i>IPA_INT_FLAG_WCF</i>	配置错误中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET或RESET

例如：

```
/* check whether full transfer finish interrupt flag is SET */

while(ipa_interrupt_flag_get(IPA_INT_FLAG_FTF) == RESET);
```

### 函数 ipa\_interrupt\_flag\_clear

函数ipa\_interrupt\_flag\_clear描述见下表：

**表 3-518. 函数 ipa\_interrupt\_flag\_clear**

<b>函数名称</b>	ipa_interrupt_flag_clear
<b>函数原型</b>	void ipa_interrupt_flag_clear(uint32_t int_flag);
<b>功能描述</b>	清除IPA中断标志位
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>int_flag</b>	IPA中断标志位
<i>IPA_INT_FLAG_TAE</i>	传输访问错误中断标志
<i>IPA_INT_FLAG_FTF</i>	传输完成中断标志
<i>IPA_INT_FLAG_TLM</i>	传输行标记中断标志
<i>IPA_INT_FLAG_LAC</i>	LUT访问冲突中断标志位
<i>IPA_INT_FLAG_LLFI</i>	LUT加载完成中断标志LUT
<i>IPA_INT_FLAG_WCF</i>	配置错误中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
if(ipa_interrupt_flag_get(IPA_INT_FLAG_FTF) != RESET){
    /* clear full transfer finish interrupt flag */
    ipa_interrupt_flag_clear(IPA_INT_FLAG_FTF);
}
```

## 3.18. IREF

IREF 是对可编程电流源（IREF）操作的软件包。章节外设寄存器说明 [3.18.1](#) 描述了 IREF 的寄存器列表，章节 [3.18.2](#) 对 IREF 库函数进行说明

### 3.18.1. 外设寄存器说明

**表 3-519. IREF 寄存器**

寄存器名称	寄存器描述
IREF_CTL	控制寄存器

### 3.18.2. 外设库函数说明

IREF 库函数列表如下表所示：

**表 3-520. IREF 库函数**

库函数名称	库函数描述
iref_deinit	复位外设IREF
iref_enable	使能IREF
iref_disable	失能IREF
iref_mode_set	设置IREF模式
iref_sink_set	设置IREF源电流/灌电流模式
iref_precision_trim_value_set	设置电流校准值
iref_step_data_config	设置IREF电流值

#### 函数 iref\_deinit

函数iref\_deinit描述见下表：

**表 3-521. 函数 iref\_deinit**

函数名称	iref_deinit
函数原型	void iref_deinit (void);
功能描述	复位外设IREF
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 IREF */
iref_deinit();
```

### 函数 **iref\_enable**

函数**iref\_enable**描述见下表：

**表 3-522. 函数 **iref\_enable****

函数名称	iref_enable
函数原型	void iref_enable (void);
功能描述	使能外设IREF
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 IREF */
iref_enable();
```

### 函数 **iref\_disable**

函数**iref\_disable**描述见下表：

**表 3-523. 函数 **iref\_disable****

函数名称	iref_disable
函数原型	void iref_disable(void);
功能描述	失能外设IREF
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 IREF */
iref_disable();
```

### 函数 iref\_mode\_set

函数iref\_mode\_set描述见下表：

**表 3-524. 函数 iref\_mode\_set**

函数名称	iref_mode_set
函数原型	void iref_mode_set(uint32_t step);
功能描述	设置IREF模式
先决条件	-
被调用函数	-
输入参数{in}	
step	IREF模式
IREF_MODE_LOW_POWER	低功耗模式，电流步长为1uA
IREF_MODE_HIGH_CURRENT	大电流模式，电流步长为8uA step
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set IREF mode */
iref_mode_set(IREF_MODE_LOW_POWER);
```

### 函数 iref\_sink\_set

函数iref\_sink\_set描述见下表：

**表 3-525. 函数 iref\_sink\_set**

函数名称	iref_sink_set
函数原型	void iref_sink_set(uint32_t sinkmode);
功能描述	设置IREF源电流/灌电流模式
先决条件	-

<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>sinkmode</b>	电流模式
<i>IREF_SOURCE_CU RRENT</i>	源电流模式
<i>IREF_SINK_CURR ENT</i>	灌电流模式
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* set IREF source current mode */

iref_sink_set(IREF_SOURCE_CURRENT);
```

#### 函数 **iref\_precision\_trim\_value\_set**

函数 **iref\_precision\_trim\_value\_set** 描述见下表：

**表 3-526. 函数 **iref\_precision\_trim\_value\_set****

<b>函数名称</b>	iref_precision_trim_value_set
<b>函数原型</b>	void iref_precision_trim_value_set (uint32_t precisiontrim);
<b>功能描述</b>	设置电流校准值
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>precisiontrim</b>	电流校准值
<i>IREF_CUR_PRECI SION_TRIM_X</i>	X=0..31 校准值-15%..+16%
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* set IREF precision trim value -15% */

iref_precision_trim_value_set(IREF_CUR_PRECISION_TRIM_0);
```

#### 函数 **iref\_step\_data\_config**

函数 **iref\_step\_data\_config** 描述见下表：

表 3-527 函数 iref\_step\_data\_config

函数名称	iref_step_data_config
函数原型	void iref_step_data_config (uint32_t stepdata);
功能描述	设置电流步数
先决条件	-
被调用函数	-
输入参数{in}	
stepdata	步进值
IREF_CUR_STEP_DATA_X	X=0..63 对应0-63步
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set IREF step data */

iref_step_data_config(IREF_CUR_STEP_DATA_0);
```

## 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 3-528. 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
IP <sup>(1)</sup>	中断优先级寄存器
STIR <sup>(1)</sup>	软触发中断寄存器
CPUID <sup>(2)</sup>	CPUID寄存器

寄存器名称	寄存器描述
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
CFSR <sup>(2)</sup>	配置错误状态寄存器
HFSR <sup>(2)</sup>	硬错误状态寄存器
DFSR <sup>(2)</sup>	调试错误状态寄存器
MMFAR <sup>(2)</sup>	存储管理错误地址寄存器
BFAR <sup>(2)</sup>	总线错误地址寄存器
AFSR <sup>(2)</sup>	辅助错误地址寄存器
PFR <sup>(2)</sup>	处理器特性寄存器
DFR <sup>(2)</sup>	调试特性寄存器
ADR <sup>(2)</sup>	辅助特性寄存器
MMFR <sup>(2)</sup>	存储模型特性寄存器
ISAR <sup>(2)</sup>	指令设置属性寄存器
CPACR <sup>(2)</sup>	协处理器访问控制寄存器

1. 参考 core\_cm4.h 文件中定义的结构体类型 NVIC\_Type

2. 参考 core\_cm4.h 文件中定义的结构体类型 SCB\_Type

**表 3-529. 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\_cm4.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 3-530. 枚举类型 IRQn\_Type**

成员名称	功能描述
WWDGT_IRQHandler	窗口看门狗中断
LVD_IRQHandler	连接到 EXTI 线的 LVD 中断
TAMPER_STAMP_IRQHandler	侵入和时间戳检测中断
RTC_WKUP_IRQHandler	RTC 唤醒中断
FMC_IRQHandler	FMC 全局中断
RCU_CTC_IRQHandler	RCU 和 CTC 中断
EXTI0_IRQHandler	EXTI 线 0 中断
EXTI1_IRQHandler	EXTI 线 1 中断
EXTI2_IRQHandler	EXTI 线 2 中断
EXTI3_IRQHandler	EXTI 线 3 中断
EXTI4_IRQHandler	EXTI 线 4 中断
DMA0_Channel0_IRQHandler	DMA0 通道 0 全局中断
DMA0_Channel1_IRQHandler	DMA0 通道 1 全局中断
DMA0_Channel2_IRQHandler	DMA0 通道 2 全局中断
DMA0_Channel3_IRQHandler	DMA0 通道 3 全局中断
DMA0_Channel4_IRQHandler	DMA0 通道 4 全局中断
DMA0_Channel5_IRQHandler	DMA0 通道 5 全局中断
DMA0_Channel6_IRQHandler	DMA0 通道 6 全局中断
ADC_IRQHandler	ADC 全局中断
CAN0_TX_IRQHandler	CAN0 发送中断
CAN0_RX0_IRQHandler	CAN0 接收 0 中断
CAN0_RX1_IRQHandler	CAN0 接收 1 中断
CAN0_EWMC_IRQHandler	CAN0 EWMC 中断
EXTI5_9_IRQHandler	EXTI 线[9:5] 中断
TIMER0_BRK_TIMER8_IRQHandler	TIMER0 中止中断和 TIMER8 全局中断

TIMER0_UP_TIME_R9_IRQHandler	TIMER0 更新中断和 TIMER9 全局中断
TIMER0_TRG_CMT_TIMER10_IRQHandler	TIMER0 触发与通道换相中断和 TIMER10 全局中断
TIMER0_Channel_IRQHandler	TIMER0 通道捕获比较中断
TIMER1_IRQHandler	TIMER1 全局中断
TIMER2_IRQHandler	TIMER2 全局中断
TIMER3_IRQHandler	TIMER3 全局中断
I2C0_EV_IRQHandler	I2C0 事件中断
I2C0_ER_IRQHandler	I2C0 错误中断
I2C1_EV_IRQHandler	I2C1 事件中断
I2C1_ER_IRQHandler	I2C1 错误中断
SPI0_IRQHandler	SPI0 全局中断
SPI1_IRQHandler	SPI1 全局中断
USART0_IRQHandler	USART0 全局中断
USART1_IRQHandler	USART1 全局中断
USART2_IRQHandler	USART2 全局中断
EXTI10_15_IRQHandler	EXTI 线[15:10] 中断
RTC_Alarm_IRQHandler	连接 EXTI 线的 RTC 闹钟中断
USBFS_WKUP_IRQHandler	连接 EXTI 线的 USBFS 唤醒中断
TIMER7_BRK_TIM_ER11_IRQHandler	TIMER7 中止中断和 TIMER11 全局中断
TIMER7_UP_TIME_R12_IRQHandler	TIMER7 更新中断和 TIMER12 全局中断
TIMER7_TRG_CMT_TIMER13_IRQHandler	TIMER7 触发与通道换相中断和 TIMER13 全局中断
TIMER7_Channel_IRQHandler	TIMER7 通道捕获比较中断
DMA0_Channel7_IRQHandler	DMA0 通道 7 全局中断
EXMC_IRQHandler	EXMC 全局中断
SDIO_IRQHandler	SDIO 全局中断
TIMER4_IRQHandler	TIMER4 全局中断
SPI2_IRQHandler	SPI2 全局中断
UART3_IRQHandler	UART3 全局中断
UART4_IRQHandler	UART4 全局中断
TIMER5_DAC_IRQHandler	TIMER5 和 DAC 全局中断
TIMER6_IRQHandler	TIMER6 全局中断

DMA1_Channel0_IRQHandler	DMA1 通道 0 全局中断
DMA1_Channel1_IRQHandler	DMA1 通道 1 全局中断
DMA1_Channel2_IRQHandler	DMA1 通道 2 全局中断
DMA1_Channel3_IRQHandler	DMA1 通道 3 全局中断
DMA1_Channel4_IRQHandler	DMA1 通道 4 全局中断
ENET_IRQHandler	ENET 全局中断
ENET_WKUP_IRQHandler	ENET 唤醒中断
CAN1_TX_IRQHandler	CAN1 发送中断
CAN1_RX0_IRQHandler	CAN1 接收 0 中断
CAN1_RX1_IRQHandler	CAN1 接收 1 中断
CAN1_EWMC_IRQHandler	CAN1 EWMC 中断
USBFS_IRQHandler	USBFS 全局中断
DMA1_Channel5_IRQHandler	DMA1 通道 5 全局中断
DMA1_Channel6_IRQHandler	DMA1 通道 6 全局中断
DMA1_Channel7_IRQHandler	DMA1 通道 7 全局中断
USART5_IRQHandler	USART5 全局中断
I2C2_EV_IRQHandler	I2C2 事件中断
I2C2_ER_IRQHandler	I2C2 错误中断
USBHS_EP1_Out_IRQHandler	USBHS 端点 1 输出中断
USBHS_EP1_In_IRQHandler	USBHS 端点 1 输入中断
USBHS_WKUP_IRQHandler	USBHS 唤醒中断
USBHS_IRQHandler	USBHS 中断
DCI_IRQHandler	DCI 中断
TRNG_IRQHandler	TRNG 中断
FPU_IRQHandler	FPU 中断
UART6_IRQHandler	UART6 全局中断
UART7_IRQHandler	UART7 全局中断
SPI3_IRQHandler	SPI3 全局中断
SPI4_IRQHandler	SPI4 全局中断
SPI5_IRQHandler	SPI5 全局中断

TLI_IRQn	TLI 全局中断
TLI_ER_IRQn	TLI 错误全局中断
IPA_IRQn	IPA 全局中断

MISC库函数列表如下表所示:

**表 3-531. MISC 库函数**

库函数名称	库函数描述
nvic_priority_group_set	设置优先级组
nvic_irq_enable	使能NVIC的中断
nvic_irq_disable	禁能NVIC的中断
nvic_vector_table_set	设置向量表地址
system_lowpower_set	设置系统低功耗模式状态
system_lowpower_reset	复位系统低功耗模式状态
systick_clksource_set	设置系统定时器时钟源

### 函数 nvic\_priority\_group\_set

函数nvic\_priority\_group\_set描述见下表:

**表 3-532. 函数 nvic\_priority\_group\_set**

函数名称	nvic_priority_group_set
函数原形	void nvic_priority_group_set(uint32_t nvic_prigroup);
功能描述	配置优先级组的位长度
先决条件	-
被调用函数	-
输入参数{in}	
<b>nvic_prigroup</b>	优先级组
<b>NVIC_PRIGROUP_PRE0_SUB4</b>	0位用于抢占优先级, 4位用于响应优先级
<b>NVIC_PRIGROUP_PRE1_SUB3</b>	1位用于抢占优先级, 3位用于响应优先级
<b>NVIC_PRIGROUP_PRE2_SUB2</b>	2位用于抢占优先级, 2位用于响应优先级
<b>NVIC_PRIGROUP_</b>	3位用于抢占优先级, 1位用于响应优先级

<i>PRE3_SUB1</i>	
<i>NVIC_PRIGROUP_PRE4_SUB0</i>	4位用于抢占优先级，0位用于响应优先级
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-533. 函数 nvic\_irq\_enable**

函数名称	<b>nvic_irq_enable</b>
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	<code>nvic_priority_group_set</code>
<b>输入参数{in}</b>	
<b>nvic_irq</b>	NVIC中断，参考枚举类型 <a href="#">表 3-530. 枚举类型IRQn_Type</a>
<b>输入参数{in}</b>	
<b>nvic_irq_pre_priority</b>	抢占优先级（0~4）
<b>输入参数{in}</b>	
<b>nvic_irq_sub_priority</b>	响应优先级（0~4）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	

--	--

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 3-534. 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, 参考枚举类型 <a href="#">表 3-530. 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\_vector\_table\_set**

函数nvic\_vector\_table\_set描述见下表：

**表 3-535. 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_vict_tab, uint32_t offset);
功能描述	设置向量表地址
先决条件	-

<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>nvic_vict_tab</b>	RAM 或者 FLASH基地址
<b>NVIC_VECTTAB_RAM</b>	RAM 基地址
<b>NVIC_VECTTAB_FLASH</b>	FLASH基地址
<b>输入参数{in}</b>	
<b>offset</b>	向量表偏移量（向量表地址=基地址+偏移量）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-536. 函数 system\_lowpower\_set**

<b>函数名称</b>	system_lowpower_set
<b>函数原形</b>	void system_lowpower_set(uint8_t lowpower_mode);
<b>功能描述</b>	系统低功耗模式状态的管理
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>lowpower_mode</b>	系统低功耗模式的状态
<b>SCB_LPM_SLEEP_EXIT_ISR</b>	该位为1时，退出ISR时一直处于低功耗模式
<b>SCB_LPM_DEEPSL</b>	该位为1时，系统处于deep sleep模式

<i>EEP</i>	
<i>SCB_LPM_WAKE_BY_ALL_INT</i>	该位为1时，低功耗模式可以被所有中断唤醒（无论中断是否被使能）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-537. 函数 system\_lowpower\_reset**

<b>函数名称</b>	system_lowpower_reset
<b>函数原形</b>	void system_lowpower_reset(uint8_t lowpower_mode);
<b>功能描述</b>	系统低功耗模式状态的管理
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>lowpower_mode</b>	系统低功耗模式的状态
<i>SCB_LPM_SLEEP_EXIT_ISR</i>	该位为0时，退出ISR时退出低功耗模式
<i>SCB_LPM_DEEPSLEEP_EEP</i>	该位为0时，系统进入sleep模式
<i>SCB_LPM_WAKE_BY_ALL_INT</i>	该位为0时，系统只能被使能的中断唤醒
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-538. 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}	
<b>systick_clksource</b>	SysTick时钟源
<b>SYSTICK_CLKSOU RCE_HCLK</b>	SysTick时钟源为AHB时钟
<b>SYSTICK_CLKSOU RCE_HCLK_DIV8</b>	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三种省电模式，包括睡眠模式，深度睡眠模式和待机模式。章节 [3.20.1](#) 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 3-539. PMU 寄存器**

寄存器名称	寄存器描述
PMU_CTL	控制寄存器
PMU_CS	电源控制和状态寄存器

### 3.20.2. 外设库函数说明

PMU 库函数列表如下表所示：

**表 3-540. 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_ldo_output_select	LDO输出电压选择
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_low_driver_mode_enable	深度睡眠下使能低驱动模式
pmu_lowdriver_lowpower_config	使用低功耗LDO时低驱动模式配置
pmu_lowdriver_normalpower_config	使用正常功耗LDO时低驱动模式配置
pmu_to_sleepmode	进入睡眠模式
pmu_to_deepsleepmode	进入深度睡眠模式
pmu_to_standbymode	进入待机模式
pmu_wakeup_pin_enable	WKUP引脚唤醒使能
pmu_wakeup_pin_disable	WKUP引脚唤醒失能
pmu_backup_ldo_config	备份SRAM LDO使能
pmu_backup_write_enable	备份域写使能
pmu_backup_write_disable	备份域写失能

库函数名称	库函数描述
pmu_flag_reset	复位标志位
pmu_flag_get	获取标志位

### 函数 pmu\_deinit

函数pmu\_deinit描述见下表：

表 3-541. 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 3-542. 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	-

<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>lvdt_n</b>	电压阈值
<i>PMU_LVDT_0</i>	电压阈值为2.2V
<i>PMU_LVDT_1</i>	电压阈值为2.3V
<i>PMU_LVDT_2</i>	电压阈值为2.4V
<i>PMU_LVDT_3</i>	电压阈值为2.5V
<i>PMU_LVDT_4</i>	电压阈值为2.6V
<i>PMU_LVDT_5</i>	电压阈值为2.7V
<i>PMU_LVDT_6</i>	电压阈值为2.8V
<i>PMU_LVDT_7</i>	电压阈值为2.9V
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* select low voltage detector threshold as 2.9V */
pmu_lvd_select (PMU_LVDT_7);
```

#### 函数 **pmu\_ldo\_output\_select**

函数pmu\_ldo\_output\_select描述见下表：

**表 3-543. 函数 pmu\_ldo\_output\_select**

<b>函数名称</b>	pmu_ldo_output_select
<b>函数原型</b>	void pmu_ldo_output_select(uint32_t ldo_output);
<b>功能描述</b>	内部电压调节器（LDO）输出电压选择
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>ldo_output</b>	输出电压模式

<i>PMU_LDOVS_LOW</i>	输出低电压模式
<i>PMU_LDOVS_MID</i>	输出中电压模式
<i>PMU_LDOVS_HIG H</i>	输出高电压模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select output low voltage mode */

pmu_ldo_output_select (PMU_LDOVS_LOW);
```

#### 函数 **pmu\_low\_driver\_mode\_enable**

函数pmu\_low\_driver\_mode\_enable描述见下表：

**表 3-544. 函数 pmu\_low\_driver\_mode\_enable**

函数名称	pmu_low_driver_mode_enable
函数原型	void pmu_low_driver_mode_enable (uint32_t lowdr_mode);
功能描述	内部电压调节器（LDO）输出电压选择
先决条件	-
被调用函数	-
输入参数{in}	
<b>lowdr_mode</b>	低驱动模式使能或失能
<i>PMU_LOWDRIVER_ENABLE</i>	低电压模式使能
<i>PMU_LOWDRIVER_DISABLE</i>	低电压模式失能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable low-driver mode */
pmu_low_driver_mode_enable (PMU_LOWDRIVER_ENABLE);
```

### **函数 pmu\_highdriver\_switch\_select**

函数pmu\_highdriver\_switch\_select描述见下表：

**表 3-545. 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	高驱动模式切换器
PMU_HIGHDR_SWITC_H_NONE	失能高驱动模式切换器
PMU_HIGHDR_SWITC_H_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 3-546. 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 3-547. 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_lvd_disable`

函数`pmu_lvd_disable`描述见下表:

**表 3-548. 函数 `pmu_lvd_disable`**

函数名称	<code>pmu_lvd_disable</code>
函数原型	<code>void pmu_lvd_disable (void);</code>
功能描述	关闭低压检测器
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* disable PMU lvd */
pmu_lvd_disable();
```

### 函数 `pmu_lowdriver_lowpower_config`

函数`pmu_lowdriver_lowpower_config`描述见下表:

**表 3-549. 函数 `pmu_lowdriver_lowpower_config`**

函数名称	<code>pmu_lowdriver_lowpower_config</code>
函数原型	<code>void pmu_lowdriver_lowpower_config(uint32_t mode);</code>
功能描述	使用低驱动LDO时低驱动模式配置
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>mode</b>	驱动模式

<i>PMU_NORMALDR_LOWPWR</i>	使用低驱动LDO时工作在正常驱动模式
<i>PMU_LOWDR_LOWPWR</i>	使用低驱动LDO且LDEN为1时低驱动模式使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* low-driver mode when use low power LDO */
pmu_lowdriver_lowpower_config (PMU_NORMALDR_LOWPWR);
```

#### 函数 pmu\_lowdriver\_normalpower\_config

函数pmu\_lowdriver\_normalpower\_config描述见下表：

表 3-550. 函数 pmu\_lowdriver\_normalpower\_config

函数名称	pmu_lowdriver_normalpower_config
函数原型	void pmu_lowdriver_normalpower_config(uint32_t mode);
功能描述	使用正常驱动LDO时低驱动模式配置
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>mode</b>	驱动模式
<i>PMU_NORMALDR_NORMALPWR</i>	使用正常驱动LDO时工作在正常驱动模式
<i>PMU_NORMALDR_NORMALPWR</i>	使用正常驱动LDO且LDEN为1时时低驱动驱动模式使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* low-driver mode when use low power LDO */

pmu_lowdriver_lowpower_config(PMU_NORMALDR_LOWPWR)
```

### **函数 pmu\_to\_sleepmode**

函数pmu\_to\_sleepmode描述见下表：

**表 3-551. 函数 pmu\_to\_sleepmode**

函数名称	pmu_to_sleepmode
函数原型	void pmu_to_sleepmode(uint8_t sleepmodecmd);
功能描述	进入睡眠模式
先决条件	-
被调用函数	-
输入参数{in}	
sleepmodecmd	进入睡眠模式命令
WFI_CMD	WFI命令
WFE_CMD	WFE命令
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PMU work at sleep mode */

pmu_to_sleepmode(WFI_CMD);
```

### **函数 pmu\_to\_deepsleepmode**

函数pmu\_to\_deepsleepmode描述见下表：

**表 3-552. 函数 pmu\_to\_deepsleepmode**

函数名称	pmu_to_deepsleepmode
函数原型	void pmu_to_deepsleepmode(uint32_t ldo,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}	
<b>ldo</b>	LDO工作模式
<i>PMU_LDO_NORMAL</i>	当系统进入深度睡眠模式时，LDO仍正常工作
<i>PMU_LDO_LOWPOWER</i>	当系统进入深度睡眠模式时，LDO进入低功耗模式
输入参数{in}	
<b>deepsleepmodecmd</b>	进入深度睡眠模式命令
<i>WFI_CMD</i>	WFI命令
<i>WFE_CMD</i>	WFE命令
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PMU work at deepsleep mode */
pmu_to_deepsleepmode (PMU_LDO_NORMAL, WFI_CMD);
```

### 函数 **pmu\_to\_standbymode**

函数pmu\_to\_standbymode描述见下表：

**表 3-553. 函数 pmu\_to\_standbymode**

函数名称	pmu_to_standbymode
函数原型	void pmu_to_standbymode(uint8_t standbymodecmd);
功能描述	进入待机模式
先决条件	-
被调用函数	-
输入参数{in}	

<b>standbymodecmd</b>	进入待机模式命令
<i>WFI_CMD</i>	WFI命令
<i>WFE_CMD</i>	WFE命令
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* PMU work at standby mode */
pmu_to_standbymode (WFI_CMD);
```

#### 函数 **pmu\_backup\_ldo\_config**

函数pmu\_backup\_ldo\_config描述见下表：

**表 3-554. 函数 pmu\_backup\_ldo\_config**

<b>函数名称</b>	pmu_backup_ldo_config
<b>函数原型</b>	void pmu_backup_ldo_config(uint32_t bkp_ldo);
<b>功能描述</b>	开启备份SRAM LDO
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>bkp_ldo</b>	备份SRAM开启或关闭
<i>PMU_BLDOON_OF_F</i>	备份SRAM LDO关闭
<i>PMU_BLDOON_ON</i>	备份SRAM LDO开启
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

---

```
/* backup SRAM LDO on */
pmu_backup_ldo_config (PMU_BLDOON_ON);
```

### 函数 pmu\_wakeup\_pin\_enable

函数pmu\_wakeup\_pin\_enable描述见下表:

**表 3-555. 函数 pmu\_wakeup\_pin\_enable**

函数名称	pmu_wakeup_pin_enable
函数原型	void pmu_wakeup_pin_enable(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable wakeup pin */
pmu_wakeup_pin_enable();
```

### 函数 pmu\_wakeup\_pin\_disable

函数pmu\_wakeup\_pin\_disable描述见下表:

**表 3-556. 函数 pmu\_wakeup\_pin\_disable**

函数名称	pmu_wakeup_pin_disable
函数原型	void pmu_wakeup_pin_disable (void);
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}	

-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable wakeup pin */

pmu_wakeup_pin_disable();
```

#### 函数 **pmu\_backup\_write\_enable**

函数pmu\_backup\_write\_enable描述见下表：

表 3-557. 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 3-558. 函数 pmu\_backup\_write\_disable

函数名称	pmu_backup_write_disable
函数原型	void pmu_backup_write_disable (void);
功能描述	备份域写失能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable backup domain write */
pmu_backup_write_disable();
```

### 函数 pmu\_flag\_get

函数pmu\_flag\_get描述见下表：

表 3-559. 函数 pmu\_flag\_get

函数名称	pmu_flag_get
函数原型	FlagStatus pmu_flag_get(uint32_t pmu_flag);
功能描述	获取标志位
先决条件	-
被调用函数	-
输入参数{in}	
pmu_flag	标志位
PMU_FLAG_WAKE_UP	唤醒标志
PMU_FLAG_STANDBY	待机标志

<i>PMU_FLAG_LVD</i>	低电压状态标志
<i>PMU_FLAG_BLDO_RF</i>	备份SRAM LDO就绪标志
<i>PMU_FLAG_LDOV_SRF</i>	LDO电压选择就绪标志
<i>PMU_FLAG_HDRF</i>	高驱动就绪标志
<i>PMU_FLAG_HDSRF</i>	高驱动切换器就绪标志
<i>PMU_FLAG_LDRF</i>	低驱动模式就绪标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET或RESET

例如：

```
/* get flag state */

FlagStatus status;

status = pmu_flag_get (PMU_FLAG_WAKEUP);
```

#### 函数 **pmu\_flag\_reset**

函数pmu\_flag\_reset描述见下表：

**表 3-560. 函数 pmu\_flag\_reset**

函数名称	pmu_flag_reset
函数原型	void pmu_flag_reset(uint32_t flag_reset);
功能描述	复位标志位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>flag_reset</b>	标志位
<i>PMU_FLAG_RESET_WAKEUP</i>	复位唤醒标志

<i>PMU_FLAG_RESET_STANDBY</i>	复位机标志
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 3-561. RCU 寄存器

寄存器名称	寄存器描述
RCU_CTL	控制寄存器
RCU_PLL	锁相环寄存器
RCU_CFG0	时钟配置寄存器0
RCU_INT	时钟中断寄存器
RCU_AHB1RST	AHB1复位寄存器
RCU_AHB2RST	AHB2复位寄存器
RCU_AHB3RST	AHB3复位寄存器
RCU_APB1RST	APB1复位寄存器
RCU_APB2RST	APB2复位寄存器
RCU_AHB1EN	AHB1使能寄存器
RCU_AHB2EN	AHB2使能寄存器

寄存器名称	寄存器描述
RCU_AHB3EN	AHB3使能寄存器
RCU_APB1EN	APB1使能寄存器
RCU_APB2EN	APB2使能寄存器
RCU_AHB1SPEN	AHB1睡眠模式使能寄存器
RCU_AHB2SPEN	AHB2睡眠模式使能寄存器
RCU_AHB3SPEN	AHB3睡眠模式使能寄存器
RCU_APB1SPEN	APB1睡眠模式使能寄存器
RCU_APB2SPEN	APB2睡眠模式使能寄存器
RCU_BDCTL	备份域控制寄存器
RCU_RSTSCK	复位源/时钟寄存器
RCU_PLLSSCTL	PLL 时钟扩频控制寄存器
RCU_PLLI2S	PLLI2S 寄存器
RCU_PLLSAI	PLLSAI 寄存器
RCU_CFG1	配置寄存器1
RCU_ADDCTL	附加控制寄存器
RCU_ADDINT	附加中断寄存器
RCU_ADDAPB1RS T	APB1附加复位寄存器
RCU_ADDAPB1EN	APB1附加使能寄存器
RCU_ADDAPB1SP EN	APB1睡眠模式附加使能寄存器
RCU_VKEY	电源解锁寄存器
RCU_DSV	深度睡眠模式电压寄存器

### 3.21.2. 外设库函数说明

RCU库函数列表如下表所示：

**表 3-562. 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	除能外设时钟
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，除能外设时钟
rcu_periph_reset_enable	使能外设复位
rcu_periph_reset_disable	除能外设复位
rcu_bkp_reset_enable	使能BKP复位
rcu_bkp_reset_disable	除能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_ckout1_config	配置CKOUT1时钟源选择
rcu_pll_config	配置主PLL时钟
rcu_plli2s_config	配置PLLI2S时钟
rcu_pll sai config	配置PLLSAI时钟
rcu_rtc_clock_config	配置RTC时钟源选择
rcu_i2s_clock_config	配置I2S时钟源选择
rcu_ck48m_clock_config	配置CK48M时钟源选择
rcu_pll48m_clock_config	配置PLL48M时钟源选择
rcu_timer_clock_prescaler_config	配置TIMER时钟预分频
rcu_tli_clock_div_config	配置TLI时钟分频系数
rcu_flag_get	获取时钟稳定和外设复位标志

库函数名称	库函数描述
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	除能时钟稳定中断
rcu_lxtal_drive_capability_config	配置LXTAL驱动能力
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	除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	除能HXTAL时钟监视器
rcu_irc16m_adjust_value_set	设置内部16MHz RC振荡器时钟调整值
rcu_spread_spectrum_config	为主PLL时钟配置扩频调制
rcu_spread_spectrum_enable	使能扩频调制
rcu_spread_spectrum_disable	禁能扩频调制
rcu_deepsleep_voltage_set	设置深度睡眠模式电压值
rcu_voltage_key_unlock	解锁电源寄存器
rcu_clock_freq_get	获取系统时钟、总线频率

### 函数 `rcu_deinit`

函数`rcu_deinit`描述见下表：

**表 3-563. 函数 `rcu_deinit`**

函数名称	<code>rcu_deinit</code>
函数原形	<code>void rcu_deinit(void);</code>
功能描述	复位RCU，将RCU所有寄存器的值复位成初始值
先决条件	-

被调用函数	rcu_osc1_stab_wait
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 3-564. 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}	
<i>periph</i>	RCU外设，具体参考rcu_periph_enum
<i>RCU_GPIOx</i>	GPIOx时钟(x=A,B,C,D,E,F,G,H,I)
<i>RCU_CRC</i>	CRC时钟
<i>RCU_BKPSRAM</i>	BKPSRAM时钟
<i>RCU_TCMSRAM</i>	TCMSRAM时钟
<i>RCU_DMAMx</i>	DMAx时钟(x=0,1)
<i>RCU_IPA</i>	IPA时钟
<i>RCU_ENET</i>	以太网ENET时钟
<i>RCU_ENETTX</i>	ENETTX时钟

<i>RCU_ENETRX</i>	ENETRX时钟
<i>RCU_ENETPTP</i>	ENETPTP时钟
<i>RCU_USBHS</i>	USBHS时钟
<i>RCU_USBHSULPI</i>	USBHSULPI时钟
<i>RCU_DCI</i>	DCI时钟
<i>RCU_TRNG</i>	TRNG时钟
<i>RCU_USBFS</i>	USBFS时钟
<i>RCU_EXMC</i>	EXMC时钟
<i>RCU_TIMERx</i>	TIMERx时钟(x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGT</i>	WWDGT时钟
<i>RCU_SPIx</i>	SPIx时钟(x=0,1,2,3,4,5)
<i>RCU_USARTx</i>	USARTx时钟(x=0,1,2,5)
<i>RCU_UARTx</i>	UARTx时钟(x=3,4 ,6,7)
<i>RCU_I2Cx</i>	I2Cx时钟(x=0,1,2)
<i>RCU_CANx</i>	CANx时钟(x=0,1)
<i>RCU_PMU</i>	PMU时钟
<i>RCU_DAC</i>	DAC时钟
<i>RCU_RTC</i>	RTC时钟
<i>RCU_ADCx</i>	ADCx时钟(x=0,1,2)
<i>RCU_SDIO</i>	SDIO时钟
<i>RCU_SYSCFG</i>	SYSCFG接口时钟
<i>RCU_TLI</i>	TLI时钟
<i>RCU_CTC</i>	CTC时钟
<i>RCU_IREF</i>	IREF时钟
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-565. 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	除能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，具体参考rcu_periph_enum
RCU_GPIOx	GPIOx时钟(x=A,B,C,D,E,F,G,H,I)
RCU_CRC	CRC时钟
RCU_BKPSRAM	BKPSRAM时钟
RCU_TCMSRAM	TCMSRAM时钟
RCU_DMAX	DMAx时钟(x=0,1)
RCU_IPA	IPA时钟
RCU_ENET	以太网ENET时钟
RCU_ENETTX	ENETTX时钟
RCU_ENETRX	ENETRX时钟
RCU_ENETPTP	ENETPTP时钟
RCU_USBHS	USBHS时钟
RCU_USBHSULPI	USBHSULPI时钟
RCU_DCI	DCI时钟
RCU_TRNG	TRNG时钟
RCU_USBFS	USBFS时钟

<i>RCU_EXMC</i>	EXMC时钟
<i>RCU_TIMERx</i>	TIMERx时钟(x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGT</i>	WWDGT时钟
<i>RCU_SPIx</i>	SPIx时钟(x=0,1,2,3,4,5)
<i>RCU_USARTx</i>	USARTx时钟(x=0,1,2,5)
<i>RCU_UARTx</i>	UARTx时钟(x=3,4,6,7)
<i>RCU_I2Cx</i>	I2Cx时钟(x=0,1,2)
<i>RCU_CANx</i>	CANx时钟(x=0,1)
<i>RCU_PMU</i>	PMU时钟
<i>RCU_DAC</i>	DAC时钟
<i>RCU_RTC</i>	RTC时钟
<i>RCU_ADCx</i>	ADCx时钟(x=0,1,2)
<i>RCU_SDIO</i>	SDIO时钟
<i>RCU_SYSCFG</i>	SYSCFG接口时钟
<i>RCU_TLI</i>	TLI时钟
<i>RCU_CTC</i>	CTC时钟
<i>RCU_IREF</i>	IREF时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the USART0 clock */
rcu_periph_clock_disable(RCU_USART0);
```

#### 函数 **rcu\_periph\_clock\_sleep\_enable**

函数rcu\_periph\_clock\_sleep\_enable描述见下表：

表 3-566. 函数 **rcu\_periph\_clock\_sleep\_enable**

函数名称	rcu_periph_clock_sleep_enable
------	-------------------------------

<b>函数原形</b>	void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);
<b>功能描述</b>	在睡眠模式下，使能外设时钟
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>periph</b>	RCU外设，参考rcu_periph_sleep_enum
<i>RCU_GPIOx_SLP</i>	GPIOx时钟(x=A,B,C,D,E,F,G ,H,I)时钟
<i>RCU_CRC_SLP</i>	CRC时钟
<i>RCU_FMC_SLP</i>	FMC时钟
<i>RCU_SRAM1_SLP</i>	SRAM0时钟
<i>RCU_SRAM1_SLP</i>	SRAM1时钟
<i>RCU_BKPSRAM</i>	BKPSRAM时钟
<i>RCU_SRAM2_SLP</i>	SRAM2时钟
<i>RCU_DMAX_SLP</i>	DMAx时钟(x=0,1)
<i>RCU_IPA_SLP</i>	IPA时钟
<i>RCU_ENET_SLP</i>	以太网ENET时钟
<i>RCU_ENETTX_SLP</i>	ENETTX时钟
<i>RCU_ENETRX_SLP</i>	ENETRX时钟
<i>RCU_ENETPTP_SLP</i>	ENETPTP时钟
<i>RCU_USBHS_SLP</i>	USBHS时钟
<i>RCU_USBHSULPI_SLP</i>	USBHSULPI时钟
<i>RCU_DCI_SLP</i>	DCI时钟
<i>RCU_TRNG_SLP</i>	TRNG时钟
<i>RCU_USBFS_SLP</i>	USBFS时钟
<i>RCU_EXMC_SLP</i>	EXMC时钟
<i>RCU_TIMERx_SLP</i>	TIMERx时钟(x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)

<i>RCU_WWDGT_SLP</i>	WWDGT时钟
<i>RCU_SPIx_SLP</i>	SPIx时钟(x=0,1,2,3,4,5)
<i>RCU_USARTx_SLP</i>	USARTx时钟(x=0,1,2,5)
<i>RCU_UARTx_SLP</i>	UARTx时钟(x=3,4,6,7)
<i>RCU_I2Cx_SLP</i>	I2Cx时钟(x=0,1,2)
<i>RCU_CANx_SLP</i>	CANx时钟(x=0,1)
<i>RCU_PMU_SLP</i>	PMU时钟
<i>RCU_DAC_SLP</i>	DAC时钟
<i>RCU_RTC_SLP</i>	RTC时钟
<i>RCU_ADCx_SLP</i>	ADCx时钟(x=0,1,2)
<i>RCU_SDIO_SLP</i>	SDIO时钟
<i>RCU_SYSCFG_SLP</i>	SYSCFG接口时钟
<i>RCU_TLI_SLP</i>	TLI时钟
<i>RCU_CTC_SLP</i>	CTC时钟
<i>RCU_IREF_SLP</i>	IREF时钟
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable the CRC clock when in sleep mode */
rcu_periph_clock_sleep_enable(RCU_CRC_SLP);
```

#### 函数 **rcu\_periph\_clock\_sleep\_disable**

函数rcu\_periph\_clock\_sleep\_disable描述见下表：

**表 3-567. 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);

<b>功能描述</b>	在睡眠模式下，除能外设时钟
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>periph</b>	RCU外设，参考rcu_periph_sleep_enum
<i>RCU_GPIOx_SLP</i>	GPIOx时钟(x=A,B,C,D,E,F,G,H,I)时钟
<i>RCU_CRC_SLP</i>	CRC时钟
<i>RCU_FMC_SLP</i>	FMC时钟
<i>RCU_SRAM1_SLP</i>	SRAM0时钟
<i>RCU_SRAM1_SLP</i>	SRAM1时钟
<i>RCU_BKPSRAM</i>	BKPSRAM时钟
<i>RCU_SRAM2_SLP</i>	SRAM2时钟
<i>RCU_DMAx_SLP</i>	DMAx时钟(x=0,1)
<i>RCU_IPA_SLP</i>	IPA时钟
<i>RCU_ENET_SLP</i>	以太网ENET时钟
<i>RCU_ENETTX_SLP</i>	ENETTX时钟
<i>RCU_ENETRX_SLP</i> <i>P</i>	ENETRX时钟
<i>RCU_ENETPTP_SLP</i> <i>P</i>	ENETPTP时钟
<i>RCU_USBHS_SLP</i>	USBHS时钟
<i>RCU_USBHSULPI_SLP</i>	USBHSULPI时钟
<i>RCU_DCI_SLP</i>	DCI时钟
<i>RCU_TRNG_SLP</i>	TRNG时钟
<i>RCU_USBFS_SLP</i>	USBFS时钟
<i>RCU_EXMC_SLP</i>	EXMC时钟
<i>RCU_TIMERx_SLP</i>	TIMERx时钟(x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGT_SLP</i>	WWWDGT时钟

<i>RCU_SPIx_SLP</i>	SPIx时钟(x=0,1,2,3,4,5)
<i>RCU_USARTx_SLP</i>	USARTx时钟(x=0,1,2,5)
<i>RCU_UARTx_SLP</i>	UARTx时钟(x=3,4 ,6,7)
<i>RCU_I2Cx_SLP</i>	I2Cx时钟(x=0,1,2)
<i>RCU_CANx_SLP</i>	CANx时钟(x=0,1)
<i>RCU_PMU_SLP</i>	PMU时钟
<i>RCU_DAC_SLP</i>	DAC时钟
<i>RCU_RTC_SLP</i>	RTC时钟
<i>RCU_ADCx_SLP</i>	ADCx时钟(x=0,1,2)
<i>RCU_SDIO_SLP</i>	SDIO时钟
<i>RCU_SYSCFG_SLP<sub>P</sub></i>	SYSCFG接口时钟
<i>RCU_TLI_SLP</i>	TLI时钟
<i>RCU_CTC_SLP</i>	CTC时钟
<i>RCU_IREF_SLP</i>	IREF时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the CRC clock when in sleep mode */
rcu_periph_clock_sleep_disable(RCU_CRC_SLP);
```

#### 函数 **rcu\_periph\_reset\_enable**

函数rcu\_periph\_reset\_enable描述见下表：

**表 3-568. 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

<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>periph_reset</b>	RCU外设复位，参考rcu_periph_reset_enum
<i>RCU_GPIOxRST</i>	复位GPIOx时钟(x=A,B,C,D,E,F,G,H,I)
<i>RCU_CRCRST</i>	复位CRC时钟
<i>RCU_DMAMxRST</i>	复位DMAx时钟(x=0,1)
<i>RCU_IPARST</i>	复位IPA时钟
<i>RCU_ENETRST</i>	复用以太网时钟
<i>RCU_USBHSRST</i>	复位USBHS时钟
<i>RCU_DCIRST</i>	复位DCI时钟
<i>RCU_TRNGRST</i>	复位TRNG时钟
<i>RCU_USBFSRST</i>	复位USBFS时钟
<i>RCU_EXMCRST</i>	复位EXMC时钟
<i>RCU_TIMERxRST</i>	复位TIMERx时钟(x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGTRST</i>	复位WWDGTRST时钟
<i>RCU_SPIxRST</i>	复位SPIx时钟(x=0,1,2,3,4,5)
<i>RCU_USARTxRST</i>	复位USARTx时钟(x=0,1,2,5)
<i>RCU_UARTxRST</i>	复位UARTx时钟(x=3,4,6,7)
<i>RCU_I2CxRST</i>	复位I2Cx时钟(x=0,1,2)
<i>RCU_CANxRST</i>	复位CANx时钟(x=0,1)
<i>RCU_PMURST</i>	复位PMU时钟
<i>RCU_DACRST</i>	复位DAC时钟
<i>RCU_ADCxRST</i>	复位ADCx时钟(x=0,1,2)
<i>RCU_SDIORST</i>	复位SDIO时钟
<i>RCU_SYSCFGRST</i>	复位SYSCFG时钟
<i>RCU_TLIRST</i>	复位TLI时钟
<i>RCU_CTCRST</i>	复位CAU时钟

<i>RCU_IREFRST</i>	复位IREFR时钟
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-569. 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	除能外设复位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<i>periph_reset</i>	RCU外设复位，参考rcu_periph_reset_enum
<i>RCU_GPIOxRST</i>	复位GPIOx时钟(x=A,B,C,D,E,F,G,H,I)
<i>RCU_CRCRST</i>	复位CRC时钟
<i>RCU_DMAMxRST</i>	复位DMAx时钟(x=0,1)
<i>RCU_IPARST</i>	复位IPA时钟
<i>RCU_ENETRST</i>	复用以太网时钟
<i>RCU_USBHSRST</i>	复位USBHS时钟
<i>RCU_DCIRST</i>	复位DCI时钟
<i>RCU_TRNGRST</i>	复位TRNG时钟
<i>RCU_USBFSRST</i>	复位USBFS时钟
<i>RCU_EXMCRST</i>	复位EXMC时钟

<i>RCU_TIMERxRST</i>	复位TIMERx时钟(x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGTRST</i>	复位WWDGTRST时钟
<i>RCU_SPIxRST</i>	复位SPIx时钟(x=0,1,2,3,4,5)
<i>RCU_USARTxRST</i>	复位USARTx时钟(x=0,1,2,5)
<i>RCU_UARTxRST</i>	复位UARTx时钟(x=3,4,6,7)
<i>RCU_I2CxRST</i>	复位I2Cx时钟(x=0,1,2)
<i>RCU_CANxRST</i>	复位CANx时钟(x=0,1)
<i>RCU_PMURST</i>	复位PMU时钟
<i>RCU_DACRST</i>	复位DAC时钟
<i>RCU_ADCxRST</i>	复位ADCx时钟(x=0,1,2)
<i>RCU_SDIORST</i>	复位SDIO时钟
<i>RCU_SYSCFGRST</i>	复位SYSCFG时钟
<i>RCU_TLIRST</i>	复位TLI时钟
<i>RCU_CTCRST</i>	复位CAU时钟
<i>RCU_IREFRST</i>	复位IREFR时钟
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-570. 函数 **rcu\_bkp\_reset\_enable****

函数名称	<b>rcu_bkp_reset_enable</b>
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 3-571. 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	除能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 3-572. 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_CKSYSRC_I RC16M	选择CK_IRC16M时钟作为CK_SYS时钟源
RCU_CKSYSRC_HXTAL	选择CK_HXTAL时钟作为CK_SYS时钟源
RCU_CKSYSRC_PLLP	选择CK_PLLP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_CKSYSRC_HXTAL);
```

### 函数 rcu\_system\_clock\_source\_get

函数rcu\_system\_clock\_source\_get描述见下表:

表 3-573. 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	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<i>uint32_t</i>	选择一个时钟作为CK_SYS时钟源
<i>RCU_SCSS_IRC16M</i>	选择CK_IRC16M作为CK_SYS时钟源
<i>RCU_SCSS_HXTAL_L</i>	选择CK_HXTAL作为CK_SYS时钟源
<i>RCU_SCSS_PLLP</i>	选择CK_PLLP作为CK_SYS时钟源

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 3-574. 函数 **rcu\_ahb\_clock\_config****

函数名称	<b>rcu_ahb_clock_config</b>
函数原形	<code>void rcu_ahb_clock_config(uint32_t ck_ahb);</code>
功能描述	配置AHB时钟预分频选择
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>ck_ahb</b>	AHB预分频选择
<i>RCU_AHB_CKSYS_DIVx</i>	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 3-575. 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_CKAH_B_DIV1</i>	选择CK_AHB为CK_APB1
<i>RCU_APB1_CKAH_B_DIV2</i>	选择CK_AHB/2为CK_APB1
<i>RCU_APB1_CKAH_B_DIV4</i>	选择CK_AHB/4为CK_APB1
<i>RCU_APB1_CKAH_B_DIV8</i>	选择CK_AHB/8为CK_APB1
<i>RCU_APB1_CKAH_B_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 3-576. 函数 rcu\_apb2\_clock\_config**

函数名称	rcu_apb2_clock_config
函数原形	void rcu_apb2_clock_config(uint32_t ck_apb2);
功能描述	配置APB2时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_apb2	APB2预分频选择
RCU_APB2_CKAH_B_DIV1	选择CK_AHB为CK_APB2
RCU_APB2_CKAH_B_DIV2	选择CK_AHB/2为CK_APB2
RCU_APB2_CKAH_B_DIV4	选择CK_AHB/4为CK_APB2
RCU_APB2_CKAH_B_DIV8	选择CK_AHB/8为CK_APB2
RCU_APB2_CKAH_B_DIV16	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_CKAHB_DIV8);
```

### 函数 **rcu\_ckout0\_config**

函数rcu\_ckout0\_config描述见下表：

**表 3-577. 函数 rcu\_ckout0\_config**

函数名称	rcu_ckout0_config
函数原形	void rcu_ckout0_config(uint32_t ckout0_src, uint32_t ckout0_div);
功能描述	配置CKOUT0时钟源选择
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
ckout0_src	CK_OUT0时钟源选择
<i>RCU_CKOUT0SRC _IRC16M</i>	选择内部16M RC振荡器时钟
<i>RCU_CKOUT0SRC _LXTAL</i>	选择低速晶体振荡器时钟 (LXTAL)
<i>RCU_CKOUT0SRC _HXTAL</i>	选择高速晶体振荡器时钟 (HXTAL)
<i>RCU_CKOUT0SRC _CKPLL_PLLP</i>	选择PLL时钟
<b>输入参数{out}</b>	
ckout0_div	CK_OUT0分频器
<i>RCU_CKOUT0_DIV x(x=1,2,3,4,5)</i>	CK_OUT0分频系数
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the HXTAL as CK_OUT0 clock source */
rcu_ckout0_config(RCU_CKOUT0SRC_HXTAL, RCU_CKOUT0_DIV1);
```

### 函数 `rcu_ckout1_config`

函数`rcu_ckout1_config`描述见下表：

**表 3-578. 函数 `rcu_ckout1_config`**

函数名称	<code>rcu_ckout1_config</code>
函数原形	<code>void rcu_ckout1_config(uint32_t ckout1_src, uint32_t ckout1_div);</code>
功能描述	配置CKOUT1时钟源选择
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<code>ckout1_src</code>	CK_OUT1时钟源选择
<code>RCU_CKOUT1SRC_SYSTEMCLOCK</code>	选择系统时钟
<code>RCU_CKOUT1SRC_PLLI2SR</code>	选择PLLI2SR时钟
<code>RCU_CKOUT1SRC_HXTAL</code>	选择高速晶体振荡器时钟 (HXTAL)
<code>RCU_CKOUT1SRC_PLLP</code>	选择PLLP时钟
<b>输入参数{out}</b>	
<code>ckout1_div</code>	CK_OUT1分频器
<code>RCU_CKOUT1_DIVx(x=1,2,3,4,5)</code>	CK_OUT1分频系数
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the HXTAL as CK_OUT1 clock source */
rcu_ckout1_config(RCU_CKOUT1SRC_HXTAL, RCU_CKOUT1_DIV1);
```

### 函数 rcu\_pll\_config

函数rcu\_pll\_config描述见下表：

**表 3-579. 函数 rcu\_pll\_config**

函数名称	rcu_pll_config
函数原形	ErrStatus rcu_pll_config(uint32_t pll_src, uint32_t pll_psc, uint32_t pll_n, uint32_t pll_p, uint32_t pll_q);
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
RCU_PLLSRC_IRC_16M	IRC16M被选择为PLL, PLLSAI, PLLI2S时钟的时钟源
RCU_PLLSRC_HXTAL	HXTAL时钟被选择为PLL, PLLSAI, PLLI2S时钟的时钟源
输入参数{in}	
pll_psc	对PLL时钟源分频得到PLL VCO时钟源
uint32_t	2~63
输入参数{in}	
pll_n	对 PLL VCO 时钟源倍频得到PLL VCO
uint32_t	64~500
输入参数{in}	
pll_p	PLL VCO分频得到PLLP
uint32_t	2,4,6,8
输入参数{in}	
pll_q	PLL VCO分频得到PLLQ
uint32_t	2~15
输出参数{out}	
-	-

返回值	
ErrStatus	SUCCESS or ERROR

例如：

```
/* Configure the main PLL, PSC = 8, PLL_N = 240, PLL_P = 2, PLL_Q = 5 */
rcu_pll_config(RCU_PLLSRC_HXTAL,8,240,2,5);
```

### 函数 **rcu\_plli2s\_config**

函数rcu\_plli2s\_config描述见下表：

**表 3-580. 函数 rcu\_plli2s\_config**

函数名称	rcu_plli2s_config
函数原形	ErrStatus rcu_plli2s_config(uint32_t plli2s_n, uint32_t plli2s_r);
功能描述	配置PLLI2S时钟
先决条件	-
被调用函数	-
输入参数{in}	
plli2s_n	对 PLLI2S VCO 时钟源倍频得到PLLI2S VCO
uint32_t	50~500
输入参数{in}	
plli2s_r	对 PLLI2S VCO分频得到PLLI2S R
uint32_t	2~7
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS or ERROR

例如：

```
/* configure the PLLI2S n = 100, PLLI2S r =2*/
rcu_plli2s_config (100, 2);
```

### 函数 **rcu\_pll sai config**

函数rcu\_pll sai config描述见下表：

表 3-581. 函数 rcu\_pllsai\_config

函数名称	rcu_pllsai_config
函数原形	ErrStatus rcu_pllsai_config(uint32_t pllsai_n, uint32_t pllsai_p, uint32_t pllsai_r);
功能描述	配置PLLSAI时钟
先决条件	-
被调用函数	-
输入参数{in}	
pllsai_n	对PLLSAI VCO 时钟源倍频得到PLLSAI VCO
uint32_t	50~500
输入参数{in}	
pllsai_p	对 PLLSAI VCO分频得到PLLSAI P
uint32_t	2~15
输入参数{in}	
pllsai_r	对 PLLSAI VCO分频得到PLLSAI R
uint32_t	2~7
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS or ERROR

例如：

```
/* configure the PLLSAI n = 100, PLLSAI p = 2, PLLSAI r = 2 */
```

```
rcu_pllsai_config (100, 2,2);
```

### 函数 rcu\_rtc\_clock\_config

函数rcu\_rtc\_clock\_config描述见下表：

表 3-582. 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);

<b>功能描述</b>	配置RTC的时钟源选择
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>rtc_clock_source</b>	RTC时钟源选择
<i>RCU_RTC_SRC_NO_NE</i>	没有时钟
<i>RCU_RTC_SRC_LXTAL</i>	选择CK_LXTAL时钟作为RTC的时钟源
<i>RCU_RTC_SRC_IRC32K</i>	选择CK_IRC32K时钟作为RTC的时钟源
<i>RCU_RTC_SRC_HXTAL_DIV_RTCDIV</i>	选择CK_HXTAL或者RTCDIV后时钟作为RTC的时钟源
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the RTC clock source selection */

rcu_rtc_clock_config(RCU_RTC_SRC_IRC32K);
```

### 函数 **rcu\_i2s\_clock\_config**

函数rcu\_i2s\_clock\_config描述见下表：

**表 3-583. 函数 **rcu\_i2s\_clock\_config****

<b>函数名称</b>	<b>rcu_i2s_clock_config</b>
<b>函数原形</b>	void rcu_i2s_clock_config(uint32_t i2s_clock_source);
<b>功能描述</b>	配置I2S的时钟源选择
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	

<b>i2s_clock_source</b>	I2S时钟源选择
<i>RCU_I2SSRC_PLLI2S</i>	PLL2S被选择为I2S时钟的时钟源
<i>RCU_I2SSRC_I2S_CKIN</i>	外部引脚输入被选择为I2S时钟的时钟源
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the I2S clock source selection */
rcu_i2s_clock_config(RCU_I2SSRC_PLLI2S);
```

#### 函数 **rcu\_ck48m\_clock\_config**

函数rcu\_ck48m\_clock\_config描述见下表：

**表 3-584. 函数 rcu\_ck48m\_clock\_config**

<b>函数名称</b>	rcu_ck48m_clock_config
<b>函数原形</b>	void rcu_ck48m_clock_config(uint32_t ck48m_clock_source);
<b>功能描述</b>	配置CK48M的时钟源选择
<b>先决条件</b>	-
<b>被调用函数</b>	
<b>输入参数{in}</b>	
<i>ck48m_clock_souce</i>	CK48M时钟源选择
<i>RCU_CK48MSRC_PLL48M</i>	PLL48M被选择为CK48M时钟的时钟源
<i>RCU_CK48MSRC_IRC48M</i>	IRC48M被选择为CK48M时钟的时钟源
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	

-	-
---	---

例如：

```
/* configure the CK48M clock source selection */

rcu_ck48m_clock_config(RCU_CK48MSRC_IRC48M);
```

### 函数 **rcu\_pll48m\_clock\_config**

函数rcu\_pll48m\_clock\_config描述见下表：

**表 3-585. 函数 rcu\_pll48m\_clock\_config**

函数名称	rcu_pll48m_clock_config
函数原形	void rcu_pll48m_clock_config(uint32_t pll48m_clock_source);
功能描述	配置PLL48M的时钟源选择
先决条件	-
被调用函数	
<b>输入参数{in}</b>	
<b>pll48m_clock_sour ce</b>	PLL48M时钟源选择
<b>RCU_PLL48MSRC_ PLLQ</b>	PLLQ被选择为PLL48M时钟的时钟源
<b>RCU_PLL48MSRC_ PLLSAIP</b>	PLLSAIP被选择为PLL48M时钟的时钟源
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the PLL48M clock selection */

rcu_pll48m_clock_config(RCU_PLL48MSRC_PLLQ);
```

### 函数 **rcu\_timer\_clock\_prescaler\_config**

函数rcu\_timer\_clock\_prescaler\_config描述见下表：

表 3-586. 函数 rcu\_timer\_clock\_prescaler\_config

函数名称	rcu_timer_clock_prescaler_config
函数原形	void rcu_timer_clock_prescaler_config(uint32_t timer_clock_prescaler);
功能描述	配置TIMER时钟源
先决条件	-
被调用函数	
输入参数{in}	
timer_clock_prescaler	TIMER时钟源选择
RCU_TIMER_PSC_MUL2	如果CK_APBx = CK_AHB 或者 CK_APBx = CK_AHB/2, CK_TIMERx = CK_AHB, 否则CK_TIMERx = 2 x CK_APBx
RCU_TIMER_PSC_MUL4	如果CK_APBx = CK_AHB 或者 CK_APBx = CK_AHB/2 或者CK_APBx = CK_AHB/4, CK_TIMERx = CK_AHB, 否则CK_TIMERx = 4 x CK_APBx
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER clock source */
rcu_timer_clock_prescaler_config(RCU_TIMER_PSC_MUL4);
```

#### 函数 rcu\_tli\_clock\_div\_config

函数rcu\_tli\_clock\_div\_config描述见下表：

表 3-587. 函数 rcu\_tli\_clock\_div\_config

函数名称	rcu_tli_clock_div_config
函数原形	void rcu_tli_clock_div_config(uint32_t pllsai_r_div);
功能描述	配置TLI分频系数
先决条件	-
被调用函数	
输入参数{in}	

<b>pllsai_r_div</b>	TLI分频系数
<i>RCU_PLLSAIR_DIV</i> 2	PLLSAIR/2
<i>RCU_PLLSAIR_DIV</i> 4	PLLSAIR/4
<i>RCU_PLLSAIR_DIV</i> 8	PLLSAIR/8
<i>RCU_PLLSAIR_DIV</i> 16	PLLSAIR/16
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the TLI prescaler factor from PLLSAIR clock */
rcu_tli_clock_div_config (RCU_PLLSAIR_DIV4);
```

### 函数 **rcu\_flag\_get**

函数rcu\_flag\_get描述见下表：

**表 3-588. 函数 rcu\_flag\_get**

<b>函数名称</b>	rcu_flag_get
<b>函数原形</b>	FlagStatus rcu_flag_get(rcu_flag_enum flag);
<b>功能描述</b>	获取时钟稳定和外设复位标志
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>flag</b>	时钟稳定和外设复位标志，参考rcu_flag_enum
<i>RCU_FLAG_IRC16_MSTB</i>	IRC16M稳定标志
<i>RCU_FLAG_HXTAL_STB</i>	HXTAL稳定标志

<i>RCU_FLAG_PLLI2S_STB</i>	PLLI2S稳定标志
<i>RCU_FLAG_PLLSAI_STB</i>	PLLSAI稳定标志
<i>RCU_FLAG_PLLST_B</i>	PLL稳定标志
<i>RCU_FLAG_LXTAL_STB</i>	LXTAL稳定标志
<i>RCU_FLAG_IRC32K_STB</i>	IRC32K稳定标志
<i>RCU_FLAG_IRC48M_STB</i>	IRC48M稳定标志
<i>RCU_FLAG_EPRS_T</i>	外部引脚复位标志
<i>RCU_FLAG_PORR_ST</i>	电源复位标志
<i>RCU_FLAG_BORR_ST</i>	欠压复位标志
<i>RCU_FLAG_SWRS_T</i>	软件复位标志
<i>RCU_FLAG_FWDG_TRST</i>	独立看门狗定时器复位标志
<i>RCU_FLAG_WWD_GTRST</i>	窗口看门狗定时器复位标志
<i>RCU_FLAG_LPRST</i>	low-power复位标志
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 3-589. 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 3-590. 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rcu_int_flag_enum

<i>RCU_INT_FLAG_IR_C32KSTB</i>	IRC32K稳定中断标志
<i>RCU_INT_FLAG_L_XTALSTB</i>	LXTAL稳定中断标志
<i>RCU_INT_FLAG_IR_C16MSTB</i>	IRC16M稳定中断标志
<i>RCU_INT_FLAG_H_XTALSTB</i>	HXTAL稳定中断标志
<i>RCU_INT_FLAG_P_LLSTB</i>	PLL稳定中断标志
<i>RCU_INT_FLAG_P_LLI2SSTB</i>	PLLI2S稳定中断标志
<i>RCU_INT_FLAG_P_LLSAISTB</i>	PLLSAI稳定中断标志
<i>RCU_INT_FLAG_C_KM</i>	HXTAL时钟阻塞中断标志
<i>RCU_INT_FLAG_IR_C48MSTB</i>	IRC48M稳定中断标志
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	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 3-591. 函数 **rcu\_interrupt\_flag\_clear****

函数名称	rcu_interrupt_flag_clear
函数原形	void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag_clear);

<b>功能描述</b>	清除中断标志和时钟阻塞中断标志
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>int_flag_clear</b>	时钟稳定和阻塞中断标志清除，参考rcu_int_flag_clear_enum
<i>RCU_INT_FLAG_IR</i> <i>C32KSTB_CLR</i>	IRC32K稳定中断标志清除
<i>RCU_INT_FLAG_L</i> <i>XTALSTB_CLR</i>	LXTAL稳定中断标志清除
<i>RCU_INT_FLAG_IR</i> <i>C16MSTB_CLR</i>	IRC16M稳定中断标志清除
<i>RCU_INT_FLAG_H</i> <i>XTALSTB_CLR</i>	HXTAL稳定中断标志清除
<i>RCU_INT_FLAG_P</i> <i>LLSTB_CLR</i>	PLL稳定中断标志清除
<i>RCU_INT_FLAG_P</i> <i>LLI2SSTB_CLR</i>	PLLI2S稳定中断标志清除
<i>RCU_INT_FLAG_P</i> <i>LLSAISTB_CLR</i>	PLLSAI稳定中断标志清除
<i>RCU_INT_FLAG_C</i> <i>KM_CLR</i>	时钟阻塞中断标志清除
<i>RCU_INT_FLAG_IR</i> <i>C48MSTB_CLR</i>	IRC48M稳定中断标志清除
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-592. 函数 rcu\_interrupt\_enable**

函数名称	rcu_interrupt_enable
函数原形	void rcu_interrupt_enable(rcu_int_enum stab_int);
功能描述	使能时钟稳定中断
先决条件	-
被调用函数	-
输入参数{in}	
<b>stab_int</b>	时钟稳定中断
<i>RCU_INT_IRC32KS</i> <i>TB</i>	IRC32K稳定中断
<i>RCU_INT_LXTALS</i> <i>TB</i>	LXTAL稳定中断
<i>RCU_INT_IRC16M</i> <i>STB</i>	IRC16M稳定中断使能
<i>RCU_INT_IRC48M</i> <i>STB</i>	IRC48M稳定中断使能
<i>RCU_INT_HXTALS</i> <i>TB</i>	HXTAL稳定中断
<i>RCU_INT_PLLSTB</i>	PLL稳定中断
<i>RCU_INT_PLLI2SS</i> <i>TB</i>	PLL I2S稳定中断
<i>RCU_INT_PLLSAIS</i> <i>TB</i>	PLLSAI稳定中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the HXTAL stabilization interrupt */
```

---

```
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

### 函数 **rcu\_interrupt\_disable**

函数rcu\_interrupt\_disable描述见下表：

**表 3-593. 函数 **rcu\_interrupt\_disable****

函数名称	rcu_interrupt_disable
函数原形	void rcu_interrupt_disable(rcu_int_enum stab_int);
功能描述	除能时钟稳定中断
先决条件	-
被调用函数	-
输入参数{in}	
<b>stab_int</b>	时钟稳定中断
<i>RCU_INT_IRC32KS</i> <i>TB</i>	IRC32K稳定中断
<i>RCU_INT_LXTALS</i> <i>TB</i>	LXTAL稳定中断
<i>RCU_INT_IRC16M</i> <i>STB</i>	IRC16M稳定中断使能
<i>RCU_INT_IRC48M</i> <i>STB</i>	IRC48M稳定中断使能
<i>RCU_INT_HXTALS</i> <i>TB</i>	HXTAL稳定中断
<i>RCU_INT_PLLSTB</i>	PLL稳定中断
<i>RCU_INT_PLLI2SS</i> <i>TB</i>	PLLI2S稳定中断
<i>RCU_INT_PLLSAIS</i> <i>TB</i>	PLLSAI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);
```

### 函数 `rcu_lxtal_drive_capability_config`

函数`rcu_lxtal_drive_capability_config`描述见下表：

**表 3-594. 函数 `rcu_lxtal_drive_capability_config`**

函数名称	<code>rcu_lxtal_drive_capability_config</code>
函数原形	<code>void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);</code>
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}	
<code>lxtal_dricap</code>	LXTAL驱动能力
<code>RCU_LXTAL_LOW_DRI</code>	弱驱动能力
<code>RCU_LXTAL_HIGH_DRI</code>	强驱动能力
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the LXTAL drive capability */
rcu_lxtal_drive_capability_config(RCU_LXTAL_LOWDRI);
```

### 函数 `rcu_osc_stab_wait`

函数`rcu_osc_stab_wait`描述见下表：

**表 3-595. 函数 `rcu_osc_stab_wait`**

函数名称	<code>rcu_osc_stab_wait</code>
函数原形	<code>ErrStatus rcu_osc_stab_wait(rcu_osc_type_enum osci);</code>
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}	
<b>osci</b>	振荡器类型, 参考rcu_osc_type_enum
<i>RCU_HXTAL</i>	高速晶体振荡器
<i>RCU_LXTAL</i>	低速晶体振荡器
<i>RCU_IRC16M</i>	内部16M RC振荡器
<i>RCU_IRC48M</i>	内部48M RC振荡器
<i>RCU_IRC32K</i>	内部32K RC振荡器
<i>RCU_PLL_CK</i>	PLL锁相环
<i>RCU_PLLI2S_CK</i>	PLLI2S锁相环
<i>RCU_PLLSAI_CK</i>	PLLSAI锁相环
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	SUCCESS 或 ERROR

例如：

```
/* wait for oscillator stabilization flag */

if(SUCCESS == rcu_osc_stab_wait(RCU_HXTAL)){
}
```

#### 函数 **rcu\_osc\_on**

函数rcu\_osc\_on描述见下表：

表 3-596. 函数 **rcu\_osc\_on**

函数名称	rcu_osc_on
函数原形	void rcu_osc_on(rcu_osc_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}	
<b>osci</b>	振荡器类型, 参考rcu_osc_type_enum
<i>RCU_HXTAL</i>	高速晶体振荡器
<i>RCU_LXTAL</i>	低速晶体振荡器
<i>RCU_IRC16M</i>	内部16M RC振荡器
<i>RCU_IRC48M</i>	内部48M RC振荡器
<i>RCU_IRC32K</i>	内部32K RC振荡器
<i>RCU_PLL_CK</i>	PLL锁相环
<i>RCU_PLLI2S_CK</i>	PLLI2S锁相环
<i>RCU_PLLSAI_CK</i>	PLLSAI锁相环
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* turn on the high speed crystal oscillator */
rcu_osc_on(RCU_HXTAL);
```

### 函数 **rcu\_osc\_off**

函数rcu\_osc\_off描述见下表:

表 3-597. 函数 **rcu\_osc\_off**

函数名称	rcu_osc_off
函数原形	void rcu_osc_off(rcu_osc_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}	
<b>osci</b>	振荡器类型, 参考rcu_osc_type_enum
<i>RCU_HXTAL</i>	高速晶体振荡器

<i>RCU_LXTAL</i>	低速晶体振荡器
<i>RCU_IRC16M</i>	内部16M RC振荡器
<i>RCU_IRC48M</i>	内部48M RC振荡器
<i>RCU_IRC32K</i>	内部32K RC振荡器
<i>RCU_PLL_CK</i>	PLL锁相环
<i>RCU_PLLI2S_CK</i>	PLLI2S锁相环
<i>RCU_PLLSAI_CK</i>	PLLSAI锁相环
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* turn off the high speed crystal oscillator */
rcu_osc_i_off(RCU_HXTAL);
```

#### 函数 **rcu\_osc\_i\_bypass\_mode\_enable**

函数rcu\_osc\_i\_bypass\_mode\_enable描述见下表：

**表 3-598. 函数 rcu\_osc\_i\_bypass\_mode\_enable**

函数名称	rcu_osc_i_bypass_mode_enable
函数原形	void rcu_osc_i_bypass_mode_enable(rcu_osc_i_type_enum osci);
功能描述	使能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
<b>输入参数{in}</b>	
<b>osci</b>	振荡器类型，参考rcu_osc_i_type_enum
<i>RCU_HXTAL</i>	高速晶体振荡器
<i>RCU_LXTAL</i>	低速晶体振荡器
<b>输出参数{out}</b>	
-	-

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 3-599. 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	除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rcu_osc_type_enum
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 3-600. 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 3-601. 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	除能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\_irc16m\_adjust\_value\_set**

函数rcu\_irc16m\_adjust\_value\_set描述见下表：

**表 3-602. 函数 rcu\_irc16m\_adjust\_value\_set**

函数名称	rcu_irc16m_adjust_value_set
函数原形	void rcu_irc16m_adjust_value_set(uint8_t irc16m_adjval);
功能描述	设置内部16MHz RC振荡器时钟调整值
先决条件	-
被调用函数	-
输入参数{in}	
irc16m_adjval	IRC16M调整值（0到0x1F之间）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the IRC16M adjust value */
rcu_irc16m_adjust_value_set(0x10);
```

### 函数 **rcu\_spread\_spectrum\_config**

函数rcu\_spread\_spectrum\_config描述见下表：

**表 3-603. 函数 rcu\_spread\_spectrum\_config**

函数名称	rcu_spread_spectrum_config
函数原形	void rcu_spread_spectrum_config(uint32_t spread_spectrum_type, uint32_t modstep, uint32_t modcnt)
功能描述	配置扩频调制
先决条件	-
被调用函数	-

输入参数{in}	
<b>spread_spectrum_type</b>	PLL扩频调制类型选择
<i>RCU_SS_TYPE_CENTER</i>	选择中心扩频
<i>RCU_SS_TYPE_DOWN</i>	选择向下扩频
输入参数{in}	
<b>modstep</b>	这些位配置PLL扩频调制曲线振幅
<i>uint32_t</i>	0 ~ 0x7FFF, 满足modstep * modcnt <=2^15-1
输入参数{in}	
<b>modcnt</b>	这些位配置PLL扩频调制曲线频率
<i>uint32_t</i>	0 ~ 0x1FFF, 满足modstep * modcnt <=2^15-1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure PLL spread_spectrum */
rcu_spread_spectrum_config (RCU_SS_TYPE_CENTER, 0x0F,0x0F);
```

### 函数 **rcu\_spread\_spectrum\_enable**

函数rcu\_spread\_spectrum\_enable描述见下表：

**表 3-604. 函数 rcu\_spread\_spectrum\_enable**

函数名称	rcu_spread_spectrum_enable
函数原形	void rcu_spread_spectrum_enable(void);
功能描述	使能扩频调制
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 the PLL spread spectrum modulation */

rcu_spread_spectrum_enable();
```

#### 函数 **rcu\_spread\_spectrum\_disable**

函数rcu\_spread\_spectrum\_disable描述见下表：

表 3-605. 函数 **rcu\_spread\_spectrum\_disable**

函数名称	rcu_spread_spectrum_disable
函数原形	void rcu_spread_spectrum_disable(void);
功能描述	禁止扩频调制
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 the PLL spread spectrum modulation */

rcu_spread_spectrum_disable();
```

#### 函数 **rcu\_voltage\_key\_unlock**

函数rcu\_voltage\_key\_unlock描述见下表：

**表 3-606. 函数 rcu\_voltage\_key\_unlock**

函数名称	rcu_voltage_key_unlock
函数原形	void rcu_voltage_key_unlock (void);
功能描述	解锁电源寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* unlock the voltage key register */
rcu_voltage_key_unlock();
```

#### 函数 rcu\_deepsleep\_voltage\_set

函数rcu\_deepsleep\_voltage\_set描述见下表：

**表 3-607. 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}	
<b>dsvol</b>	深度睡眠模式电压值
<i>RCU_DEEPSLEEP_V_0</i>	在深度睡眠模式下内核电压为缺省值
<i>RCU_DEEPSLEEP_V_1</i>	在深度睡眠模式下内核电压为（缺省值-0.1）V（不建议客户使用）

<i>RCU_DEEPSLEEP_V_2</i>	在深度睡眠模式下内核电压为（缺省值-0.2）V（不建议客户使用）
<i>RCU_DEEPSLEEP_V_3</i>	在深度睡眠模式下内核电压为（缺省值-0.3）V（不建议客户使用）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* set the deep-sleep mode voltage */
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1);
```

#### 函数 **rcu\_clock\_freq\_get**

函数rcu\_clock\_freq\_get描述见下表：

**表 3-608. 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	-
<b>输入参数{in}</b>	
<b>clock</b>	要获取的时钟频率
<b>CK_SYS</b>	系统时钟
<b>CK_AHB</b>	AHB时钟
<b>CK_APB1</b>	APB1时钟
<b>CK_APB2</b>	APB2时钟
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	

<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);
```

## 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 3-609. RTC 寄存器

寄存器名称	寄存器描述
RTC_TIME	时间寄存器
RTC_DATE	日期寄存器
RTC_CTL	控制寄存器
RTC_STAT	状态寄存器
RTC_PSC	预分频寄存器
RTC_WUT	唤醒定时器寄存器
RTC_ALRM0TD	闹钟0时间日期寄存器
RTC_ALRM1TD	闹钟1时间日期寄存器
RTC_WPK	写保护钥匙寄存器
RTC_SS	亚秒寄存器
RTC_SHIFTCTL	移位控制寄存器
RTC_TTS	时间戳时间寄存器
RTC_DTS	时间戳日期寄存器
RTC_SSTS	时间戳亚秒寄存器

寄存器名称	寄存器描述
RTC_HRFC	高精度频率补偿寄存器
RTC_TAMP	侵入寄存器
RTC_ALRM0SS	闹钟0亚秒寄存器
RTC_ALRM1SS	闹钟1亚秒寄存器
RTC_BKPx(x = 0, 1, ...18, 19)	备份寄存器

### 3.22.2. 外设库函数描述

RTC库函数列表如下表所示：

**表 3-610. RTC 库函数**

库函数名称	库函数描述
rtc_deinit	复位RTC大部分寄存器
rtc_init	初始化RTC寄存器
rtc_init_mode_enter	进入RTC配置模式
rtc_init_mode_exit	退出RTC配置模式
rtc_register_sync_wait	等待RTC寄存器(RTC_TIME、RTC_DATE)与RTC的APB时钟同步并且影子寄存器更新
rtc_current_time_get	获取当前日期和时间
rtc_subsecond_get	获取当前亚秒值
rtc_alarm_config	配置RTC闹钟
rtc_alarm_subsecond_config	配置RTC闹钟亚秒
rtc_alarm_get	获取RTC闹钟
rtc_alarm_subsecond_get	获取RTC闹钟亚秒
rtc_alarm_enable	使能RTC闹钟
rtc_alarm_disable	失能RTC闹钟
rtc_timestamp_enable	使能RTC时间戳
rtc_timestamp_disable	失能RTC时间戳
rtc_timestamp_get	获取RTC时间戳时间和日期

库函数名称	库函数描述
rtc_timestamp_subsecond_get	获取RTC时间戳亚秒
rtc_timestamp_pin_map	RTC时间戳输入映射选择
rtc_tamper_enable	使能侵入检测
rtc_tamper_disable	失能侵入检测
rtc_tamper0_pin_map	tamper0输入映射选择
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
rtc_flag_clear	清除RTC标志位
rtc_alarm_output_config	配置RTC闹钟输出
rtc_calibration_output_config	配置RTC校准输出选择
rtc_hour_adjust	通过在当前时间上增加或者减少一个小时来适应夏令时和冬令时
rtc_second_adjust	调整RTC当前时间的秒或亚秒值
rtc_bypass_shadow_enable	使能RTC影子寄存器
rtc_bypass_shadow_disable	失能RTC影子寄存器
rtc_refclock_detection_enable	使能RTC参考时钟检测功能
rtc_refclock_detection_disable	失能RTC参考时钟检测功能
rtc_wakeup_enable	使能RTC自动唤醒功能
rtc_wakeup_disable	失能RTC自动唤醒功能
rtc_wakeup_clock_set	设置RTC自动唤醒定时器时钟
rtc_wakeup_timer_set	设置自动唤醒定时器值
rtc_wakeup_timer_get	获取自动唤醒定时器值
rtc_smooth_calibration_config	配置RTC平滑校准
rtc_coarse_calibration_enable	使能RTC粗校准
rtc_coarse_calibration_disable	失能RTC粗校准
rtc_coarse_calibration_config	配置RTC粗校准

### 结构体 `rtc_parameter_struct`

**表 3-611. 结构体 `rtc_parameter_struct`**

<b>Member name</b>	<b>Function description</b>
year	RTC年份值: 0x0 - 0x99(BCD格式)
month	RTC月份值
date	RTC日期值: 0x1 - 0x31(BCD格式)
day_of_week	RTC星期值
hour	RTC 小时值
minute	RTC分钟值: 0x0 - 0x59(BCD格式)
second	RTC秒值: 0x0 - 0x59(BCD格式)
factor_asyn	RTC一步分频值: 0x0 - 0x7F
factor_syn	RTC同步分频值: 0x0 - 0x7FFF
am_pm	RTC AM/PM值
display_format	RTC时间格式

### 结构体 `rtc_alarm_struct`

**表 3-612. 结构体 `rtc_alarm_struct`**

<b>Member name</b>	<b>Function description</b>
alarm_mask	RTC闹钟屏蔽
weekday_or_date	指定RTC闹钟是日期还是星期几
alarm_day	RTC闹钟日期或者星期几的值
alarm_hour	RTC闹钟小时值
alarm_minute	RTC闹钟分钟值: 0x0 - 0x59(BCD格式)
alarm_second	RTC闹钟秒数值: 0x0 - 0x59(BCD格式)
am_pm	RTC闹钟AM/PM数值

### 结构体 `rtc_timestamp_struct`

**表 3-613. 结构体 `rtc_timestamp_struct`**

<b>Member name</b>	<b>Function description</b>
timestamp_month	RTC时间戳月份值

timestamp_date	RTC 时间戳日期值: 0x1 - 0x31(BCD 格式)
timestamp_day	RTC时间戳星期值
timestamp_hour	RTC时间戳小时值
timestamp_minute	RTC时间戳分钟值: 0x0 - 0x59(BCD格式)
timestamp_second	RTC时间戳秒数值: 0x0 - 0x59(BC 格式)
am_pm	RTC时间戳AM/PM数值

### 结构体 `rtc_tamper_struct`

表 3-614. 结构体 `rtc_tamper_struct`

Member name	Function description
tamper_source	RTC侵入检测源
tamper_trigger	RTC侵入事件检测触发沿
tamper_filter	RTC 侵入事件检测在电平检测期间需要的连续采样次数
tamper_sample_frequency	RTC侵入事件电平模式检测的采样频率
tamper_preamble_enable	RTC在电压电平检测期间的预充电功能
tamper_preamble_time	RTC侵入事件电平检测采样预充电时间, 如果预充电功能使能
tamper_with_timestamp	RTC侵入事件触发时间戳

### 函数 `rtc_deinit`

函数`rtc_deinit`描述见下表:

表 3-615. 函数 `rtc_deinit`

函数名称	<code>rtc_deinit</code>
函数原型	<code>ErrStatus rtc_deinit(void);</code>
功能描述	复位大部分RTC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>ErrStatus</b>	ERROR or SUCCESS

例如：

```
/* reset most of the RTC registers*/
ErrStatus error_status = rtc_deinit();
```

### 函数 **rtc\_init**

函数**rtc\_init**描述见下表：

**表 3-616. 函数 **rtc\_init****

函数名称	rtc_init
函数原型	ErrStatus rtc_init(rtc_parameter_struct* rtc_initpara_struct);
功能描述	初始化RTC寄存器
先决条件	-
被调用函数	-
输入参数{in}	
<b>rtc_initpara_struct</b>	初始化结构体，结构体成员参考 <a href="#">表 3-611. 结构体 <b>rtc_parameter_struct</b></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize RTC registers */
rtc_parameter_struct rtc_initpara;
rtc_interrupt_disable(RTC_INT_SECOND);
rtc_initpara.factor_asyn = prescaler_a;
rtc_initpara.factor_syn = prescaler_s;
```

```

rtc_initpara.year = 0x16;
rtc_initpara.day_of_week = RTC_SATURDAY;
rtc_initpara.month = RTC_APR;
rtc_initpara.date = 0x30;
rtc_initpara.display_format = RTC_24HOUR;
rtc_initpara.am_pm = RTC_AM;
rtc_init(&rtc_initpara);

```

### **函数 `rtc_init_mode_enter`**

函数`rtc_init_mode_enter`描述见下表：

**表 3-617. 函数 `rtc_init_mode_enter`**

函数名称	rtc_init_mode_enter
函数原型	ErrStatus rtc_init_mode_enter(void);
功能描述	进入RTC初始化模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```

/* enter RTC init mode */

ErrStatus error_status = rtc_init_mode_enter ();

```

### **函数 `rtc_init_mode_exit`**

函数`rtc_init_mode_exit`描述见下表：

**表 3-618. 函数 `rtc_init_mode_exit`**

函数名称	rtc_init_mode_exit
------	--------------------

函数原型	void rtc_init_mode_exit(void);
功能描述	退出RTC初始化模式
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 init mode */

rtc_init_mode_exit();
```

#### 函数 `rtc_register_sync_wait`

函数`rtc_register_sync_wait`描述见下表：

表 3-619. 函数 `rtc_register_sync_wait`

函数名称	rtc_register_sync_wait
函数原型	ErrStatus rtc_register_sync_wait(void);
功能描述	等待RTC寄存器(RTC_TIME、RTC_DATE)与RTC的APB时钟同步并且影子寄存器更新
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 RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the
shadow registers are updated*/
```

```
ErrStatus error_status = rtc_register_sync_wait();
```

### 函数 **rtc\_current\_time\_get**

函数`rtc_current_time_get`描述见下表：

**表 3-620. 函数 `rtc_current_time_get`**

函数名称	rtc_current_time_get
函数原型	void rtc_current_time_get(rtc_parameter_struct* rtc_initpara_struct);
功能描述	获取当前的时间和日期
先决条件	-
被调用函数	-
输入参数{in}	
<code>rtc_initpara_struct</code>	初始化结构体，结构体成员参考 <a href="#">表 3-611. 结构体 <code>rtc_parameter_struct</code></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*get current time and date*/
rtc_parameter_struct rtc_initpara_struct;
rtc_current_time_get (&rtc_initpara_struct);
```

### 函数 **rtc\_subsecond\_get**

函数`rtc_subsecond_get`描述见下表：

**表 3-621. 函数 `rtc_subsecond_get`**

函数名称	rtc_subsecond_get
函数原型	uint32_t rtc_subsecond_get(void);
功能描述	获取当前亚秒值

先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint32_t</b>	当前的亚秒值(0x00-0xFFFF)

例如：

```
/*get current subsecond value*/
uint32_t sub_second = rtc_subsecond_get();
```

### 函数 **rtc\_alarm\_config**

函数`rtc_alarm_config`描述见下表：

**表 3-622. 函数 `rtc_alarm_config`**

函数名称	rtc_alarm_config
函数原型	<code>void rtc_alarm_config(uint8_t rtc_alarm, rtc_alarm_struct* rtc_alarm_time);</code>
功能描述	配置RTC闹钟
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>rtc_alarm</b>	闹钟选择
<i>RTC_ALARM0</i>	闹钟0
<i>RTC_ALARM1</i>	闹钟1
<b>输入参数{in}</b>	
<b>rtc_alarm_time</b>	闹钟结构体，结构体成员参考 <a href="#">表 3-612. 结构体<code>rtc_alarm_struct</code></a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	

-	-
---	---

例如：

```
/*rtc_alarm_config*/
rtc_alarm_struct rtc_alarm_time;
rtc_alarm_config (&rtc_alarm_time);
```

### 函数 **rtc\_alarm\_subsecond\_config**

函数`rtc_alarm_subsecond_config`描述见下表：

**表 3-623. 函数 `rtc_alarm_subsecond_config`**

函数名称	rtc_alarm_subsecond_config
函数原型	void rtc_alarm_subsecond_config(uint8_t rtc_alarm, uint32_t mask_subsecond, uint32_t subsecond)
功能描述	配置RTC闹钟的亚秒值
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>rtc_alarm</b>	闹钟选择
<i>RTC_ALARM0</i>	闹钟0
<i>RTC_ALARM1</i>	闹钟1
<b>输入参数{in}</b>	
<b>mask_subsecond</b>	闹钟亚秒屏蔽位
<i>RTC_MASKSSC_0_14</i>	屏蔽闹钟亚秒设置
<i>RTC_MASKSSC_1_14</i>	屏蔽RTC_ALRM0SS_SSC[14:1], SSC[0]位用于时间匹配
<i>RTC_MASKSSC_2_14</i>	屏蔽RTC_ALRM0SS_SSC[14:2], SSC[1:0]位用于时间匹配
<i>RTC_MASKSSC_3_14</i>	屏蔽RTC_ALRM0SS_SSC[14:3], SSC[2:0]位用于时间匹配
<i>RTC_MASKSSC_4_14</i>	屏蔽RTC_ALRM0SS_SSC[14:4], SSC[3:0]位用于时间匹配
<i>RTC_MASKSSC_5_14</i>	屏蔽RTC_ALRM0SS_SSC[14:5], SSC[4:0]位用于时间匹配
<i>RTC_MASKSSC_6_14</i>	屏蔽RTC_ALRM0SS_SSC[14:6], SSC[5:0]位用于时间匹配
<i>RTC_MASKSSC_7_14</i>	屏蔽RTC_ALRM0SS_SSC[14:7], SSC[6:0]位用于时间匹配

<i>RTC_MASKSSC_8_14</i>	屏蔽RTC_ALRM0SS_SSC[14:8], SSC[7:0]位用于时间匹配
<i>RTC_MASKSSC_9_14</i>	屏蔽RTC_ALRM0SS_SSC[14:9], SSC[8:0]位用于时间匹配
<i>RTC_MASKSSC_10_14</i>	屏蔽RTC_ALRM0SS_SSC[14:10], SSC[9:0]位用于时间匹配
<i>RTC_MASKSSC_11_14</i>	屏蔽RTC_ALRM0SS_SSC[14:11], SSC[10:0]位用于时间匹配
<i>RTC_MASKSSC_12_14</i>	屏蔽RTC_ALRM0SS_SSC[14:12], SSC[11:0]位用于时间匹配
<i>RTC_MASKSSC_13_14</i>	屏蔽RTC_ALRM0SS_SSC[14:13], SSC[12:0]位用于时间匹配
<i>RTC_MASKSSC_14</i>	屏蔽RTC_ALRM0SS_SSC[14], SSC[13:0]位用于时间匹配
<i>RTC_MASKSSC_NON_E</i>	无屏蔽, SSC[14:0]位用于时间匹配
<b>输入参数{in}</b>	
<b>subsecond</b>	闹钟亚秒值(0x000 - 0x7FF)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/*configure subsecond of RTC alarm0*/
rtc_subsecond_config(RTC_ALARM0, RTC_MASKSSC_9_14, 0x7FFF);
```

### 函数 **rtc\_alarm\_get**

函数`rtc_alarm_get`描述见下表:

**表 3-624. 函数 `rtc_alarm_get`**

函数名称	<code>rtc_alarm_get</code>
函数原型	<code>void rtc_alarm_get(uint8_t rtc_alarm, rtc_alarm_struct* rtc_alarm_time);</code>
功能描述	获取RTC闹钟
先决条件	-
被调用函数	-

输入参数{in}	
<b>rtc_alarm</b>	闹钟选择
<i>RTC_ALARM0</i>	闹钟0
<i>RTC_ALARM1</i>	闹钟1
输出参数{out}	
<b>rtc_alarm_time</b>	闹钟结构体, 结构体成员参考 <a href="#">表 3-612. 结构体 rtc_alarm_struct</a>
返回值	
-	-

例如:

```
/* get RTC alarm0*/
rtc_alarm_get (RTC_ALARM0, &rtc_alarm_time);
```

#### 函数 **rtc\_alarm\_subsecond\_get**

函数 **rtc\_alarm\_subsecond\_get** 描述见下表:

**表 3-625. 函数 rtc\_alarm\_subsecond\_get**

函数名称	rtc_alarm_subsecond_get
函数原型	uint32_t rtc_alarm_subsecond_get(uint8_t rtc_alarm);
功能描述	获取RTC闹钟亚秒值
先决条件	-
被调用函数	-
输入参数{in}	
<b>rtc_alarm</b>	闹钟选择
<i>RTC_ALARM0</i>	闹钟0
<i>RTC_ALARM1</i>	闹钟1
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	RTC 闹钟亚秒值(0x0-0x3FFF)

例如:

---

```
/*get RTC alarm0 subsecond*/
uint32_t subsecond =  rtc_alarm_subsecond_get(RTC_ALARM0);
```

### 函数 **rtc\_alarm\_enable**

函数`rtc_alarm_enable`描述见下表：

**表 3-626. 函数 `rtc_alarm_enable`**

函数名称	rtc_alarm_enable
函数原型	<code>void rtc_alarm_enable(uint8_t rtc_alarm);</code>
功能描述	使能RTC闹钟
先决条件	-
被调用函数	-
输入参数{in}	
<code>rtc_alarm</code>	闹钟选择
<code>RTC_ALARM0</code>	闹钟0
<code>RTC_ALARM1</code>	闹钟1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*enable RTC alarm0*/
rtc_alarm_enable(RTC_ALARM0);
```

### 函数 **rtc\_alarm\_disable**

函数`rtc_alarm_disable`描述见下表：

**表 3-627. 函数 `rtc_alarm_disable`**

函数名称	rtc_alarm_disable
函数原型	<code>ErrStatus rtc_alarm_disable(uint8_t rtc_alarm);</code>
功能描述	失能RTC闹钟
先决条件	-

<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>rtc_alarm</b>	闹钟选择
<i>RTC_ALARM0</i>	闹钟0
<i>RTC_ALARM1</i>	闹钟1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>ErrStatus</b>	ERROR或SUCCESS

例如：

```
/*disable RTC alarm1*/
ErrStatus error_status = rtc_alarm_disable(RTC_ALARM0);
```

#### 函数 **rtc\_timestamp\_enable**

函数can\_init描述见下表：

**表 3-628. 函数 rtc\_timestamp\_enable**

<b>函数名称</b>	rtc_timestamp_enable
<b>函数原型</b>	void rtc_timestamp_enable(uint32_t edge);
<b>功能描述</b>	使能RTC时间戳
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>edge</b>	选定哪种边沿触发时间戳检测
<i>RTC_TIMESTAMP_RISING_EDGE</i>	上升沿是时间戳事件有效检测沿
<i>RTC_TIMESTAMP_FALLING_EDGE</i>	下降沿是时间戳事件有效检测沿
<b>输出参数{out}</b>	
-	-

返回值	
-	-

例如：

```
/*enable RTC time-stamp*/
rtc_timestamp_enable (RTC_TIMESTAMP_RISING_EDGE);
```

### 函数 **rtc\_timestamp\_disable**

函数`rtc_timestamp_disable`描述见下表：

**表 3-629. 函数 `rtc_timestamp_disable`**

函数名称	rtc_timestamp_disable
函数原型	<code>void rtc_timestamp_disable(void);</code>
功能描述	失能RTC时间戳
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 time-stamp*/
rtc_timestamp_disable ();
```

### 函数 **rtc\_timestamp\_get**

函数`rtc_timestamp_get`描述见下表：

**表 3-630. 函数 `rtc_timestamp_get`**

函数名称	rtc_timestamp_get
函数原型	<code>void rtc_timestamp_get(rtc_timestamp_struct* rtc_timestamp);</code>
功能描述	获取RTC时间戳时间和日期

先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
<b>rtc_timestamp</b>	时间戳结构体, 结构体成员参考 <a href="#">表 3-613. 结构体 rtc_timestamp struct</a>
<b>返回值</b>	
-	-

例如:

```
/* get RTC timestamp time and date */
rtc_timestamp_struct rtc_timestamp;
rtc_timestamp_get(& rtc_timestamp);
```

#### 函数 **rtc\_timestamp\_subsecond\_get**

函数`rtc_timestamp_subsecond_get`描述见下表:

**表 3-631. 函数 rtc\_timestamp\_subsecond\_get**

函数名称	rtc_timestamp_subsecond_get
函数原型	uint32_t rtc_timestamp_subsecond_get(void);
功能描述	获取RTC时间戳亚秒值
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint32_t</b>	RTC时间戳亚秒值

例如:

```
/* get RTC time-stamp subsecond */
```

```
uint32_t subsecond = rtc_timestamp_subsecond_get();
```

### 函数 **rtc\_timestamp\_pin\_map**

函数`rtc_timestamp_pin_map`描述见下表:

**表 3-632. 函数 `rtc_timestamp_pin_map`**

函数名称	rtc_timestamp_pin_map
函数原型	<code>void rtc_timestamp_pin_map(uint32_t rtc_af);</code>
功能描述	时间戳输入映射选择
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<code>rtc_af</code>	输入映射选择
<code>RTC_AF0_TIMESTAMP</code> <i>P</i>	时间戳输入映射到RTC_AF0
<code>RTC_AF1_TIMESTAMP</code> <i>P</i>	时间戳输入映射到RTC_AF1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* RTC_AF0 use for timestamp */
rtc_timestamp_pin_map (RTC_AF0_TIMESTAMP);
```

### 函数 **rtc\_tamper\_enable**

函数`rtc_tamper_enable`描述见下表:

**表 3-633. 函数 `rtc_timestamp_enable`**

函数名称	rtc_tamper_enable
函数原型	<code>void rtc_tamper_enable(rtc_tamper_struct* rtc_tamper);</code>
功能描述	使能RTC侵入检测
先决条件	-

<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>rtc_tamper</b>	tamper化结构体, 结构体成员参考 <a href="#">表 3-614. 结构体 rtc_tamper_struct</a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* enable RTC tamper */

rtc_tamper_struct rtc_tamper

rtc_tamper_enable(& rtc_tamper);
```

#### 函数 **rtc\_tamper\_disable**

函数`rtc_tamper_disable`描述见下表:

**表 3-634. 函数 `rtc_tamper_disable`**

<b>函数名称</b>	rtc_tamper_disable
<b>函数原型</b>	void rtc_tamper_disable(uint32_t source);
<b>功能描述</b>	失能RTC侵入检测
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>source</b>	选定被失能的侵入检测来源
<i>RTC_TAMPER0</i>	RTC tamper0
<i>RTC_TAMPER1</i>	RTC tamper1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

---

```
/* disable RTC tamper */

rtc_tamper_disable(RTC_TAMPER0);
```

### 函数 **rtc\_tamper0\_pin\_map**

函数`rtc_tamper0_pin_map`描述见下表：

**表 3-635. 函数 `rtc_tamper0_pin_map`**

函数名称	rtc_tamper0_pin_map
函数原型	void rtc_tamper0_pin_map(uint32_t rtc_af);
功能描述	Tamper0输入映射选择
先决条件	-
被调用函数	-
输入参数{in}	
<code>rtc_af</code>	输入映射选择
<code>RTC_AF0_TAMPER0</code>	Tamper0输入映射到RTC_AF0
<code>RTC_AF1_TAMPER0</code>	Tamper0输入映射到RTC_AF1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* RTC_AF0 use for tamper0 */

rtc_tamper0_pin_map (RTC_AF0_TAMPER0);
```

### 函数 **rtc\_interrupt\_enable**

函数`rtc_interrupt_enable`描述见下表：

**表 3-636. 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指定的中断
先决条件	-

<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>interrupt</b>	选定被使能的中断源
<i>RTC_INT_TIMESTAMP</i>	时间戳中断
<i>RTC_INT_ALARM0</i>	闹钟0中断
<i>RTC_INT_ALARM1</i>	闹钟1中断
<i>RTC_INT_TAMP</i>	侵入检测中断
<i>RTC_INT_WAKEUP</i>	唤醒定时器中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable specified RTC interrupt*/
rtc_interrupt_enable(RTC_INT_TAMP);
```

#### 函数 **rtc\_interrupt\_disable**

函数 **rtc\_interrupt\_disable** 描述见下表：

**表 3-637. 函数 **rtc\_interrupt\_disable****

<b>函数名称</b>	rtc_interrupt_disable
<b>函数原型</b>	void rtc_interrupt_disable(uint32_t interrupt);
<b>功能描述</b>	失能RTC指定中断
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>interrupt</b>	选定被失能的RTC中断
<i>RTC_INT_TIMESTAMP</i>	时间戳中断
<i>RTC_INT_ALARM0</i>	闹钟0中断
<i>RTC_INT_ALARM1</i>	闹钟1中断

<i>RTC_INT_TAMP</i>	侵入检测中断
<i>RTC_INT_WAKEUP</i>	唤醒定时器中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable RTC ALARM interrupt */

rtc_interrupt_disable(RTC_INT_TAMP);
```

### 函数 **rtc\_flag\_get**

函数`rtc_flag_get`描述见下表：

表 3-638. 函数 **rtc\_flag\_get**

函数名称	<code>rtc_flag_get</code>
函数原型	<code>FlagStatus rtc_flag_get(uint32_t flag);</code>
功能描述	获取指定中断标志位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>flag</b>	选定被获取的中断标志
<i>RTC_STAT_SCP</i>	平滑校准挂起标志
<i>RTC_FLAG_TP1</i>	tamper 1事件标志
<i>RTC_FLAG_TPO</i>	tamper 0事件标志
<i>RTC_FLAG_TSOVR</i>	时间戳事件溢出标志
<i>RTC_FLAG_TS</i>	时间戳事件标志
<i>RTC_FLAG_ALARM0</i>	Alarm0发生标志
<i>RTC_FLAG_ALARM1</i>	Alarm1发生标志
<i>RTC_FLAG_WT</i>	唤醒定时器发生标志
<i>RTC_FLAG_INIT</i>	进入初始化模式

<i>RTC_FLAG_RSYN</i>	寄存器同步标志
<i>RTC_FLAG_YCM</i>	年份配置标志
<i>RTC_FLAG_SOP</i>	移位功能操作挂起标志
<i>RTC_FLAG_ALRM0W</i>	Alarm0配置可写标志
<i>RTC_FLAG_ALRM1W</i>	Alarm1配置可写标志
<i>RTC_FLAG_WTW</i>	唤醒定时器可写标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
FlagStatus	SET 或 RESET

例如：

```
/* check time-stamp event flag */
FlagStatus = rtc_flag_get(RTC_FLAG_TIMESTAMP)
```

### 函数 **rtc\_flag\_clear**

函数**rtc\_flag\_clear**描述见下表：

**表 3-639. 函数 **rtc\_flag\_clear****

函数名称	rtc_flag_clear
函数原型	void rtc_flag_clear(uint32_t flag);
功能描述	清除指定中断标志位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>flag</b>	要清除的中断标志位
<i>RTC_FLAG_TP1</i>	tamper 1事件标志
<i>RTC_FLAG_TPO</i>	tamper 0事件标志
<i>RTC_FLAG_TS0VR</i>	时间戳事件溢出标志
<i>RTC_FLAG_TS</i>	时间戳事件标志
<i>RTC_FLAG_WT</i>	唤醒定时器可写标志

<i>RTC_FLAG_ALARM0</i>	闹钟0发生标志
<i>RTC_FLAG_ALARM1</i>	闹钟1发生标志
<i>RTC_FLAG_RSYN</i>	寄存器同步标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* cleartime-stamp event flag */
rtc_flag_clear (RTC_FLAG_TIMESTAMP);
```

#### 函数 **rtc\_alter\_output\_config**

函数`rtc_alter_output_config`描述见下表：

**表 3-640. 函数 `rtc_alter_output_config`**

函数名称	<code>rtc_alter_output_config</code>
函数原型	<code>void rtc_alter_output_config(uint32_t source, uint32_t mode);</code>
功能描述	配置RTC备用输出源
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>source</b>	指定输出信号
<i>RTC_ALARM0_HIGH</i>	当设置了闹钟0标志置位，输出引脚为高电平
<i>RTC_ALARM0_LOW</i>	当设置了闹钟0标志置位，输出引脚为低电平
<i>RTC_ALARM1_HIGH</i>	当设置了闹钟1标志置位，输出引脚为高电平
<i>RTC_ALARM1_LOW</i>	当设置了闹钟1标志置位，输出引脚为低电平
<i>RTC_WAKEUP_HIGH</i>	当设置了唤醒定时器标志置位，输出引脚位高电平
<i>RTC_WAKEUP_LOW</i>	当设置了唤醒定时器标志置位，输出引脚位低电平
<b>输入参数{in}</b>	
<b>mode</b>	当输出闹钟信号时指定输出引脚(PC13)的模式

<i>RTC_ALARM_OUTPU T_OD</i>	开漏输出
<i>RTC_ALARM_OUTPU T_PP</i>	推挽输出
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure rtc alternate output source */
rtc_alter_output_config(RTC_ALARM0_HIGH, RTC_ALARM_OUTPUT_PP);
```

### 函数 **rtc\_calibration\_output\_config**

函数`rtc_calibration_output_config`描述见下表：

**表 3-641. 函数 `rtc_calibration_output_config`**

函数名称	rtc_calibration_output_config
函数原型	void rtc_calibration_output_config(uint32_t source);
功能描述	配置RTC校准输出源
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>source</b>	指定输出信号
<i>RTC_CALIBRATION_5 12HZ</i>	当外部低速时钟频率为32768Hz并且RTC_PSC为默认值，输出512Hz信号
<i>RTC_CALIBRATION_1 HZ</i>	当外部低速时钟频率为32768Hz并且RRTC_PSC为默认值，输出1Hz信号
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 1Hz
signal */

rtc_calibration_output_config(RTC_CALIBRATION_1HZ);
```

### **函数 `rtc_hour_adjust`**

函数`rtc_hour_adjust`描述见下表：

**表 3-642. 函数 `rtc_hour_adjust`**

函数名称	rtc_hour_adjust
函数原型	void rtc_hour_adjust(uint32_t operation);
功能描述	通过在当前时间上增加或者减少一个小时来适应夏令时和冬令时
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>operation</b>	小时调整操作
<i>RTC_CTL_A1H</i>	增加一个小时
<i>RTC_CTL_S1H</i>	减少一个小时
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* adjust the daylight saving time by adding one hour from the current time */

rtc_hour_adjust(RTC_CTL_A1H);
```

### **函数 `rtc_second_adjust`**

函数`rtc_second_adjust`描述见下表：

**表 3-643. 函数 `rtc_second_adjust`**

函数名称	rtc_second_adjust
函数原型	ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);

功能描述		调整RTC当前时间的秒或亚秒值
先决条件		-
被调用函数		-
输入参数{in}		
<b>add</b>		在当前时间上增加1S或者不增加
<i>RTC_SHIFT_ADD1S_RESET</i>		无影响
<i>RTC_SHIFT_ADD1S_SET</i>		在当前时间增加1秒
输入参数{in}		
<b>minus</b>		在当前是时间上减少的亚秒值(0x0 - 0x7FFF)
输出参数{out}		
-		-
返回值		
-		-

例如：

```
/* adjust RTC second or subsecond value of current time */
ErrStatus error_status = rtc_second_adjust(RTC_SHIFT_ADD1S_SET, 0);
```

#### 函数 **rtc\_bypass\_shadow\_enable**

函数**rtc\_bypass\_shadow\_enable**描述见下表：

**表 3-644. 函数 **rtc\_bypass\_shadow\_enable****

函数名称	rtc_bypass_shadow_enable
函数原型	void rtc_bypass_shadow_enable(void);
功能描述	使能RTC影子寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable RTC bypass shadow registers function*/
rtc_bypass_shadow_enable();
```

#### 函数 **rtc\_bypass\_shadow\_disable**

函数`rtc_bypass_shadow_disable`描述见下表：

**表 3-645. 函数 `rtc_bypass_shadow_disable`**

函数名称	rtc_bypass_shadow_disable
函数原型	void rtc_bypass_shadow_disable (void);
功能描述	失能RTC影子寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable RTC bypass shadow registers function*/
rtc_bypass_shadow_disable();
```

#### 函数 **rtc\_refclock\_detection\_enable**

函数`rtc_refclock_detection_enable`描述见下表：

**表 3-646. 函数 `rtc_refclock_detection_enable`**

函数名称	rtc_refclock_detection_enable
------	-------------------------------

<b>函数原型</b>	ErrStatus rtc_refclock_detection_enable(void);
<b>功能描述</b>	使能RTC参考时钟检测功能
<b>先决条件</b>	-
<b>被调用函数</b>	rtc_init_mode_enter/rtc_init_mode_exit
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>ErrStatus</b>	ERROR 或 SUCCESS

例如：

```
/* enable RTC reference clock detection function*/
ErrStatus error_status = rtc_refclock_detection_enable();
```

#### 函数 **rtc\_refclock\_detection\_disable**

函数`rtc_refclock_detection_disable`描述见下表：

**表 3-647. 函数 `rtc_refclock_detection_disable`**

<b>函数名称</b>	rtc_refclock_detection_disable
<b>函数原型</b>	ErrStatus rtc_refclock_detection_disable(void);
<b>功能描述</b>	失能RTC参考时钟检测功能
<b>先决条件</b>	-
<b>被调用函数</b>	rtc_init_mode_enter/rtc_init_mode_exit
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

例如：

---

```
/* disableRTC reference clock detection function*/
ErrStatus error_status = rtc_refclock_detection_disable();
```

### **函数 `rtc_wakeup_enable`**

函数`rtc_wakeup_enable`描述见下表：

**表 3-648. 函数 `rtc_wakeup_enable`**

函数名称	rtc_wakeup_enable
函数原型	void rtc_wakeup_enable(void);
功能描述	使能RTC自动唤醒功能
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 RTC auto wakeup function*/
rtc_wakeup_enable( );
```

### **函数 `rtc_wakeup_disable`**

函数`rtc_wakeup_disable`描述见下表：

**表 3-649. 函数 `rtc_wakeup_disable`**

函数名称	rtc_wakeup_disable
函数原型	ErrStatus rtc_wakeup_disable(void);
功能描述	失能RTC自动唤醒功能
先决条件	-
被调用函数	-
输入参数{in}	

-	
输出参数{out}	
-	
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* disable RTC auto wakeup function*/
ErrStatus error_status = rtc_wakeup_disable( );
```

### 函数 **rtc\_wakeup\_clock\_set**

函数`rtc_wakeup_clock_set`描述见下表：

表 3-650. 函数 **rtc\_wakeup\_clock\_set**

函数名称	rtc_wakeup_clock_set
函数原型	ErrStatus rtc_wakeup_clock_set(uint8_t wakeup_clock);
功能描述	设置RTC自动唤醒定时器时钟
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_clock	时钟选择
WAKEUP_RTCCK_DIV 16	RTC时钟的16分频
WAKEUP_RTCCK_DIV 8	RTC时钟的8分频
WAKEUP_RTCCK_DIV 4	RTC时钟的4分频
WAKEUP_RTCCK_DIV 2	RTC时钟的2分频
WAKEUP_CKSPRE	ck_spre(默认1Hz)时钟
WAKEUP_CKSPRE_2 EXP16	ck_spre(默认1Hz)时钟并且将唤醒计数器值增加 $2^{16}$
输出参数{out}	

-	-
返回值	
<b>ErrStatus</b>	ERROR or SUCCESS

例如：

```
/* RTC auto wakeup timer clock is ckspre */

ErrStatus error_status = rtc_wakeup_clock_set (WAKEUP_CKSPRE);
```

### 函数 **rtc\_wakeup\_timer\_set**

函数**rtc\_wakeup\_timer\_set**描述见下表：

**表 3-651. 函数 **rtc\_wakeup\_timer\_set****

函数名称	rtc_wakeup_timer_set
函数原型	ErrStatus rtc_wakeup_timer_set(uint16_t wakeup_timer);
功能描述	设置RTC自动唤醒定时器值
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
wakeup_timer	定时器选择
value	0x0000-0xffff
<b>输出参数{out}</b>	
-	-
返回值	
<b>ErrStatus</b>	ERROR or SUCCESS

例如：

```
/* set wakeup timer value */

ErrStatus error_status = rtc_wakeup_timer_set (0xFFEE);
```

### 函数 **rtc\_wakeup\_timer\_get**

函数**rtc\_wakeup\_timer\_get**描述见下表：

**表 3-652. 函数 **rtc\_wakeup\_timer\_get****

函数名称	rtc_wakeup_timer_get
------	----------------------

<b>函数原型</b>	uint16_t rtc_wakeup_timer_get(void);
<b>功能描述</b>	获取唤醒定时器值
<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>	
<b>uint16_t</b>	0-0xFFFF

例如：

```
/* get wakeup timer value*/
uint16_t wakeuptimer_value;
wakeuptimer_value = rtc_wakeup_timer_get();
```

### 函数 **rtc\_smooth\_calibration\_config**

函数**rtc\_smooth\_calibration\_config**描述见下表：

**表 3-653. 函数 **rtc\_smooth\_calibration\_config****

<b>函数名称</b>	rtc_smooth_calibration_config
<b>函数原型</b>	ErrStatus rtc_smooth_calibration_config(uint32_t window, uint32_t plus, uint32_t minus);
<b>功能描述</b>	配置RTC平滑校准
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>window</b>	校准窗口选择
<b>RTC_CALIBRATION_WINDOW_32S</b>	采用32S校准周期
<b>RTC_CALIBRATION_WINDOW_16S</b>	采用16S校准周期

<i>RTC_CALIBRATION_WINDOW_8S</i>	采用8S校准周期
<b>输入参数{in}</b>	
<i>RTC_CALIBRATION_PLUS_SET</i>	每2048个脉冲增加一个RTCCLK脉冲
<i>RTC_CALIBRATION_PLUS_RESET</i>	无影响
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

例如：

```
/* configure RTC smooth calibration*/
ErrStatus error_status;
error_status = rtc_smooth_calibration_config(RTC_CALIBRATION_WINDOW_32S,
RT_CALIBRATION_PLUS_SET);
```

#### 函数 **rtc\_coarse\_calibration\_enable**

函数 **rtc\_coarse\_calibration\_enable** 描述见下表：

**表 3-654. 函数 **rtc\_coarse\_calibration\_enable****

函数名称	rtc_coarse_calibration_enable
函数原型	ErrStatus rtc_coarse_calibration_enable(void);
功能描述	使能RTC粗校准功能
先决条件	-
被调用函数	rtc_init_mode_enter/rtc_init_mode_exit
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	

<b>ErrStatus</b>	ERROR or SUCCESS
------------------	------------------

例如：

```
/* enable RTC coarse calibration */

ErrStatus error_status = rtc_coarse_calibration_enable();
```

#### 函数 **rtc\_coarse\_calibration\_disable**

函数`rtc_coarse_calibration_disable`描述见下表：

**表 3-655. 函数 `rtc_coarse_calibration_disable`**

函数名称	rtc_coarse_calibration_disable
函数原型	ErrStatus rtc_coarse_calibration_disable(void);
功能描述	失能RTC粗校准功能
先决条件	-
被调用函数	rtc_init_mode_enter/rtc_init_mode_exit
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR or SUCCESS

例如：

```
/* disable RTC coarse calibration */

ErrStatus error_status = rtc_coarse_calibration_disable();
```

#### 函数 **rtc\_coarse\_calibration\_config**

函数`rtc_coarse_calibration_config`描述见下表：

**表 3-656. 函数 `rtc_coarse_calibration_config`**

函数名称	rtc_coarse_calibration_config
函数原型	ErrStatus rtc_coarse_calibration_config(uint8_t direction, uint8_t step);
功能描述	配置RTC粗校准方向和步伐
先决条件	-

被调用函数	rtc_init_mode_enter/rtc_init_mode_exit
<b>输入参数{in}</b>	
<b>direction</b>	粗校准方向
<b>CALIB_INCREASE</b>	增加日历更新频率
<b>CALIB_DECREASE</b>	降低日历更新频率
<b>粗校准步伐</b>	
<b>step</b>	粗校准步伐 当COSD=0: 0x00: +0PPM 0x01: +4PPM(近似值) 0x02: +8PPM (近似值) ..... 0x1F: +126PPM (近似值)  当COSD=1: 0x00: -0PPM 0x01: -2PPM(近似值) 0x02: -4PPM (近似值) ..... 0x1F: -63PPM (近似值)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

例如：

```
/* config coarse calibration direction and step */
ErrStatus error_status = rtc_coarse_calibration_config (CALIB_INCREASE, 0x01);
```

### 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APB2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 3-657. SDIO 寄存器**

寄存器名称	寄存器描述
SDIO_PWRCTL	电源控制寄存器
SDIO_CLKCTL	时钟控制寄存器
SDIO_CMDAGMT	命令参数寄存器
SDIO_CMDCTL	命令控制寄存器
SDIO_RSPCMDIDX	命令索引响应寄存器
SDIO_RESPx x=0..3	响应寄存器
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 3-658. 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

库函数名称	库函数描述
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
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

库函数名称	库函数描述
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 3-659. 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 3-660. 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
<b>SDIO_SDIOCLKED GE_RISING</b>	选择SDIOCLK的上升沿产生SDIO_CLK
<b>SDIO_SDIOCLKED GE_FALLING</b>	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
<b>SDIO_CLOCKBYPASS SS_ENABLE</b>	使能旁路时钟
<b>SDIO_CLOCKBYPASS SS_DISABLE</b>	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

<i>SDIO_CLOCKPWR SAVE_ENABLE</i>	SDIO_CLK时钟在总线空闲时关闭
<i>SDIO_CLOCKPWR SAVE_DISABLE</i>	SDIO_CLK时钟总是开启
<b>输入参数{in}</b>	
<b>clock_division</b>	时钟分频， 小于512
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the SDIO clock */

sdio_clock_config(SDIO_SDIOCLKEDGE_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 3-661. 函数 **sdio\_hardware\_clock\_enable**

函数名称	<b>sdio_hardware_clock_enable</b>
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	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-662. 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 3-663. 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}	

<b>bus_mode</b>	SDIO卡总线模式
<b>SDIO_BUSMODE_1BIT</b>	1位SDIO卡总线模式
<b>SDIO_BUSMODE_4BIT</b>	4位SDIO卡总线模式
<b>SDIO_BUSMODE_8BIT</b>	8位SDIO卡总线模式
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-664. 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	-
<b>输入参数{in}</b>	
<b>power_state</b>	SDIO电源状态
<b>SDIO_POWER_ON</b>	SDIO上电
<b>SDIO_POWER_OFF</b>	SDIO断电
<b>输出参数{out}</b>	
-	-

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 3-665. 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 3-666. 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 3-667. 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 3-668. 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}	
<b>cmd_index</b>	命令索引，请参阅相关规范
输入参数{in}	
<b>cmd_argument</b>	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
<b>SDIO_RESPONSETYPE_NO</b>	无响应
<b>SDIO_RESPONSETYPE_SHORT</b>	短响应
<b>SDIO_RESPONSETYPE_LONG</b>	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 3-669. 函数 **sdio\_wait\_type\_set****

函数名称	sdio_wait_type_set
函数原形	void sdio_wait_type_set(uint32_t wait_type);
功能描述	设置命令状态机等待类型
先决条件	-
被调用函数	-
输入参数{in}	
<b>wait_type</b>	等待类型
<b>SDIO_WAITTYPE_NO</b>	不等待中断
<b>SDIO_WAITTYPE_INTERRUPT</b>	等待中断
<b>SDIO_WAITTYPE_DATAEND</b>	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 3-670. 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 3-671. 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 3-672. 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	
<b>uint8_t</b>	上一次响应的命令索引

例如:

```
/* get SDIO command index */

uint8_t sdio_command_value;

sdio_command_value = sdio_command_index_get();
```

### 函数 **sdio\_response\_get**

函数**sdio\_response\_get**描述见下表:

**表 3-673. 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}	
<b>responsex</b>	SDIO响应

<i>SDIO_RESPONSE0</i>	卡响应 [31:0]/卡响应 [127:96]
<i>SDIO_RESPONSE1</i>	卡响应 [95:64]
<i>SDIO_RESPONSE2</i>	卡响应 [63:32]
<i>SDIO_RESPONSE3</i>	卡响应 [31:1], 加上位0
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint32_t</b>	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 3-674. 函数 **sdio\_data\_config****

函数名称	<b>sdio_data_config</b>
函数原形	<code>void sdio_data_config(uint32_t data_timeout, uint32_t data_length, uint32_t data_blocksize);</code>
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	-
<b>输入参数{in}</b>	
<b>data_timeout</b>	卡总线时钟周期中的数据超时周期
<b>输入参数{in}</b>	
<b>data_length</b>	要传输的数据字节数
<b>输入参数{in}</b>	
<b>data_blocksize</b>	块传输中数据块的大小
<b>SDIO_DATABLOCK_SIZE_1BYTE</b>	块大小 = 1字节

<code>SDIO_DATABLOCK_SIZE_2BYTES</code>	块大小 = 2字节
<code>SDIO_DATABLOCK_SIZE_4BYTES</code>	块大小 = 4字节
<code>SDIO_DATABLOCK_SIZE_8BYTES</code>	块大小 = 8字节
<code>SDIO_DATABLOCK_SIZE_16BYTES</code>	块大小 = 16字节
<code>SDIO_DATABLOCK_SIZE_32BYTES</code>	块大小 = 32字节
<code>SDIO_DATABLOCK_SIZE_64BYTES</code>	块大小 = 64字节
<code>SDIO_DATABLOCK_SIZE_128BYTES</code>	块大小 = 128字节
<code>SDIO_DATABLOCK_SIZE_256BYTES</code>	块大小 = 256字节
<code>SDIO_DATABLOCK_SIZE_512BYTES</code>	块大小 = 512字节
<code>SDIO_DATABLOCK_SIZE_1024BYTES</code>	块大小 = 1024字节
<code>SDIO_DATABLOCK_SIZE_2048BYTES</code>	块大小 = 2048字节
<code>SDIO_DATABLOCK_SIZE_4096BYTES</code>	块大小 = 4096字节
<code>SDIO_DATABLOCK_SIZE_8192BYTES</code>	块大小 = 8192字节
<code>SDIO_DATABLOCK_SIZE_16384BYTES</code>	块大小 = 16384字节
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-675. 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}	
<b>transfer_mode</b>	数据传输模式
<i>SDIO_TRANSMODE_E_BLOCK</i>	块传输模式
<i>SDIO_TRANSMODE_E_STREAM</i>	流传输或SDIO多字节传输模式
输入参数{in}	
<b>transfer_direction</b>	数据传输方向
<i>SDIO_TRANSDIR_ECTION_TOCARD</i>	写数据到卡上
<i>SDIO_TRANSDIR_ECTION_TOSDIO</i>	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_TRANSDIR_ECTION_TOSDIO,
```

SDIO\_TRANSMODE\_BLOCK);

### 函数 **sdio\_dsm\_enable**

函数**sdio\_dsm\_enable**描述见下表:

**表 3-676. 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	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-677. 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	-
<b>输入参数{in}</b>	
-	-

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 3-678. 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}	
data	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 3-679. 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 3-680. 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	
<b>uint32_t</b>	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 3-681. 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 3-682. 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 3-683. 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 3-684. 函数 **sdio\_flag\_get****

函数名称	<b>sdio_flag_get</b>
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	-
<b>输入参数{in}</b>	
<b>flag</b>	SDIO标志位状态
<i>SDIO_FLAG_CCRC ERR</i>	命令响应已接收 (CRC检测失败)
<i>SDIO_FLAG_DTCR CERR</i>	数据块已发送/已接收 (CRC检测失败)
<i>SDIO_FLAG_CMDT MOUT</i>	命令响应超时
<i>SDIO_FLAG_DTTM OUT</i>	数据超时
<i>SDIO_FLAG_TXUR E</i>	发送FIFO下溢错误发生
<i>SDIO_FLAG_RXOR E</i>	接收FIFO上溢错误发生
<i>SDIO_FLAG_CMDR ECV</i>	命令响应已接收 (CRC检测通过)
<i>SDIO_FLAG_CMDS END</i>	命令已发送 (不需响应)
<i>SDIO_FLAG_DTEN D</i>	数据结束 (数据计数器, SDIO_DATACNT为零)
<i>SDIO_FLAG_STBIT E</i>	总线上起始位错误
<i>SDIO_FLAG_DTBL KEND</i>	数据块已发送/已接收 (CRC检测通过)

<code>SDIO_FLAG_CMDR UN</code>	正在传输命令
<code>SDIO_FLAG_TXRU N</code>	正在传输数据
<code>SDIO_FLAG_RXRU N</code>	正在接收数据
<code>SDIO_FLAG_TFH</code>	发送FIFO半空：至少还有8个字可被写入到FIFO中
<code>SDIO_FLAG_RFH</code>	接收FIFO半满：FIFO中至少还有8个字可被读取
<code>SDIO_FLAG_TFF</code>	发送FIFO为满
<code>SDIO_FLAG_RFF</code>	接收FIFO为满
<code>SDIO_FLAG_TFE</code>	发送FIFO为空
<code>SDIO_FLAG_RFE</code>	接收FIFO为空
<code>SDIO_FLAG_TXDT VAL</code>	发送FIFO中的数据有效
<code>SDIO_FLAG_RXDT VAL</code>	接收FIFO中的数据有效
<code>SDIO_FLAG_SDIOI NT</code>	SD I/O中断已接收
<code>SDIO_FLAG_ATAE ND</code>	CE-ATA命令完成信号已接收（仅用于CMD61）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	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 3-685. 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	-
<b>输入参数{in}</b>	
<b>flag</b>	SDIO标志位状态
<i>SDIO_FLAG_CCRC ERR</i>	命令响应已接收 (CRC检测失败)
<i>SDIO_FLAG_DTCR CERR</i>	数据块已发送/已接收 (CRC检测失败)
<i>SDIO_FLAG_CMDT MOUT</i>	命令响应超时
<i>SDIO_FLAG_DTTM OUT</i>	数据超时
<i>SDIO_FLAG_TXUR E</i>	发送FIFO下溢错误发生
<i>SDIO_FLAG_RXOR E</i>	接收FIFO上溢错误发生
<i>SDIO_FLAG_CMDR ECV</i>	命令响应已接收 (CRC检测通过)
<i>SDIO_FLAG_CMDS END</i>	命令已发送 (不需响应)
<i>SDIO_FLAG_DTEN D</i>	数据结束 (数据计数器, SDIO_DATACNT为零)
<i>SDIO_FLAG_STBIT E</i>	总线上起始位错误
<i>SDIO_FLAG_DTBL KEND</i>	数据块已发送/已接收 (CRC检测通过)
<i>SDIO_FLAG_SDIOI NT</i>	SD I/O中断已接收
<i>SDIO_FLAG_ATAE</i>	CE-ATA命令完成信号已接收 (仅用于CMD61)

<i>ND</i>	
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 3-686. 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}	
<b>int_flag</b>	SDIO中断标志位状态
<i>SDIO_INT_CCRCE</i> <i>RR</i>	命令响应CRC错误中断
<i>SDIO_INT_DTCRC</i> <i>ERR</i>	数据CRC错误中断
<i>SDIO_INT_CMDTM</i> <i>OUT</i>	命令响应超时中断
<i>SDIO_INT_DTTMO</i> <i>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</i> <i>CV</i>	命令响应已接收中断

<i>SDIO_INT_CMDSEN</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_DTBLSN</i>	数据块已发送/已接收中断
<i>SDIO_INT_CMDRUN</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_TXDTVAL</i>	发送FIFO中的数据有效中断
<i>SDIO_INT_RXDTVAL</i>	接收FIFO中的数据有效中断
<i>SDIO_INT_SDIOINT</i>	SD I/O中断已接收中断
<i>SDIO_INT_ATAEN</i>	CE-ATA命令完成信号已接收中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable the SDIO corresponding interrupts */
```

```
sdio_interrupt_enable(SDIO_INT_CCRCERR | 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 3-687. 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_DTBLKE ND	数据块已发送/已接收中断
SDIO_INT_CMDRU N	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_TXDTV<sub>L</sub></i>	发送FIFO中的数据有效中断
<i>SDIO_INT_RXDTV<sub>AL</sub></i>	接收FIFO中的数据有效中断
<i>SDIO_INT_SDIOIN<sub>T</sub></i>	SD I/O中断已接收中断
<i>SDIO_INT_ATAEN<sub>D</sub></i>	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 3-688. 函数 **sdio\_interrupt\_flag\_get****

函数名称	<b>sdio_interrupt_flag_get</b>
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

<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>int_flag</b>	SDIO中断标志位状态
<b>SDIO_INT_FLAG_C CRCERR</b>	命令响应CRC错误中断
<b>SDIO_INT_FLAG_D TCRCERR</b>	数据CRC错误中断
<b>SDIO_INT_FLAG_C MDTMOUT</b>	命令响应超时中断
<b>SDIO_INT_FLAG_D TTMOUT</b>	数据超时中断
<b>SDIO_INT_FLAG_T XURE</b>	发送FIFO下溢错误中断
<b>SDIO_INT_FLAG_R XORE</b>	接收FIFO上溢错误中断
<b>SDIO_INT_FLAG_C MDRECV</b>	命令响应已接收中断
<b>SDIO_INT_FLAG_C MDSEND</b>	命令已发送中断
<b>SDIO_INT_FLAG_D TEND</b>	数据结束中断
<b>SDIO_INT_FLAG_S TBITE</b>	起始位错误中断
<b>SDIO_INT_FLAG_D TBLKEND</b>	数据块已发送/已接收中断
<b>SDIO_INT_FLAG_C MDRUN</b>	正在传输命令中断
<b>SDIO_INT_FLAG_T XRUN</b>	正在传输数据中断
<b>SDIO_INT_FLAG_R XRUN</b>	正在接收数据中断
<b>SDIO_INT_FLAG_T</b>	发送FIFO半满中断

<i>FH</i>	
<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 3-689. 函数 **sdio\_interrupt\_flag\_clear**

函数名称	sdio_interrupt_flag_clear
------	---------------------------

<b>函数原形</b>	void sdio_interrupt_flag_clear(uint32_t int_flag);
<b>功能描述</b>	清除SDIO的中断标志位状态
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>int_flag</b>	SDIO中断标志位状态
<b>SDIO_INT_FLAG_C CRCERR</b>	命令响应CRC错误中断
<b>SDIO_INT_FLAG_D TCRCERR</b>	数据CRC错误中断
<b>SDIO_INT_FLAG_C MDTMOUT</b>	命令响应超时中断
<b>SDIO_INT_FLAG_D TTMOUT</b>	数据超时中断
<b>SDIO_INT_FLAG_T XURE</b>	发送FIFO下溢错误中断
<b>SDIO_INT_FLAG_R XORE</b>	接收FIFO上溢错误中断
<b>SDIO_INT_FLAG_C MDRECV</b>	命令响应已接收中断
<b>SDIO_INT_FLAG_C MDSEND</b>	命令已发送中断
<b>SDIO_INT_FLAG_D TEND</b>	数据结束中断
<b>SDIO_INT_FLAG_S TBITE</b>	起始位错误中断
<b>SDIO_INT_FLAG_D TBLKEND</b>	数据块已发送/已接收中断
<b>SDIO_INT_FLAG_S DIOINT</b>	SD I/O中断已接收中断
<b>SDIO_INT_FLAG_A TAEND</b>	CE-ATA命令完成信号已接收中断
<b>输出参数{out}</b>	

-	-
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 3-690. 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 3-691. 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 3-692. 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 3-693. 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 3-694. 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

<i>SDIO_READWAITT YPE_CLK</i>	通过停止SDIO_CLK控制读等待
<i>SDIO_READWAITT YPE_DAT2</i>	使用SDIO_DAT[2] 控制读等待
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-695. 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	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-696. 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 3-697. 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休眠模式（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 3-698. 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休眠模式（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 3-699. 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 3-700. 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 3-701. 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 3-702. 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 3-703. 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 3-704. 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 3-705. 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	四路SPI控制寄存器
I2S_ADD_CTL0	I2S_ADD 控制寄存器0

寄存器名称	寄存器描述
I2S_ADD_CTL1	I2S_ADD 控制寄存器1
I2S_ADD_STAT	I2S_ADD 状态寄存器
I2S_ADD_DATA	I2S_ADD 数据寄存器
I2S_ADD_CRCPOLY	I2S_ADD CRC多项式寄存器
I2S_ADD_RCRC	I2S_ADD 接收CRC寄存器
I2S_ADD_TCRC	I2S_ADD 发送CRC寄存器
I2S_ADD_I2SCTL	I2S_ADD I2S控制寄存器
I2S_ADD_I2SPSC	I2S_ADD I2S时钟分频寄存器

### 3.24.2. 外设库函数说明

SPI/I2S库函数列表如下表所示：

**表 3-706. 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_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_ti_mode_enable	使能SPI TI模式
spi_ti_mode_disable	禁能SPI TI模式

库函数名称	库函数描述
i2s_full_duplex_mode_config	配置I2S全双工模式
qspi_enable	使能四线SPI模式
qspi_disable	禁能四线SPI模式
qspi_write_enable	使能四线SPI写
qspi_read_enable	使能四线SPI读
qspi_io23_output_enable	使能SPI_IO2和SPI_IO3输出
qspi_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
spi_crc_error_clear	清除SPIx CRC错误标志状态

### 结构体 **spi\_parameter\_struct**

表 3-707. 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_FRAMESIZE_16BIT, SPI_FRAMESIZE_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 3-708. 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=0,1,2,3,4,5
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 3-709. 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	
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 3-710. 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=0,1,2,3,4,5
输入参数{in}	
spi_struct	初始化结构体, 结构体成员参考 <a href="#">表 3-707. 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_FRAMESIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_CK_PL_HIGH_PH_2EDGE;
spi_init_struct.nss                = SPI_NSS_SOFT;
spi_init_struct.prescale            = SPI_PSC_8;
spi_init_struct.endian              = SPI_ENDIAN_MSB;

spi_init(SPI0, &spi_init_struct);
```

### 函数 spi\_enable

函数spi\_enable描述见下表:

表 3-711. 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=0,1,2,3,4,5
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 3-712. 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=0,1,2,3,4,5
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 3-713. 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}	
<b>i2s_mode</b>	I2S运行模式
<i>I2S_MODE_SLAVE</i> TX	I2S从机发送模式
<i>I2S_MODE_SLAVE</i> RX	I2S从机接收模式
<i>I2S_MODE_MASTER</i> RTX	I2S主机发送模式
<i>I2S_MODE_MASTER</i> RRX	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_PCMSHORT</i> RT	I2S PCM短帧标准
<i>I2S_STD_PCMLONG</i> 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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize I2S1 */

i2s_init(SPI1, I2S_MODE_MASTER_TX, I2S_STD_PHILLIPS, I2S_CKPL_LOW);
```

### 函数 i2s\_psc\_config

函数i2s\_psc\_config描述见下表：

表 3-714. 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_i2s_clock_config/ rcu_osc_on/ rcu_osc_stab_wait

输入参数{in}	
<b>spi_periph</b>	外设I2Sx
<b>SPIx</b>	x=1,2
输入参数{in}	
<b>i2s_audiosample</b>	I2S音频采样频率
<b>I2S_AUDIOSAMPL_E_8K</b>	音频采样频率为8KHz
<b>I2S_AUDIOSAMPL_E_11K</b>	音频采样频率为11KHz
<b>I2S_AUDIOSAMPL_E_16K</b>	音频采样频率为16KHz
<b>I2S_AUDIOSAMPL_E_22K</b>	音频采样频率为22KHz
<b>I2S_AUDIOSAMPL_E_32K</b>	音频采样频率为32KHz
<b>I2S_AUDIOSAMPL_E_44K</b>	音频采样频率为44KHz
<b>I2S_AUDIOSAMPL_E_48K</b>	音频采样频率为48KHz
<b>I2S_AUDIOSAMPL_E_96K</b>	音频采样频率为96KHz
<b>I2S_AUDIOSAMPL_E_192K</b>	音频采样频率为192KHz
输入参数{in}	
<b>i2s_frameformat</b>	I2S数据长度和通道长度
<b>I2S_FRAMEFORMA_T_DT16B_CH16B</b>	I2S数据长度为16位，通道长度为16位
<b>I2S_FRAMEFORMA_T_DT16B_CH32B</b>	I2S数据长度为16位，通道长度为32位
<b>I2S_FRAMEFORMA_T_DT24B_CH32B</b>	I2S数据长度为24位，通道长度为32位
<b>I2S_FRAMEFORMA_T_DT32B_CH32B</b>	I2S数据长度为32位，通道长度为32位
输入参数{in}	
<b>i2s_mckout</b>	2S_MCK输出使能
<b>I2S_MCKOUT_ENA_BLE</b>	I2S_MCK输出使能
<b>I2S_MCKOUT_DISABLE</b>	I2S_MCK输出禁止
输出参数{out}	
-	-
返回值	

-	-
---	---

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 3-715. 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}	
<b>spi_periph</b>	外设SPI/I2Sx
<b>SPIx</b>	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 3-716. 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}	
<b>spi_periph</b>	外设SPI/I2Sx
<b>SPIx</b>	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 3-717. 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=0,1,2,3,4,5
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 3-718. 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=0,1,2,3,4,5

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 3-719. 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=0,1,2,3,4,5
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 3-720. 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=0,1,2,3,4,5
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-721. 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	-
<b>输入参数{in}</b>	
spi_periph	外设SPIx
SPIx	x=0,1,2,3,4,5
<b>输入参数{in}</b>	
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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-722. 函数 `spi_dma_disable`

函数名称	spi_dma_disable
函数原形	<code>void spi_dma_disable(uint32_t spi_periph, uint8_t dma);</code>
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}	
<code>spi_periph</code>	外设SPIx
<code>SPIx</code>	x=0,1,2,3,4,5
输入参数{in}	
<code>dma</code>	SPI DMA模式
<code>SPI_DMA_TRANSMIT</code>	SPI发送缓冲区DMA使能
<code>SPI_DMA_RECEIVE</code>	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 3-723. 函数 `spi_i2s_data_frame_format_config`

函数名称	spi_i2s_data_frame_format_config
函数原形	<code>void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);</code>
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}	
<code>spi_periph</code>	外设SPIx
<code>SPIx</code>	x=0,1,2,3,4,5
输入参数{in}	
<code>frame_format</code>	SPI帧大小
<code>SPI_FRAMESIZE_16BIT</code>	SPI 16位数据帧格式
<code>SPI_FRAMESIZE_8</code>	SPI 8位数据帧格式

<i>BIT</i>	
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure SPI1/I2S1 data frame format size is 16 bits */
spi_i2s_data_frame_format_config(SPI1, SPI_FRAMESIZE_16BIT);
```

### 函数 **spi\_i2s\_data\_transmit**

函数**spi\_i2s\_data\_transmit**描述见下表：

**表 3-724. 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	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
SPIx	x=0,1,2,3,4,5
<b>输入参数{in}</b>	
<b>data</b>	16位数据
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* SPI0 transmit data */

uint16_t spi_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 3-725. 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	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=0,1,2,3,4,5
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	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 3-726. 函数 **spi\_bidirectional\_transfer\_config**

函数名称	<b>spi_bidirectional_transfer_config</b>
函数原形	void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);
功能描述	配置外设SPIx的数据传输方向
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=0,1,2,3,4,5
<b>输入参数{in}</b>	
<b>transfer_direction</b>	SPI双向传输输出使能
<b>SPI_BIDIRECTION</b>	SPI工作在只发送模式
<b>AL_TRANSMIT</b>	
<b>SPI_BIDIRECTION</b>	SPI工作在只接收模式
<b>AL_RECEIVE</b>	
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* SPI0 works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

### 函数 **spi\_crc\_polynomial\_set**

函数spi\_crc\_polynomial\_set描述见下表:

**表 3-727. 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}	
spi_periph	外设SPIx
SPIx	x=0,1,2,3,4,5
输入参数{in}	
crc_poly	CRC多项式值
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set SPI0 CRC polynomial */
uint16 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 3-728. 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=0,1,2,3,4,5
输出参数{out}	
-	-
返回值	

<b>uint16_t</b>	16位CRC多项式值 (0-0xFFFF)
-----------------	-----------------------

例如：

```
/* get SPI0 CRC polynomial */

uint16_t crc_val;

crc_val = spi_crc_polynomial_get(SPI0);
```

### 函数 **spi\_crc\_on**

函数**spi\_crc\_on**描述见下表：

**表 3-729. 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}	
<b>spi_periph</b>	外设SPIx
SPIx	x=0,1,2,3,4,5
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 3-730. 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}	
<b>spi_periph</b>	外设SPIx
SPIx	x=0,1,2,3,4,5
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 3-731. 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=0,1,2,3,4,5
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 3-732. 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 spi_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=0,1,2,3,4,5

输入参数{in}	
<b>spi_crc</b>	SPI crc值
<b>SPI_CRC_TX</b>	获取发送CRC寄存器值
<b>SPI_CRC_RX</b>	获取接收CRC寄存器值
输出参数{out}	
-	-
返回值	
<b>uint16_t</b>	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\_ti\_mode\_enable**

函数**spi\_ti\_mode\_enable**描述见下表：

**表 3-733. 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}	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=0,1,2,3,4,5
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 3-734. 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	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=0,1,2,3,4,5
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable SPI0 TI mode */
spi_ti_mode_disable(SPI0);
```

### 函数 i2s\_full\_duplex\_mode\_config

函数i2s\_full\_duplex\_mode\_config描述见下表：

表 3-735. 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 frameformat);
功能描述	I2S全双工模式配置
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>i2s_add_periph</b>	外设I2Sx_ADD
<b>I2Sx_ADD</b>	x=1,2
<b>输入参数{in}</b>	
<b>i2s_mode</b>	I2S模式
<b>I2S_MODE_SLAVE_TX</b>	I2S从机发送模式
<b>I2S_MODE_SLAVE_RX</b>	I2S从机接收模式
<b>I2S_MODE_MASTER_RTX</b>	I2S主机发送模式
<b>I2S_MODE_MASTER_RRX</b>	I2S主机接收模式
<b>输入参数{in}</b>	
<b>i2s_standard</b>	I2S标准
<b>I2S_STD_PHILLIPS</b>	I2S飞利浦标准
<b>I2S_STD_MSB</b>	I2S MSB对其标准

<i>I2S_STD_LSB</i>	I2S LSB 对其标准
<i>I2S_STD_PCMSHO</i> <i>RT</i>	I2S PCM短帧标准
<i>I2S_STD_PCMLON</i> <i>G</i>	I2S PCM长帧标准
<b>输入参数{in}</b>	
<i>i2s_ckpl</i>	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>输出参数{out}</b>	
<i>i2s_frameformat</i>	I2S数据长度和通道长度
<i>I2S_FRAMEFORMAT_DT16B_CH16B</i>	I2S数据长度为16位，通道长度为16位
<i>I2S_FRAMEFORMAT_DT16B_CH32B</i>	I2S数据长度为16位，通道长度为32位
<i>I2S_FRAMEFORMAT_DT24B_CH32B</i>	I2S数据长度为24位，通道长度为32位
<i>I2S_FRAMEFORMAT_DT32B_CH32B</i>	I2S数据长度为32位，通道长度为32位
<b>返回值</b>	
-	-
-	-

例如：

```
/* configure i2s full duplex mode */

i2s_full_duplex_mode_config(I2S1_ADD,I2S_MODE_SLAVETX,I2S_STD_LSB,I2S_CKPL_
LOW,I2S_FRAMEFORMAT_DT16B_CH16B);
```

### 函数 **qspi\_enable**

函数**qspi\_enable**描述见下表：

**表 3-736. 函数 **qspi\_enable****

函数名称	<b>qspi_enable</b>
函数原形	void qspi_enable(uint32_t spi_periph);
功能描述	使能四线SPI模式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=5
<b>输出参数{out}</b>	

-	-
返回值	
-	-

例如：

```
/* enable SPI5 quad wire mode */

qspi_enable(SPI5);
```

### 函数 **qspi\_disable**

函数**qspi\_disable**描述见下表：

**表 3-737. 函数 qspi\_disable**

函数名称	qspi_disable
函数原形	qspi_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=5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI5 quad wire mode */

qspi_disable(SPI5);
```

### 函数 **qspi\_write\_enable**

函数**qspi\_write\_enable**描述见下表：

**表 3-738. 函数 qspi\_write\_enable**

函数名称	qspi_write_enable
函数原形	void qspi_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=5

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI5 quad wire write */
qspi_write_enable(SPI5);
```

### 函数 qspi\_read\_enable

函数qspi\_read\_enable描述见下表：

**表 3-739. 函数 qspi\_read\_enable**

函数名称	qspi_read_enable
函数原形	void qspi_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=5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI5 quad wire read */
qspi_read_enable(SPI5);
```

### 函数 qspi\_io23\_output\_enable

函数qspi\_io23\_output\_enable描述见下表：

**表 3-740. 函数 qspi\_io23\_output\_enable**

函数名称	qspi_io23_output_enable
函数原形	void qspi_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

<b>SPIx</b>	x=5
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable SPI5 SPI_IO2 and SPI_IO3 pin output */
qspi_io23_output_enable(SPI5);
```

### 函数 **qspi\_io23\_output\_disable**

函数qspi\_io23\_output\_disable描述见下表：

**表 3-741. 函数 qspi\_io23\_output\_disable**

<b>函数名称</b>	qspi_io23_output_disable
<b>函数原形</b>	void qspi_io23_output_disable(uint32_t spi_periph);
<b>功能描述</b>	禁能SPI_IO2和SPI_IO3输出
<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
<b>SPIx</b>	x=5
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable SPI5 SPI_IO2 and SPI_IO3 pin output */
qspi_io23_output_disable(SPI5);
```

### 函数 **spi\_i2s\_interrupt\_enable**

函数spi\_i2s\_interrupt\_enable描述见下表：

**表 3-742. 函数 spi\_i2s\_interrupt\_enable**

<b>函数名称</b>	spi_i2s_interrupt_enable
<b>函数原形</b>	void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t spi_i2s_int);
<b>功能描述</b>	使能外设SPIx/I2Sx中断
<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
<b>SPIx</b>	x=0,1,2,3,4,5
<b>输入参数{in}</b>	
<b>spi_i2s_int</b>	SPI/I2S中断
<b>SPI_I2S_INT_TBE</b>	发送缓冲区空中断使能
<b>SPI_I2S_INT_RBNE</b>	接收缓冲区非空中断使能
<b>SPI_I2S_INT_ERR</b>	错误中断使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-743. 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 spi_i2s_int);
功能描述	禁能外设SPIx/I2Sx中断
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=0,1,2,3,4,5
<b>输入参数{in}</b>	
<b>spi_i2s_int</b>	SPI/I2S中断
<b>SPI_I2S_INT_TBE</b>	发送缓冲区空中断使能
<b>SPI_I2S_INT_RBNE</b>	接收缓冲区非空中断使能
<b>SPI_I2S_INT_ERR</b>	错误中断使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-744. 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 spi_i2s_int);
功能描述	获取外设SPIx/I2Sx中断状态
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=0,1,2,3,4,5
<b>输入参数{in}</b>	
<b>spi_i2s_int</b>	SPI/I2S中断状态
<b>SPI_I2S_INT_FLAG_TBE</b>	发送缓冲区空中断
<b>SPI_I2S_INT_FLAG_RBNE</b>	接收缓冲区非空中断
<b>SPI_I2S_INT_FLAG_RXORERR</b>	接收过载错误中断
<b>SPI_INT_FLAG_CO_NFERR</b>	配置错误中断
<b>SPI_INT_FLAG_CR_CERR</b>	CRC错误中断
<b>I2S_INT_FLAG_TXURERR</b>	发送欠载错误中断
<b>I2S_INT_FLAG_FE_RR</b>	帧错误中断
<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 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 3-745. 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 spi_i2s_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=0,1,2,3,4,5
输入参数{in}	
<b>spi_i2s_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_FLAG_RXORE</b> <i>RR</i>	接收过载错误标志
<b>SPI_FLAG_CONFE</b> <i>RR</i>	配置错误标志
<b>SPI_FLAG_CRCER</b> <i>R</i>	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_RXORE</b> <i>RR</i>	接收过载错误标志
<b>I2S_FLAG_TXURE</b> <i>RR</i>	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
输出参数{out}	
-	-
返回值	
<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++]);
```

### 函数 **spi\_crc\_error\_clear**

函数**spi\_crc\_error\_clear**描述见下表：

**表 3-746. 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	-
<b>输入参数{in}</b>	
spi_periph	外设SPIx
SPIx	x=0,1,2,3,4,5
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* clear SPI0 CRC error flag status */
spi_crc_error_clear(SPI0);
```

## 3.25. SYSCFG

章节[3.25.1](#)描述了SYSCFG的寄存器列表，章节[3.25.2](#)对SYSCFG库函数进行说明。

### 3.25.1. 外设寄存器说明

SYSCFG寄存器列表如下表所示：

**表 3-747. SYSCFG 寄存器**

寄存器名称	寄存器描述
SYSCFG_CFG0	配置寄存器0
SYSCFG_CFG1	配置寄存器1
SYSCFG_EXTI0	EXTI源选择寄存器0
SYSCFG_EXTI1	EXTI源选择寄存器1
SYSCFG_EXTI2	EXTI源选择寄存器2

寄存器名称	寄存器描述
SYSCFG_EXTISSL3	EXTI源选择寄存器3
SYSCFG_CPSCTL	I/O补偿控制寄存器

### 3.25.2. 外设库函数说明

SYSCFG库函数列表如下表所示：

**表 3-748. SYSCFG 库函数**

库函数名称	库函数描述
syscfg_deinit	复位SYSCFG模块
syscfg_bootmode_config	配置引导模式
syscfg_fmc_swap_config	配置FMC存储器映射切换
syscfg_exmc_swap_config	配置EXMC存储器映射切换
syscfg_exti_line_config	配置EXTI源选择
syscfg_enet_phy_interface_config	配置以太网PHY接口
syscfg_compensation_config	使能或禁用I/O补偿单元
syscfg_flag_get	获取I/O补偿单元状态

#### 函数 `syscfg_deinit`

函数`syscfg_deinit`描述见下表：

**表 3-749. 函数 `syscfg_deinit`**

函数名称	syscfg_deinit
函数原型	void syscfg_deinit(void);
功能描述	复位SYSCFG模块
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* reset SYSCFG */

syscfg_deinit();
```

### 函数 **syscfg\_bootmode\_config**

函数**syscfg\_bootmode\_config**描述见下表：

**表 3-750. 函数 **syscfg\_bootmode\_config****

函数名称	syscfg_bootmode_config
函数原型	void syscfg_bootmode_config(uint8_t syscfg_bootmode);
功能描述	配置引导模式
先决条件	-
被调用函数	-
输入参数{in}	
<b>syscfg_bootmode</b>	引导模式
<b>SYSCFG_BOOTMO<sub>DE_FLASH</sub></b>	主FLASH存储器(0x0800 0000~0x08FF FFFF)被映射到地址0x0000 0000
<b>SYSCFG_BOOTMO<sub>DE_BOOTLOADER</sub></b>	引导装载代码所在系统存储器(0x1FFF 0000~0x1FFF 7FFF)被映射到地址0x0000 0000
<b>SYSCFG_BOOTMO<sub>DE_EXMC_SRAM</sub></b>	EXMC的SRAM/NOR 0和1(0x6000 0000~0x67FF FFFF) 被映射到地址0x0000 0000
<b>SYSCFG_BOOTMO<sub>DE_SRAM</sub></b>	片上SRAM的SRAM0 (0x2000 0000~0x2001 BFFF) 被映射到地址0x0000 0000
<b>SYSCFG_BOOTMO<sub>DE_EXMC_SDRAM</sub></b>	EXMC的SDRAM Device0(0xC000 0000~0xC7FF FFFF) 被映射到地址0x0000 0000
输出参数{out}	
-	-
返回值	
-	-

例如：

---

```
/* configure boot from main flash */
syscfg_bootmode_config(SYSCFG_BOOTMODE_FLASH);
```

### 函数 **syscfg\_fmc\_swap\_config**

函数**syscfg\_fmc\_swap\_config**描述见下表:

**表 3-751. 函数 syscfg\_fmc\_swap\_config**

函数名称	syscfg_fmc_swap_config
函数原型	void syscfg_fmc_swap_config(uint32_t syscfg_fmc_swap);
功能描述	配置FMC存储器映射切换
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_fmc_swap	FMC映射切换指向的存储器
SYSCFG_FMC_SW_P_BANK0	主FLASH存储器的Bank0被映射到地址0x0800 0000, 主FLASH存储器的Bank1被映射到地址0x0810 0000
SYSCFG_FMC_SW_P_BANK1	主FLASH存储器的Bank1映射到地址0x0800 0000, 主FLASH存储器的Bank0映射到地址0x0810 0000
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* FMC memory bank 0 is mapped at address 0x08000000 and bank 1 is mapped at address
0x08100000 */

syscfg_fmc_swap_config(SYSCFG_FMC_SWP_BANK0);
```

### 函数 **syscfg\_exmc\_swap\_config**

函数**syscfg\_exmc\_swap\_config**描述见下表:

**表 3-752. 函数 syscfg\_exmc\_swap\_config**

函数名称	syscfg_exmc_swap_config
函数原型	void syscfg_exmc_swap_config(uint32_t syscfg_exmc_swap);

功能描述	配置EXMC存储器映射切换
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>syscfg_exmc_swa_p</b>	EXMC映射切换指向的存储器
<b>SYSCFG_EXMC_SWP_ENABLE</b>	无存储器映射切换
<b>SYSCFG_EXMC_SWP_DISABLE</b>	SDRAM的Bank0和Bank1与NAND Bank1和PC CARD进行切换，然后， SDRAM的Bank0和Bank1被映射到从0x8000 0000到0x9FFF FFFF的地址范围， NAND的Bank1被映射到从0xC000 0000到0xCFFF FFFF的地址范围，PC CARD被映射到从0xD000 0000到0xDFFF FFFF的地址范围。
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable EXMC memory mapping swap */
syscfg_exmc_swap_config(SYSCFG_EXMC_SWP_ENABLE);
```

#### 函数 **syscfg\_exti\_line\_config**

函数**syscfg\_exti\_line\_config**描述见下表：

**表 3-753. 函数 syscfg\_exti\_line\_config**

函数名称	syscfg_exti_line_config
函数原型	void syscfg_exti_line_config(uint8_t exti_port, uint8_t exti_pin);
功能描述	配置EXTI源选择
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>exti_port</b>	GPIO端口
<b>EXTI_SOURCE_GP</b>	x = A,B,C,D,E,F,G,H,I

<i>I/Ox</i>	
<b>输入参数{in}</b>	
<b>exti_pin</b>	EXTI线
<i>EXTI_SOURCE_PI_Nx</i>	x = 0..15
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the PA0 pin as EXTI Line */
syscfg_exti_line_config(EXTI_SOURCE_GPIOA, EXTI_SOURCE_PIN0);
```

#### 函数 **syscfg\_enet\_phy\_interface\_config**

函数**syscfg\_enet\_phy\_interface\_config**描述见下表：

表 3-754. 函数 **syscfg\_enet\_phy\_interface\_config**

函数名称	syscfg_enet_phy_interface_config
函数原型	void syscfg_enet_phy_interface_config(uint32_t syscfg_enet_phy_interface);
功能描述	配置以太网PHY接口
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>syscfg_enet_phy_interface</b>	以太网MAC选择PHY接口
<i>SYSCFG_ENET_PHY_MII</i>	选择MII
<i>SYSCFG_ENET_PHY_RMII</i>	选择RMII
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	

-	-
---	---

例如：

```
/* configure the MII PHY interface for the ethernet MAC */
syscfg_enet_phy_interface_config(SYSCFG_ENET_PHY_MII);
```

### 函数 **syscfg\_compensation\_config**

函数**syscfg\_compensation\_config**描述见下表：

**表 3-755. 函数 syscfg\_compensation\_config**

函数名称	syscfg_compensation_config
函数原型	void syscfg_compensation_config(uint32_t syscfg_compensation);
功能描述	使能或禁用I/O补偿单元
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>syscfg_compensation</b>	I/O补偿单元使能或禁用状态
<b>SYSCFG_COMPENSATION_ENABLE</b>	I/O补偿单元使能
<b>SYSCFG_COMPENSATION_DISABLE</b>	I/O补偿单元掉电
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable the I/O compensation cell function */
syscfg_compensation_config(SYSCFG_COMPENSATION_ENABLE);
```

### 函数 **syscfg\_flag\_get**

函数**syscfg\_flag\_get**描述见下表：

**表 3-756. 函数 syscfg\_flag\_get**

函数名称	syscfg_flag_get
函数原型	FlagStatus syscfg_flag_get(void);
功能描述	获取I/O补偿单元状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如：

```
/* check whether the I/O compensation cell ready flag is set or not */
if(RESET != syscfg_flag_get());
```

## 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)，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 3-757. 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

寄存器名称	寄存器描述
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	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 3-758. 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
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

库函数名称	库函数描述
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
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	使能或禁能主通道输出
timer_channel_control_shadow_config	使能或禁能通道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
timer_channel_output_config	外设TIMERx的通道输出配置
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
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_	读取通道输入捕获值

库函数名称	库函数描述
register_read	
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重映射功能
timer_write_chxval_register_config	配置TIMERx写CHxVAL选择位
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 3-759. 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)
clockdivision	时钟分频因子 (TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4)
period	周期 (0~65535)

成员名称	功能描述
repetitioncounter	重复计数器值 (0~255)

### 结构体 **timer\_break\_parameter\_struct**

**表 3-760. 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 3-761. 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 3-762. 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)

成员名称	功能描述
	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 3-763. 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..13)	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 3-764. Function 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
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="#">表 3-759. 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 3-765. 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..13)	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="#">表 3-759. 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 3-766. 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..13)	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 3-767. 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..13)	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 3-768. 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..13)	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 3-769. 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..13)	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 3-770. 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..13)	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 3-771. 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..13)	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 3-772. 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	-
<b>输入参数{in}</b>	
timer_periph	TIMER外设
TIMERx(x=0..4,7..13)	TIMER外设选择
<b>输入参数{in}</b>	
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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-773. 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

<b>先决条件</b>	计数器设置为无中央对齐计数模式（边沿对齐模式）
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<b>TIMERx(x=0..4,7)</b>	TIMER外设选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-774. 函数 timer\_counter\_down\_direction**

<b>函数名称</b>	timer_counter_down_direction
<b>函数原型</b>	void timer_counter_down_direction(uint32_t timer_periph);
<b>功能描述</b>	设置外设TIMERx向下计数
<b>先决条件</b>	计数器设置为无中央对齐计数模式（边沿对齐模式）
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<b>TIMERx(x=0..4,7)</b>	TIMER外设选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-775. 函数 timer\_prescaler\_config**

<b>函数名称</b>	timer_prescaler_config
<b>函数原型</b>	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint8_t

	pscreload);
功能描述	配置外设TIMERx预分频器
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
<b>输入参数{in}</b>	
prescaler	预分频值, 0~65535
<b>输入参数{in}</b>	
pscreload	预分频值加载模式
TIMER_PSC_RELOAD_NOW	预分频值立即加载
TIMER_PSC_RELOAD_UPDATE	预分频值在下次更新事件发生时加载
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-776. 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	-
<b>输入参数{in}</b>	
timer_periph	TIMER外设
TIMERx(x=0..7)	TIMER外设选择
<b>输入参数{in}</b>	
repetition	重复计数器值, 取值范围0~255
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-777. 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..13)	TIMER外设选择
输入参数{in}	
autoreload	计数器自动重载值
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 3-778. 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..13)	TIMER外设选择
输入参数{in}	
counter	计数器值

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 3-779. 函数 timer\_counter\_read**

输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint32_t	外设TIMERx的计数器值

例如：

```
/* read TIMER0 counter value */

uint32_t i = 0;

i = timer_counter_read (TIMER0);
```

### 函数 **timer\_prescaler\_read**

函数timer\_prescaler\_read描述见下表：

**表 3-780. 函数 timer\_prescaler\_read**

输入参数{in}	
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	-

<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0..13)</i>	TIMER外设选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint16_t</b>	外设TIMERx的预分频器值 (0x0000~0xFFFF)

例如：

```
/* read TIMER0 prescaler value */
```

```
uint16_t i = 0;
```

```
i = timer_prescaler_read (TIMER0);
```

### 函数 **timer\_single\_pulse\_mode\_config**

函数**timer\_single\_pulse\_mode\_config**描述见下表：

**表 3-781. 函数 timer\_single\_pulse\_mode\_config**

<b>函数名称</b>	timer_single_pulse_mode_config
<b>函数原型</b>	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);
<b>功能描述</b>	配置外设TIMERx的单脉冲模式
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0..8,11)</i>	TIMER外设选择
<b>输入参数{in}</b>	
<b>spmode</b>	脉冲模式
<i>TIMER_SP_MODE_SINGLE</i>	单脉冲模式计数
<i>TIMER_SP_MODE_REPEATIVE</i>	重复模式计数
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-782. 函数 `timer_update_source_config`**

函数名称	<code>timer_update_source_config</code>
函数原型	<code>void timer_update_source_config(uint32_t timer_periph, uint32_t update);</code>
功能描述	配置外设TIMERx的更新源
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<code>timer_periph</code>	TIMER外设
<code>TIMERx(x=0..13)</code>	TIMER外设选择
<b>输入参数{in}</b>	
<code>update</code>	更新源
<code>TIMER_UPDATE_SRC_GLOBAL</code>	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
<code>TIMER_UPDATE_SRC_REGULAR</code>	只有计数器溢出/下溢才产生更新中断或DMA请求
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-783. 函数 `timer_dma_enable`**

函数名称	<code>timer_dma_enable</code>
函数原型	<code>void timer_dma_enable(uint32_t timer_periph, uint16_t dma);</code>
功能描述	外设TIMERx的DMA使能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<code>timer_periph</code>	TIMER外设
<code>TIMERx</code>	参考具体参数
<b>输入参数{in}</b>	

<b>dma</b>	DMA源
<i>TIMER_DMA_UPD</i>	更新DMA请求, TIMERx(x=0..7)
<i>TIMER_DMA_CH0_D</i>	通道0比较/捕获 DMA请求, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH1_D</i>	通道1比较/捕获 DMA请求, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH2_D</i>	通道2比较/捕获 DMA请求, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH3_D</i>	通道3比较/捕获 DMA请求, TIMERx(x=0..4,7)
<i>TIMER_DMA_CMT_D</i>	换相DMA更新请求, TIMERx(x=0..7)
<i>TIMER_DMA_TRG_D</i>	触发DMA请求使能, TIMERx(x=0..4,7)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-784. 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	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<b>TIMERx</b>	参考具体参数
<b>输入参数{in}</b>	
<b>dma</b>	DMA源
<i>TIMER_DMA_UPD</i>	更新DMA请求, TIMERx(x=0..7)
<i>TIMER_DMA_CH0_D</i>	通道0比较/捕获 DMA请求, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH1_D</i>	通道1比较/捕获 DMA请求, TIMERx(x=0..4,7)

<i>TIMER_DMA_CH2_D</i>	通道2比较/捕获 DMA请求, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH3_D</i>	通道3比较/捕获 DMA请求, TIMERx(x=0..4,7)
<i>TIMER_DMA_CMT_D</i>	换相DMA更新请求, TIMERx(x=0,7)
<i>TIMER_DMA_TRG_D</i>	触发DMA请求使能, TIMERx(x=0..4,7)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-785. 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	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0..4,7)</i>	TIMER外设选择
<b>输入参数{in}</b>	
<b>dma_request</b>	通道的DMA请求源选择
<i>TIMER_DMAREQUENT_CHANNELLEV</i>	当通道捕获/比较事件发生时, 发送通道n的DMA请求
<i>TIMER_DMAREQUEST_UPDATEEVENT</i>	当更新事件发生, 发送通道n的DMA请求
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-786. 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	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<b>TIMERx</b>	参考具体参数
<b>输入参数{in}</b>	
<b>dma_baseaddr</b>	DMA传输起始地址
<b>TIMER_DMACFG_DMATA_CTL0</b>	DMA传输起始地址：TIMER_DMACFG_DMATA_CTL0, TIMERx(x=0..4,7)
<b>TIMER_DMACFG_DMATA_CTL1</b>	DMA传输起始地址：TIMER_DMACFG_DMATA_CTL1, TIMERx(x=0..4,7)
<b>TIMER_DMACFG_DMATA_SMCFG</b>	DMA传输起始地址：TIMER_DMACFG_DMATA_SMCFG, TIMERx(x=0..4,7)
<b>TIMER_DMACFG_DMATA_DMAINTEN_N</b>	DMA传输起始地址：TIMER_DMACFG_DMATA_DMAINTEN, TIMERx(x=0..4,7)
<b>TIMER_DMACFG_DMATA_INTF</b>	DMA传输起始地址：TIMER_DMACFG_DMATA_INTF, TIMERx(x=0..4,7)
<b>TIMER_DMACFG_DMATA_SWEVG</b>	DMA传输起始地址：TIMER_DMACFG_DMATA_SWEVG, TIMERx(x=0..4,7)
<b>TIMER_DMACFG_DMATA_CHCTL0</b>	DMA传输起始地址：TIMER_DMACFG_DMATA_CHCTL0, TIMERx(x=0..4,7)
<b>TIMER_DMACFG_DMATA_CHCTL1</b>	DMA传输起始地址：TIMER_DMACFG_DMATA_CHCTL1, TIMERx(x=0..4,7)
<b>TIMER_DMACFG_DMATA_CHCTL2</b>	DMA传输起始地址：TIMER_DMACFG_DMATA_CHCTL2, TIMERx(x=0..4,7)
<b>TIMER_DMACFG_DMATA_CNT</b>	DMA传输起始地址：TIMER_DMACFG_DMATA_CNT, TIMERx(x=0..4,7)
<b>TIMER_DMACFG_DMATA_PSC</b>	DMA传输起始地址：TIMER_DMACFG_DMATA_PSC, TIMERx(x=0..4,7)

<i>DMATA_PSC</i>	
<i>TIMER_DMACFG_DMATA_CAR</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CAR</i> , <i>TIMERx(x=0..4,7)</i>
<i>TIMER_DMACFG_DMATA_CREP</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CREP</i> , <i>TIMERx(x=0,7)</i>
<i>TIMER_DMACFG_DMATA_CH0CV</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CH0CV</i> , <i>TIMERx(x=0..4,7)</i>
<i>TIMER_DMACFG_DMATA_CH1CV</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CH1CV</i> , <i>TIMERx(x=0..4,7)</i>
<i>TIMER_DMACFG_DMATA_CH2CV</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CH2CV</i> , <i>TIMERx(x=0..4,7)</i>
<i>TIMER_DMACFG_DMATA_CH3CV</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CH3CV</i> , <i>TIMERx(x=0..4,7)</i>
<i>TIMER_DMACFG_DMATA_CCHP</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CCHP</i> , <i>TIMERx(x=0,7)</i>
<i>TIMER_DMACFG_DMATA_DMACFG</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_DMACFG</i> , <i>TIMERx(x=0..4,7)</i>
<i>TIMER_DMACFG_DMATA_DMATB</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_DMATB</i> , <i>TIMERx(x=0..4,7)</i>
<b>输入参数{in}</b>	
<i>dma_lenth</i>	DMA传输长度
<i>TIMER_DMACFG_DMATC_xTRANSFER</i>	x=1..18, DMA传输 x 次
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-787. 函数 timer\_event\_software\_generate**

函数名称	<b>timer_event_software_generate</b>
函数原型	<b>void timer_event_software_generate(uint32_t timer_periph, uint16_t event);</b>
功能描述	软件产生事件
先决条件	-

<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<b>TIMERx</b>	参考具体参数
<b>输入参数{in}</b>	
<b>event</b>	事件源
<b>TIMER_EVENT_SR_C_UPG</b>	更新事件产生, TIMERx(x=0..13)
<b>TIMER_EVENT_SR_C_CH0G</b>	通道0捕获或比较事件发生, TIMERx(x=0..4,7..13)
<b>TIMER_EVENT_SR_C_CH1G</b>	通道1捕获或比较事件发生, TIMERx(x=0..4,7,8,11)
<b>TIMER_EVENT_SR_C_CH2G</b>	通道2捕获或比较事件发生, TIMERx(x=0..4,7)
<b>TIMER_EVENT_SR_C_CH3G</b>	通道3捕获或比较事件发生, TIMERx(x=0..4,7)
<b>TIMER_EVENT_SR_C_CMTG</b>	通道换相更新事件发生, TIMERx(x=0,7)
<b>TIMER_EVENT_SR_C_TRGG</b>	触发事件产生, TIMERx(x=0..4,7,8,11)
<b>TIMER_EVENT_SR_C_BRKG</b>	产生中止事件, TIMERx(x=0,7)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-788. 函数 timer\_break\_struct\_para\_init**

<b>函数名称</b>	timer_break_struct_para_init
<b>函数原型</b>	void timer_break_struct_para_init(timer_break_parameter_struct* breakpara);
<b>功能描述</b>	将TIMER中止功能参数结构体中所有参数初始化
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>breakpara</b>	中止功能配置结构体, 详见 <a href="#">表 3-760. 结构体</a>

<a href="#"><u>timer_break_parameter_struct</u></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 3-789. 函数 timer\_break\_config**

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)	TIMER外设选择
输入参数{in}	
breakpara	中止功能配置结构体，详见 <a href="#"><u>表 3-760. 结构体 timer_break_parameter_struct</u></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.idloffstate     = 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 3-790. 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}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* enable TIMER0 break function*/
timer_break_enable (TIMER0);

```

### 函数 **timer\_break\_disable**

函数timer\_break\_disable描述见下表:

**表 3-791. 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)	TIMER外设选择
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable TIMER0 break function*/
timer_break_disable (TIMER0);
```

#### 函数 **timer\_automatic\_output\_enable**

函数timer\_automatic\_output\_enable描述见下表：

**表 3-792. 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)	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 3-793. 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)	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 3-794. 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)	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 3-795. 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	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<b>TIMERx(x=0,7)</b>	TIMER外设选择
<b>输入参数{in}</b>	
<b>newvalue</b>	控制状态
<b>ENABLE</b>	使能
<b>DISABLE</b>	禁能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-796. 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, uint8_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	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<b>TIMERx(x=0,7)</b>	TIMER外设选择
<b>输入参数{in}</b>	
<b>ccuctl</b>	通道换相控制影子寄存器更新控制
<b>TIMER_UPDATEECT_L_CCU</b>	CMTG位被置1时更新影子寄存器
<b>TIMER_UPDATEECT_L_CCUTRI</b>	当CMTG位被置1或检测到TRIGI上升沿时，影子寄存器更新
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure TIMER0 channel control shadow register update when CMTG bit is set */

timer_channel_control_shadow_update_config (TIMER0, TIMER_UPDATECTL_CCU);
```

### **函数 timer\_channel\_output\_struct\_para\_init**

函数timer\_channel\_output\_struct\_para\_init描述见下表：

**表 3-797. 函数 timer\_channel\_output\_struct\_para\_init**

函数名称	timer_channel_output_struct_para_init
函数原型	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpara);
功能描述	将TIMER通道输出参数结构体中所有参数初始化
先决条件	-
被调用函数	-
输入参数{in}	
ocpara	输出通道结构体，详见 <a href="#">表 3-761. 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 3-798. 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* ocpa);
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}	

<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0, TIMERx(x=0..4,7..13)
<i>TIMER_CH_1</i>	通道1, TIMERx(x=0..4,7,8,11)
<i>TIMER_CH_2</i>	通道2, TIMERx(x=0..4,7)
<i>TIMER_CH_3</i>	通道3, TIMERx(x=0..4,7)
<b>输入参数{in}</b>	
<b>ocpara</b>	输出通道结构体, 详见 <a href="#">表 3-761. 结构体timer_oc_parameter_struct</a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-799. 函数 timer\_channel\_output\_mode\_config**

<b>函数名称</b>	timer_channel_output_mode_config
<b>函数原型</b>	void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint16_t ocmode);
<b>功能描述</b>	配置外设TIMERx通道输出比较模式
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
<b>输入参数{in}</b>	
<b>channel</b>	待配置通道

<i>TIMER_CH_0</i>	通道0, TIMERx (x=0..4,7..13)
<i>TIMER_CH_1</i>	通道1, TIMERx (x=0..4,7,8,11)
<i>TIMER_CH_2</i>	通道2, TIMERx (x=0..4,7)
<i>TIMER_CH_3</i>	通道3, TIMERx (x=0..4,7)
<b>输入参数{in}</b>	
<i>ocmode</i>	通道输出比较模式
<i>TIMER_OC_MODE_TIMING</i>	冻结模式
<i>TIMER_OC_MODE_ACTIVE</i>	匹配时设置为高
<i>TIMER_OC_MODE_INACTIVE</i>	匹配时设置为低
<i>TIMER_OC_MODE_TOGGLE</i>	匹配时翻转
<i>TIMER_OC_MODE_LOW</i>	强制为低
<i>TIMER_OC_MODE_HIGH</i>	强制为高
<i>TIMER_OC_MODE_PWM0</i>	PWM模式0
<i>TIMER_OC_MODE_PWM1</i>	PWM模式1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-800. 函数 **timer\_channel\_output\_pulse\_value\_config**

函数名称	<code>timer_channel_output_pulse_value_config</code>
函数原型	<code>void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse);</code>
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	-
<b>输入参数{in}</b>	

<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
<b>输入参数{in}</b>	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0, TIMERx (x=0..4,7..13)
<i>TIMER_CH_1</i>	通道1, TIMERx (x=0..4,7,8,11)
<i>TIMER_CH_2</i>	通道2, TIMERx (x=0..4,7)
<i>TIMER_CH_3</i>	通道3, TIMERx (x=0..4,7)
<b>输入参数{in}</b>	
<b>pulse</b>	通道输出比较值 (0~65535)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure TIMER0 channel 0 output pulse value */

timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

#### 函数 **timer\_channel\_output\_shadow\_config**

函数timer\_channel\_output\_shadow\_config描述见下表：

**表 3-801. 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	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
<b>输入参数{in}</b>	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0, TIMERx (x=0..4,7..13)
<i>TIMER_CH_1</i>	通道1, TIMERx (x=0..4,7,8,11)
<i>TIMER_CH_2</i>	通道2, TIMERx (x=0..4,7)
<i>TIMER_CH_3</i>	通道3, TIMERx (x=0..4,7)
<b>输入参数{in}</b>	
<b>ocshadow</b>	输出比较影子寄存器功能状态
<i>TIMER_OC_SHAD</i> <i>OW_ENABLE</i>	使能输出比较影子寄存器

<i>TIMER_OC_SHAD OW_DISABLE</i>	禁能输出比较影子寄存器
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-802. 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	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<b>TIMERx</b>	参考具体参数
<b>输入参数{in}</b>	
<b>channel</b>	待配置通道
<b>TIMER_CH_0</b>	通道0, TIMERx (x=0..4,7..13)
<b>TIMER_CH_1</b>	通道1, TIMERx (x=0..4,7,8,11)
<b>TIMER_CH_2</b>	通道2, TIMERx (x=0..4,7)
<b>TIMER_CH_3</b>	通道3, TIMERx (x=0..4,7)
<b>输入参数{in}</b>	
<b>ocfast</b>	通道输出比较快速功能状态
<b>TIMER_OC_FAST_ENABLE</b>	通道输出比较快速功能使能
<b>TIMER_OC_FAST_DISABLE</b>	通道输出比较快速功能禁能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-803. 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TIMERx的通道输出比较清0功能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
timer_periph	TIMER外设
TIMERx	参考具体参数
<b>输入参数{in}</b>	
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)
<b>输入参数{in}</b>	
occlear	通道比较输出清0功能状态
TIMER_OC_CLEAR_ENABLE	通道比较输出清0功能使能
TIMER_OC_CLEAR_DISABLE	通道比较输出清0功能禁能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure TIMER0 channel 0 output clear function */

timer_channel_output_clear_config (TIMER0, TIMER_CH_0,
TICKER_OC_CLEAR_ENABLE);
```

### 函数 timer\_channel\_output\_polarity\_config

函数timer\_channel\_output\_polarity\_config描述见下表：

**表 3-804. 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}	
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}	
ocpolarity	通道输出极性
TIMER_OC_POLARITY_HIGH	通道输出极性高电平有效
TIMER_OC_POLARITY_LOW	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 3-805. 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);

<b>功能描述</b>	互补通道输出极性配置
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<b>TIMERx(x=0,7)</b>	TIMER外设选择
<b>输入参数{in}</b>	
<b>channel</b>	待配置通道
<b>TIMER_CH_0</b>	通道0
<b>TIMER_CH_1</b>	通道1
<b>TIMER_CH_2</b>	通道2
<b>输入参数{in}</b>	
<b>ocpolarity</b>	互补通道输出极性
<b>TIMER_OCN_POLARITY_HIGH</b>	互补通道输出极性高电平有效
<b>TIMER_OCN_POLARITY_LOW</b>	互补通道输出极性低电平有效
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-806. 函数 timer\_channel\_output\_state\_config**

<b>函数名称</b>	timer_channel_output_state_config
<b>函数原型</b>	void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
<b>功能描述</b>	配置通道状态
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<b>TIMERx</b>	参考具体参数
<b>输入参数{in}</b>	
<b>channel</b>	待配置通道

<i>TIMER_CH_0</i>	通道0, TIMERx (x=0..4,7..13)
<i>TIMER_CH_1</i>	通道1, TIMERx (x=0..4,7,8,11)
<i>TIMER_CH_2</i>	通道2, TIMERx (x=0..4,7)
<i>TIMER_CH_3</i>	通道3, TIMERx (x=0..4,7)
<b>输入参数{in}</b>	
<b>state</b>	通道状态
<i>TIMER_CCX_ENAB</i> <i>LE</i>	通道使能
<i>TIMER_CCX_DISA</i> <i>BLE</i>	通道禁能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-807. 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	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,7)</i>	TIMER外设选择
<b>输入参数{in}</b>	
<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
<b>输入参数{in}</b>	
<b>state</b>	互补通道状态
<i>TIMER_CCXN_ENA</i> <i>BLE</i>	互补通道使能
<i>TIMER_CCXN_DIS</i>	互补通道禁能

<i>ABLE</i>	
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-808. 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
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>icpara</b>	通道输入结构体，详见 <a href="#">表 3-762. 结构体timer_ic_parameter_struct</a> 。
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-809. 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	-

<b>被调用函数</b>	timer_channel_input_capture_prescaler_config
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
<b>输入参数{in}</b>	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0, TIMERx (x=0..4,7..13)
<i>TIMER_CH_1</i>	通道1, TIMERx (x=0..4,7,8,11)
<i>TIMER_CH_2</i>	通道2, TIMERx (x=0..4,7)
<i>TIMER_CH_3</i>	通道3, TIMERx (x=0..4,7)
<b>输入参数{in}</b>	
<b>icpara</b>	输入捕获结构体, 详见 <a href="#">表 3-762. 结构体timer_ic_parameter_struct</a> 。
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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_DIRECTTI;
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 3-810. 函数 timer\_channel\_input\_capture\_prescaler\_config**

<b>函数名称</b>	timer_channel_input_capture_prescaler_config
<b>函数原型</b>	void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler);
<b>功能描述</b>	配置TIMERx通道输入捕获预分频值
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	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, TIMERx (x=0..4,7..13)
<i>TIMER_CH_1</i>	通道1, TIMERx (x=0..4,7,8,11)
<i>TIMER_CH_2</i>	通道2, TIMERx (x=0..4,7)
<i>TIMER_CH_3</i>	通道3, TIMERx (x=0..4,7)
输入参数{in}	
<b>prescaler</b>	通道输入捕获预分频值
<i>TIMER_IC_PSC_DI</i> V1	不分频
<i>TIMER_IC_PSC_DI</i> V2	2分频
<i>TIMER_IC_PSC_DI</i> V4	4分频
<i>TIMER_IC_PSC_DI</i> V8	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,
TMR_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 3-811. 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}	
<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, TIMERx (x=0..4,7..13)

<i>TIMER_CH_1</i>	通道1, TIMERx (x=0..4,7,8,11)
<i>TIMER_CH_2</i>	通道2, TIMERx (x=0..4,7)
<i>TIMER_CH_3</i>	通道3, TIMERx (x=0..4,7)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<i>uint32_t</i>	通道输入捕获值, (0x0000~0xFFFF)

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 3-812. 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
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER外设选择
<b>输入参数{in}</b>	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
<b>输入参数{in}</b>	
<b>icpwm</b>	输入捕获结构体, 详见 <a href="#">表 3-762. 结构体timer_ic_parameter_struct</a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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_DIRECTTI;
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 3-813. 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, uint8_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}	
timer_periph	TIMER外设
TIMERx(x=0..4,7)	TIMER外设选择
输入参数{in}	
hallmode	HALL接口功能状态
TIMER_HALLINTERFACE_ENABLE	HALL接口使能
TIMER_HALLINTERFACE_DISABLE	HALL接口禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* configure TIMER0 hall sensor mode */

timer_hall_mode_config(TIMER0, TIMER_HALLINTERFACE_ENABLE);

```

### **函数 timer\_input\_trigger\_source\_select**

函数timer\_input\_trigger\_source\_select描述见下表:

**表 3-814. 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);

<b>功能描述</b>	TIMERx的输入触发源选择
<b>先决条件</b>	SMC[2:0] = 000
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<b>TIMERx(x=0..4,7,8,11)</b>	TIMER外设选择
<b>输入参数{in}</b>	
<b>intrigger</b>	待选择的触发源
<b>TIMER_SMCFG_T_RGSEL_ITI0</b>	内部触发输入0(ITIO, TIMERx(x=0..4,7,8,11))
<b>TIMER_SMCFG_T_RGSEL_ITI1</b>	内部触发输入1(ITI1, TIMERx(x=0..4,7,8,11))
<b>TIMER_SMCFG_T_RGSEL_ITI2</b>	内部触发输入2(ITI2, TIMERx(x=0..4,7,8,11))
<b>TIMER_SMCFG_T_RGSEL_ITI3</b>	内部触发输入3(ITI3, TIMERx(x=0..4,7,8,11))
<b>TIMER_SMCFG_T_RGSEL_CI0F_ED</b>	CI0的边沿标志位 (CI0F_ED, TIMERx(x=0..4,7,8,11))
<b>TIMER_SMCFG_T_RGSEL_CI0FE0</b>	滤波后的通道0输入 (CI0FE0, TIMERx(x=0..4,7,8,11))
<b>TIMER_SMCFG_T_RGSEL_CI1FE1</b>	滤波后的通道1输入(CI1FE1, TIMERx(x=0..4,7,8,11))
<b>TIMER_SMCFG_T_RGSEL_ETIFP</b>	滤波后的外部触发输入(ETIFP, TIMERx(x=0..4,7))
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-815. 函数 timer\_master\_output\_trigger\_source\_select**

<b>函数名称</b>	timer_master_output_trigger_source_select
<b>函数原型</b>	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
<b>功能描述</b>	选择TIMERx主模式输出触发

<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<b>TIMERx(x=0..7)</b>	TIMER外设选择
<b>输入参数{in}</b>	
<b>outtrigger</b>	主模式输出触发
<b>TIMER_TRI_OUT_SRC_RESET</b>	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。
<b>TIMER_TRI_OUT_SRC_ENABLE</b>	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。
<b>TIMER_TRI_OUT_SRC_UPDATE</b>	更新。主模式控制器选择更新事件作为TRGO。
<b>TIMER_TRI_OUT_SRC_CC0</b>	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
<b>TIMER_TRI_OUT_SRC_O0CPRE</b>	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
<b>TIMER_TRI_OUT_SRC_O1CPRE</b>	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
<b>TIMER_TRI_OUT_SRC_O2CPRE</b>	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
<b>TIMER_TRI_OUT_SRC_O3CPRE</b>	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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-816. 函数 timer\_slave\_mode\_select**

<b>函数名称</b>	timer_slave_mode_select
<b>函数原型</b>	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
<b>功能描述</b>	TIMERx从模式配置

<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<b>TIMERx(x=0..4,7,8,11)</b>	TIMER外设选择
<b>输入参数{in}</b>	
<b>slavemode</b>	从模式
<b>TIMER_SLAVE_MODE_DISABLE</b>	关闭从模式 (TIMERx (x=0..4,7,8,11))
<b>TIMER_ENCODER_MODE0</b>	编码器模式0 (TIMERx (x=0..4,7))
<b>TIMER_ENCODER_MODE1</b>	编码器模式1 (TIMERx (x=0..4,7))
<b>TIMER_ENCODER_MODE2</b>	编码器模式2 (TIMERx (x=0..4,7))
<b>TIMER_SLAVE_MODE_RESTART</b>	复位模式 (TIMERx (x=0..4,7,8,11))
<b>TIMER_SLAVE_MODE_PAUSE</b>	暂停模式 (TIMERx (x=0..4,7,8,11))
<b>TIMER_SLAVE_MODE_EVENT</b>	事件模式 (TIMERx (x=0..4,7,8,11))
<b>TIMER_SLAVE_MODE_EXTERNAL0</b>	外部时钟模式0 (TIMERx (x=0..4,7,8,11))
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* select TIMER0 slave mode */

timer_slave_mode_select(TIMER0, TIMER_ENCODER_MODE0);
```

#### 函数 **timer\_master\_slave\_mode\_config**

函数**timer\_master\_slave\_mode\_config**描述见下表：

**表 3-817. 函数 timer\_master\_slave\_mode\_config**

<b>函数名称</b>	timer_master_slave_mode_config
<b>函数原型</b>	void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t masterslave);
<b>功能描述</b>	TIMERx主从模式配置
<b>先决条件</b>	-

<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER外设选择
<b>输入参数{in}</b>	
<b>masterslave</b>	主从模式使能状态
<i>TIMER_MASTER_SLAVE_MODE_ENA_BLE</i>	主从模式使能
<i>TIMER_MASTER_SLAVE_MODE_DISABLE_BLE</i>	主从模式禁能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-818. 函数 timer\_external\_trigger\_config**

<b>函数名称</b>	timer_external_trigger_config
<b>函数原型</b>	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>功能描述</b>	配置TIMERx外部触发输入
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0..4,7)</i>	TIMER外设选择
<b>输入参数{in}</b>	
<b>extprescaler</b>	外部触发预分频
<i>TIMER_EXT_TRI_P_SC_OFF</i>	不分频
<i>TIMER_EXT_TRI_P_SC_DIV2</i>	2分频
<i>TIMER_EXT_TRI_P</i>	4分频

<i>SC_DIV4</i>	
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV8</i>	8分频
<b>输入参数{in}</b>	
<b>expolarity</b>	外部触发输入极性
<i>TIMER_ETP_FALLING</i>	低电平或者下降沿有效
<i>TIMER_ETP_RISING</i>	高电平或者上升沿有效
<b>输入参数{in}</b>	
<b>extfilter</b>	外部触发滤波控制 (0~15)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure TIMER0 external trigger input */

timer_external_trigger_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,
    TIMER_ETP_FALLING, 10);
```

### 函数 **timer\_quadrature\_decoder\_mode\_config**

函数**timer\_quadrature\_decoder\_mode\_config**描述见下表：

**表 3-819. 函数 timer\_quadrature\_decoder\_mode\_config**

函数名称	<code>timer_quadrature_decoder_mode_config</code>
函数原型	<code>void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode,uint16_t ic0polarity, uint16_t ic1polarity);</code>
功能描述	TIMERx配置为编码器模式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0..4,7)</i>	TIMER外设选择
<b>输入参数{in}</b>	
<b>decomode</b>	编码器模式
<i>TIMER_ENCODER_MODE0</i>	根据CI0FE0的电平，计数器在CI1FE1的边沿向上/下计数
<i>TIMER_ENCODER_MODE1</i>	根据CI1FE1的电平，计数器在CI0FE0的边沿向上/下计数
<i>TIMER_ENCODER_MODE2</i>	根据另一个信号的输入电平，计数器在CI0FE0和CI1FE1的边沿向上/下计数

输入参数{in}	
<b>ic0polarity</b>	IC0极性
<i>TIMER_IC_POLARITY_RISING</i>	捕获上升边沿
<i>TIMER_IC_POLARITY_FALLING</i>	捕获下降边沿
输入参数{in}	
<b>ic1polarity</b>	IC1极性
<i>TIMER_IC_POLARITY_RISING</i>	捕获上升边沿
<i>TIMER_IC_POLARITY_FALLING</i>	捕获下降边沿
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 quadrature decoder mode */

timer_quadrature_decoder_mode_config (TIMER0, TIMER_ENCODER_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 3-820. 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}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0..4,7,8,11)</i>	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 3-821. 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
<b>输入参数{in}</b>	
timer_periph	TIMER外设
TIMERx(x=0..4,7,8, 11)	TIMER外设选择
<b>输入参数{in}</b>	
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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* configure TIMER0 the internal trigger ITI0 as external clock input */

timer_internal_trigger_as_external_clock_config (TIMER0, TIMER_SMCFG_TRGSEL_ITI0);
```

### 函数 **timer\_external\_trigger\_as\_external\_clock\_config**

函数timer\_external\_trigger\_as\_external\_clock\_config描述见下表:

**表 3-822. 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 exttrigger, uint16_t expolarity, uint32_t extfilter);

<b>功能描述</b>	配置TIMERx的外部触发作为时钟源
<b>先决条件</b>	-
<b>被调用函数</b>	timer_input_trigger_source_select
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0..4,7,8, 11)</i>	TIMER外设选择
<b>输入参数{in}</b>	
<b>extrigger</b>	外部触发源
<i>TIMER_SMCFG_T RGSEL_CI0F_ED</i>	CI0的边沿标志(CI0F_ED)
<i>TIMER_SMCFG_T RGSEL_CI0FE0</i>	滤波后的通道0输入(CI0FE0)
<i>TIMER_SMCFG_T RGSEL_CI1FE1</i>	滤波后的通道1输入(CI1FE1)
<b>输入参数{in}</b>	
<b>expolarity</b>	外部触发源极性
<i>TIMER_IC_POLARITY_RISING</i>	外部触发源高电平或者上升沿有效
<i>TIMER_IC_POLARITY_FALLING</i>	外部触发源低电平或者下降沿有效
<b>输入参数{in}</b>	
<b>extfilter</b>	滤波参数(0~15)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-823. 函数 **timer\_external\_clock\_mode0\_config**

<b>函数名称</b>	timer_external_clock_mode0_config
<b>函数原型</b>	void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>功能描述</b>	配置TIMERx外部时钟模式0，ETI作为时钟源
<b>先决条件</b>	-

<b>被调用函数</b>	timer_external_trigger_config
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0..4,7,8, 11)</i>	TIMER外设选择
<b>输入参数{in}</b>	
<b>extprescaler</b>	ETI触发源预分频值
<i>TIMER_EXT_TRI_P_SC_OFF</i>	不分频
<i>TIMER_EXT_TRI_P_SC_DIV2</i>	2分频
<i>TIMER_EXT_TRI_P_SC_DIV4</i>	4分频
<i>TIMER_EXT_TRI_P_SC_DIV8</i>	8分频
<b>输入参数{in}</b>	
<b>expolarity</b>	ETI触发源极性
<i>TIMER_ETP_FALLING</i>	下降沿或者低电平有效
<i>TIMER_ETP_RISING</i>	上升沿或者高电平有效
<b>输入参数{in}</b>	
<b>extfilter</b>	ETI触发源滤波参数 (0~15)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure TIMER0 the external clock mode0 */
timer_external_clock_mode0_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TMR_ETP_FALLING, 0);
```

#### 函数 **timer\_external\_clock\_mode1\_config**

函数**timer\_external\_clock\_mode1\_config**描述见下表：

**表 3-824. 函数 timer\_external\_clock\_mode1\_config**

<b>函数名称</b>	timer_external_clock_mode1_config
<b>函数原型</b>	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>功能描述</b>	配置TIMERx外部时钟模式1
<b>先决条件</b>	-

<b>被调用函数</b>	timer_external_trigger_config
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0..4,7)</i>	TIMER外设选择
<b>输入参数{in}</b>	
<b>extprescaler</b>	ETI触发源预分频值
<i>TIMER_EXT_TRI_P SC_OFF</i>	不分频
<i>TIMER_EXT_TRI_P SC_DIV2</i>	2分频
<i>TIMER_EXT_TRI_P SC_DIV4</i>	4分频
<i>TIMER_EXT_TRI_P SC_DIV8</i>	8分频
<b>输入参数{in}</b>	
<b>expolarity</b>	ETI触发源极性
<i>TIMER_ETP_FALLING</i>	下降沿或者低电平有效
<i>TIMER_ETP_RISING</i>	上升沿或者高电平有效
<b>输入参数{in}</b>	
<b>extfilter</b>	ETI触发源滤波参数 (0~15)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-825. 函数 **timer\_external\_clock\_mode1\_disable**

<b>函数名称</b>	timer_external_clock_mode1_disable
<b>函数原型</b>	void timer_external_clock_mode1_disable(uint32_t timer_periph);
<b>功能描述</b>	TIMERx外部时钟模式1禁能
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	

<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0..4,7)</i>	TIMER外设选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-826. 函数 timer\_channel\_remap\_config**

<b>函数名称</b>	<b>timer_channel_remap_config</b>
<b>函数原型</b>	void timer_channel_remap_config(uint32_t timer_periph, uint32_t remap);
<b>功能描述</b>	配置TIMERx通道重映射功能
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=1,4,10)</i>	TIMER外设选择
<b>输入参数{in}</b>	
<b>remap</b>	TIMER重映射选择
<i>TIMER1_ITI1_RMP</i> <i>_TIMER7_TRGO</i>	TIMER1内部触发输入1映射到TIMER7到TRGO
<i>TIMER1_ITI1_RMP</i> <i>_ETHERNET_PTP</i>	TIMER1内部触发输入1映射到ETHERNET_PTP
<i>TIMER1_ITI1_RMP</i> <i>_USB_FS_SOF</i>	TIMER1内部触发输入1映射到USB_FS_SOF
<i>TIMER1_ITI1_RMP</i> <i>_USB_HS_SOF</i>	TIMER1内部触发输入1映射到USB_HS_SOF
<i>TIMER4_CI3_RMP</i> <i>_GPIO</i>	TIMER4通道3输入映射到GPIO引脚
<i>TIMER4_CI3_RMP</i> <i>_IRC32K</i>	TIMER4通道3输入映射到内部低速32K时钟
<i>TIMER4_CI3_RMP</i> <i>_LXTAL</i>	TIMER4通道3输入映射到外部高速时钟
<i>TIMER4_CI3_RMP</i> <i>_RTC_WAKEUP_INT</i>	TIMER4通道3输入映射到RTC唤醒中断
<i>TIMER10_ITI1_RM</i>	TIMER10内部触发映射到GPIO

<i>P_GPIO</i>	
<i>TIMER10_ITI1_RM</i> <i>P_RTC_HXTAL_DIV</i>	TIMER10内部触发映射到HXTAL_DIV (RTC时钟, 是HXTAL时钟经过RCU_CFG0 寄存器中RTCDIV的位分频后得到到)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure timer1 internal trigger input1 remap to TIMER7_TRGO */
timer_channel_remap_config(TIMER1,TIMER1_ITI1_RMP_TIMER7_TRGO);
```

#### 函数 **timer\_write\_chxval\_register\_config**

函数timer\_write\_chxval\_register\_config描述见下表：

表 3-827. 函数 **timer\_write\_chxval\_register\_config**

函数名称	timer_write_chxval_register_config
函数原型	void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);
功能描述	配置TIMERx写CHxVAL选择位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<i>timer_periph</i>	TIMER外设
<i>TIMERx(x=0..4,7..3)</i>	TIMER外设选择
<b>输入参数{in}</b>	
<i>ccsel</i>	写CHxVAL寄存器选择位
<i>TIMER_CHVSEL_DENABLE</i>	无影响
<i>TIMER_CHVSEL_ENABLE</i>	当写入捕获比较寄存器的值与寄存器当前值相等时，写入操作无效。
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-828. 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}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx (x=0,7)</i>	TIMER外设选择
输入参数{in}	
<b>ccsel</b>	输出值选择位
<i>TIMER_OUTSEL_D_ISABLE</i>	无影响
<i>TIMER_OUTSEL_E_NABLE</i>	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 3-829. 函数 timer\_flag\_get**

函数名称	timer_flag_get
函数原型	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);
功能描述	获取外设TIMERx的状态标志
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
<b>flag</b>	状态标志

<i>TIMER_FLAG_UP</i>	更新标志, TIMERx(x=0..13)
<i>TIMER_FLAG_CH0</i>	通道0比较/捕获标志, TIMERx(x=0..4,7..13)
<i>TIMER_FLAG_CH1</i>	通道1比较/捕获标志, TIMERx(x=0..4,7,8,11)
<i>TIMER_FLAG_CH2</i>	通道2比较/捕获标志, TIMERx(x=0..4,7)
<i>TIMER_FLAG_CH3</i>	通道3比较/捕获标志, TIMERx(x=0..4,7)
<i>TIMER_FLAG_CMT</i>	通道换相更新标志, TIMERx(x=0,7)
<i>TIMER_FLAG_TRG</i>	触发标志, TIMERx(x=0,7,8,11)
<i>TIMER_FLAG_BRK</i>	中止标志位, TIMERx(x=0,7)
<i>TIMER_FLAG_CH0_O</i>	通道0捕获溢出标志, TIMERx(x=0..4,7..11)
<i>TIMER_FLAG_CH1_O</i>	通道1捕获溢出标志, TIMERx(x=0..4,7,8,11)
<i>TIMER_FLAG_CH2_O</i>	通道2捕获溢出标志, TIMERx(x=0..4,7)
<i>TIMER_FLAG_CH3_O</i>	通道3捕获溢出标志, TIMERx(x=0..4,7)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	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 3-830. 函数 **timer\_flag\_clear**

函数名称	timer_flag_clear
函数原型	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
功能描述	清除外设TIMERx状态标志
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
<b>输入参数{in}</b>	
<b>flag</b>	状态标志
<i>TIMER_FLAG_UP</i>	更新标志, TIMERx(x=0..13)

<i>TIMER_FLAG_CH0</i>	通道0比较/捕获标志, TIMERx(x=0..4,7..13)
<i>TIMER_FLAG_CH1</i>	通道1比较/捕获标志, TIMERx(x=0..4,7,8,11)
<i>TIMER_FLAG_CH2</i>	通道2比较/捕获标志, TIMERx(x=0..4,7)
<i>TIMER_FLAG_CH3</i>	通道3比较/捕获标志, TIMERx(x=0..4,7)
<i>TIMER_FLAG_CMT</i>	通道换相更新标志, TIMERx(x=0,7)
<i>TIMER_FLAG_TRG</i>	触发标志, TIMERx(x=0,7,8,11)
<i>TIMER_FLAG_BRK</i>	中止标志位, TIMERx(x=0,7)
<i>TIMER_FLAG_CH0_O</i>	通道0捕获溢出标志, TIMERx(x=0..4,7..11)
<i>TIMER_FLAG_CH1_O</i>	通道1捕获溢出标志, TIMERx(x=0..4,7,8,11)
<i>TIMER_FLAG_CH2_O</i>	通道2捕获溢出标志, TIMERx(x=0..4,7)
<i>TIMER_FLAG_CH3_O</i>	通道3捕获溢出标志, TIMERx(x=0..4,7)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-831. 函数 timer\_interrupt\_enable**

函数名称	<b>timer_interrupt_enable</b>
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	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
<b>输入参数{in}</b>	
<b>interrupt</b>	中断源
<i>TIMER_INT_UP</i>	更新中断, TIMERx(x=0..13)
<i>TIMER_INT_CH0</i>	通道0比较/捕获中断, TIMERx(x=0..4,7..13)
<i>TIMER_INT_CH1</i>	通道1比较/捕获中断, TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_CH2</i>	通道2比较/捕获中断, TIMERx(x=0..4,7)

<i>TIMER_INT_CH3</i>	通道3比较/捕获中断, TIMERx(x=0..4,7)
<i>TIMER_INT_CMT</i>	换相更新中断, TIMERx(x=0,7)
<i>TIMER_INT_TRG</i>	触发中断, TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_BRK</i>	中止中断, TIMERx(x=0,7)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-832. 函数 timer\_interrupt\_disable**

函数名称	timer_interrupt_disable
函数原型	void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt);
功能描述	外设TIMERx中断禁能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<i>timer_periph</i>	TIMER外设
<i>TIMERx</i>	参考具体参数
<b>输入参数{in}</b>	
<i>interrupt</i>	中断源
<i>TIMER_INT_UP</i>	更新中断, TIMERx(x=0..13)
<i>TIMER_INT_CH0</i>	通道0比较/捕获中断, TIMERx(x=0..4,7..13)
<i>TIMER_INT_CH1</i>	通道1比较/捕获中断, TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_CH2</i>	通道2比较/捕获中断, TIMERx(x=0..4,7)
<i>TIMER_INT_CH3</i>	通道3比较/捕获中断, TIMERx(x=0..4,7)
<i>TIMER_INT_CMT</i>	换相更新中断, TIMERx(x=0,7)
<i>TIMER_INT_TRG</i>	触发中断, TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_BRK</i>	中止中断, TIMERx(x=0,7)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* disable the TIMER0 update interrupt */
```

```
timer_interrupt_disable (TIMER0, TIMER_INT_UP);
```

### 函数 timer\_interrupt\_flag\_get

函数timer\_interrupt\_flag\_get描述见下表:

**表 3-833. 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 interrupt);
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}	
interrupt	中断源
TIMER_INT_FLAG_UP	更新中断, TIMERx(x=0..13)
TIMER_INT_FLAG_CH0	通道0比较/捕获中断, TIMERx(x=0..4,7..13)
TIMER_INT_FLAG_CH1	通道1比较/捕获中断, TIMERx(x=0..4,7,8,11)
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)
TIMER_INT_FLAG_BRK	中止中断, TIMERx(x=0,7)
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如:

```
/* get TIMER0 update interrupt flag */
FlagStatus Flag_interrupt = RESET;
Flag_interrupt = timer_interrupt_flag_get (TIMER0, TIMER_INT_FLAG_UP);
```

### 函数 timer\_interrupt\_flag\_clear

函数timer\_interrupt\_flag\_clear描述见下表：

**表 3-834. 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 interrupt);
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}	
interrupt	中断源
TIMER_INT_FLAG_UP	更新中断, TIMERx(x=0..13)
TIMER_INT_FLAG_CH0	通道0比较/捕获中断, TIMERx(x=0..4,7..13)
TIMER_INT_FLAG_CH1	通道1比较/捕获中断, TIMERx(x=0..4,7,8,11)
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)
TIMER_INT_FLAG_BRK	中止中断, TIMERx(x=0,7)
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. TLI

TLI(TFT-LCD 接口)连接同步的 LCD 接口，并且为无源 LCD 显示屏提供像素数据，时钟以及时序信号。TLI 计算器列举在章节 [3.27.1](#)，TLI 固件库函数介绍在章节 [3.27.2](#)。

### 3.27.1. 外设寄存器说明

TLI 寄存器列表如下表所示：

**表 3-835. TLI 寄存器**

寄存器名称	寄存器描述
TLI_SPSZ	TLI同步脉冲宽度寄存器
TLI_BPSZ	TLI后沿宽度寄存器
TLI_ASZ	TLI有效宽度寄存器
TLI_TSZ	TLI总宽度寄存器
TLI_CTL	TLI控制寄存器
TLI_RL	TLI重载配置寄存器
TLI_BGC	TLI背景色配置寄存器
TLI_INTEN	TLI中断使能寄存器
TLI_INTF	TLI中断标志寄存器
TLI_INTC	TLI中断标志清除寄存器
TLI_LM	TLI行标记寄存器
TLI_CPOOS	TLI当前像素寄存器
TLI_STAT	TLI状态寄存器
TLI_LxCTL	TLI x层控制寄存器
TLI_LxHPOS	TLI x层水平位置参数寄存器
TLI_LxVPOS	TLI x层垂直位置参数寄存器
TLI_LxCKEY	TLI x层色键值寄存器
TLI_LxPPF	TLI x层像素格式寄存器
TLI_LxSA	TLI x层特定alpha寄存器
TLI_LxDC	TLI x层默认颜色寄存器

寄存器名称	寄存器描述
TLI_LxBLEND	TLI x层混合寄存器
TLI_LxFBADDR	TLI x层帧地址寄存器
TLI_LxFLEN	TLI x层行数寄存器
TLI_LxFTLN	TLI x层帧总行数寄存器
TLI_LxLUT	TLI x层颜色查找表寄存器

### 3.27.2. 外设库函数说明

TLI 库函数列表如下表所示：

**表 3-836. TLI 库函数**

库函数名称	库函数描述
tli_deinit	复位TLI
tli_struct_para_init	用默认值初始化TLI参数结构体，建议在定义一个tli_parameter_struct结构体后调用该接口实现对结构体的初始化
tli_init	初始化TLI
tli_dither_config	配置TLI抖动功能
tli_enable	使能TLI
tli_disable	禁能TLI
tli_reload_config	配置TLI重载模式
tli_layer_struct_para_init	用默认参数初始化TLI层结构体，建议在定义一个tli_layer_parameter_struct结构体后调用该接口实现对结构体的初始化
tli_layer_init	初始化TLI层
tli_layer_window_offset_modify	重新配置窗口位置
tli_lut_struct_para_init	用默认参数初始化TLI层LUT结构体，建议在定义一个tli_layer_lut_parameter_struct结构体后调用该接口实现对结构体的初始化
tli_lut_init	初始化TLI层颜色查表
tli_color_key_init	初始化TLI层色键
tli_layer_enable	使能TLI层

库函数名称	库函数描述
tli_layer_disable	禁能TLI层
tli_color_key_enable	使能TLI层色键
tli_color_key_disable	禁能TLI层色键
tli_lut_enable	使能TLI层颜色查找
tli_lut_disable	禁能TLI层颜色查找
tli_line_mark_set	设置行标记值
tli_current_pos_get	获取当前像素位置
tli_interrupt_enable	使能TLI中断
tli_interrupt_disable	禁能TLI中断
tli_interrupt_flag_get	获取TLI中断标志
tli_interrupt_flag_clear	清除TLI中断标志
tli_flag_get	从TLI_INTF寄存器或TLI_STAT寄存器获取TLI标志或状态

### 结构体 tli\_parameter\_struct

表 3-837. 结构体 tli\_parameter\_struct

成员名称	功能描述
synpsz_vpsz	垂直同步脉冲宽度
synpsz_hpsz	水平同步脉冲宽度
backpsz_vbpsz	垂直后沿加同步脉冲的宽度
backpsz_hbpsz	水平后沿加同步脉冲的宽度
activesz_vasz	垂直有效宽度加后沿像素和水平同步像素宽度
activesz_hasz	水平有效宽度加后沿像素和垂直同步像素宽度
totalsz_vtsz	显示器的垂直总宽度
totalsz_htsz	显示器的水平总宽度
backcolor_red	背景色红色值
backcolor_green	背景色绿色值
backcolor_blue	背景色蓝色值
signalpolarity_hs	水平脉冲极性选择

signalpolarity_vs	垂直脉冲极性选择
signalpolarity_de	数据使能极性选择
signalpolarity_pixelck	像素时钟极性选择

### 结构体 **tli\_layer\_parameter\_struct**

表 3-838. 结构体 **tli\_layer\_parameter\_struct**

成员名称	功能描述
layer_window_rightpos	窗口右边位置
layer_window_leftpos	窗口左边位置
layer_window_bottompos	窗口下边位置
layer_window_toppos	窗口上边位置
layer_ppf	包像素格式
layer_sa	特定alpha
layer_default_alpha	默认颜色alpha
layer_default_red	默认红色值
layer_default_green	默认绿色值
layer_default_blue	默认蓝色值
layer_acf1	Alpha混合计算因子1
layer_acf2	Alpha混合计算因子2
layer_frame_bufaddr	帧缓存区起始地址
layer_frame_buf_stride_offset	帧缓存区步幅偏移
layer_frame_line_length	帧行长度
layer_frame_total_line_number	帧总行数

### 结构体 **tli\_layer\_lut\_parameter\_struct**

表 3-839. 结构体 **tli\_layer\_lut\_parameter\_struct**

成员名称	功能描述
layer_table_addr	查表写地址
layer_lut_channel_red	颜色查找表条目红色通道

layer_lut_channel_green	颜色查找表条目绿色通道
layer_lut_channel_blue	颜色查找表条目蓝色通道

### 函数 **tli\_deinit**

函数tli\_deinit描述见下表:

**表 3-840. 函数 tli\_deinit**

函数名称	tli_deinit
函数原形	void tli_deinit (void);
功能描述	复位TLI
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 TLI registers */

tli_deinit();
```

### 函数 **tli\_struct\_para\_init**

函数tli\_struct\_para\_init描述见下表:

**表 3-841. 函数 tli\_struct\_para\_init**

函数名称	tli_struct_para_init
函数原形	void tli_struct_para_init(tli_parameter_struct *tli_struct);
功能描述	用默认值初始化TLI参数结构体，建议在定义一个tli_parameter_struct结构体后调用该接口实现对结构体的初始化
先决条件	-
被调用函数	-

输入参数{in}	
*tli_struct	指向TLI参数结构体的指针
输出参数{out}	
-	-
返回值	
-	-

例如：

```
tli_parameter_struct tli_init_struct;
/* initialize the parameters of TLI parameter structure with the default values */
tli_struct_para_init(&tli_init_struct);
```

### 函数 tli\_init

函数tli\_init描述见下表：

表 3-842. 函数 tli\_init

函数名称	tli_init
函数原形	void tli_init(tli_parameter_struct *tli_struct);
功能描述	初始化TLI
先决条件	-
被调用函数	-
输入参数{in}	
*tli_struct	指向TLI参数结构体的指针
输出参数{out}	
-	-
返回值	
-	-

例如：

```
tli_parameter_struct tli_init_struct;
/* initialize the parameters of TLI parameter structure with the default values */
tli_struct_para_init(&tli_init_struct);
```

```

/* configure TLI parameter struct */

tli_init_struct.signalpolarity_hs = TLI_HSYN_ACTLIVE_LOW;
tli_init_struct.signalpolarity_vs = TLI_VSYN_ACTLIVE_LOW;
tli_init_struct.signalpolarity_de = TLI_DE_ACTLIVE_LOW;
tli_init_struct.signalpolarity_pixelck = TLI_PIXEL_CLOCK_TLI;

/* LCD display timing configuration */

tli_init_struct.synpsz_hpsz = 40;
tli_init_struct.synpsz_vpsz = 9;
tli_init_struct.backpsz_hbpsz = 42;
tli_init_struct.backpsz_vbpsz = 11;
tli_init_struct.activesz_hasz = 522;
tli_init_struct.activesz_vasz = 283;
tli_init_struct.totalsz_htsz = 524;
tli_init_struct.totalsz_vtsz = 285;

/* configure LCD background R,G,B values */

tli_init_struct.backcolor_red = 0xFF;
tli_init_struct.backcolor_green = 0xFF;
tli_init_struct.backcolor_blue = 0xFF;
tli_init(&tli_init_struct);

```

### **函数 tli\_dither\_config**

函数tli\_dither\_config描述见下表：

**表 3-843. 函数 tli\_dither\_config**

函数名称	tli_dither_config
函数原形	void tli_dither_config(uint8_t ditherstat);
功能描述	配置TLI抖动功能
先决条件	-
被调用函数	-
输入参数{in}	

<b>ditherstat</b>	抖动状态
<b>TLI_DITHER_ENABLE</b>	使能TLI抖动
<b>TLI_DITHER_DISABLE</b>	禁能TLI抖动
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable TLI dither function */
tli_dither_config(TLI_DITHER_ENABLE);
```

### 函数 **tli\_enable**

函数tli\_enable描述见下表：

**表 3-844. 函数 tli\_enable**

<b>函数名称</b>	tli_enable
<b>函数原形</b>	void tli_enable(void);
<b>功能描述</b>	使能TLI
<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 TLI */
tli_enable();
```

### 函数 tli\_disable

函数tli\_disable描述见下表:

**表 3-845. 函数 tli\_disable**

函数名称	tli_disable
函数原形	void tli_disable(void);
功能描述	禁能TLI
先决条件	-
被调用函数	-
输入参数{in}	
-	-
Output parameter{out}	
-	-
返回值	
-	-

例如:

```
/* disable TLI */
tli_disable();
```

### 函数 tli\_reload\_config

函数tli\_reload\_config描述见下表:

**表 3-846. 函数 tli\_reload\_config**

函数名称	tli_reload_config
函数原形	void tli_reload_config(uint8_t reloadmode);
功能描述	configure TLI reload mode
先决条件	-
被调用函数	-
输入参数{in}	
reloadmode	Reload mode
TLI_FRAME_BLANK_R	层配置在帧间隔被重载

<i>ELOAD_EN</i>	
<i>TLI_REQUEST_RELOAD_EN</i>	层配置在置位后被重载
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure TLI reload mode */

tli_reload_config(TLI_FRAME_BLANK_RELOAD_EN);
```

#### 函数 **tli\_layer\_struct\_para\_init**

函数tli\_layer\_struct\_para\_init描述见下表：

**表 3-847. 函数 tli\_layer\_struct\_para\_init**

函数名称	tli_layer_struct_para_init
函数原形	void tli_layer_struct_para_init(tli_layer_parameter_struct *layer_struct);
功能描述	用默认参数初始化TLI层结构体，建议在定义一个tli_layer_parameter_struct结构体后调用该接口实现对结构体的初始化
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
* <i>layer_struct</i>	指向tli_layer_parameter_struct结构体的指针
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
tli_layer_parameter_struct tli_layer_init_struct;

/* initialize the parameters of TLI layer structure with the default values */

tli_layer_struct_para_init(&tli_layer_init_struct);
```

### 函数 tli\_layer\_init

函数tli\_layer\_init描述见下表：

**表 3-848. 函数 tli\_layer\_init**

函数名称	tli_layer_init
函数原形	void tli_layer_init(uint32_t layerx,tli_layer_parameter_struct *layer_struct)
功能描述	初始化TLI层
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0层基地址
LAYER1	1层基地址
输入参数{in}	
*layer_struct	指向TLI层参数结构体的指针
输出参数{out}	
-	-
返回值	
-	-

例如：

```
tli_layer_parameter_struct tli_layer_init_struct;
tli_layer_struct_para_init(&tli_layer_init_struct);
/* TLI layer0 configuration */
tli_layer_init_struct.layer_window_leftpos = 20 + 43;
tli_layer_init_struct.layer_window_rightpos = (20 + 440 + 43 - 1);
tli_layer_init_struct.layer_window_toppos = 40 + 12;
tli_layer_init_struct.layer_window_bottompos = (40 + 182 + 12 - 1);
tli_layer_init_struct.layer_ppf = LAYER_PPF_RGB565;
/* TLI window specified alpha configuration */
tli_layer_init_struct.layer_sa = 255;
```

```

/* TLI layer default alpha R,G,B value configuration */

tli_layer_init_struct.layer_default_blue = 0xFF;
tli_layer_init_struct.layer_default_green = 0xFF;
tli_layer_init_struct.layer_default_red = 0xFF;
tli_layer_init_struct.layer_default_alpha = 0xFF;

/* TLI window blend configuration */

tli_layer_init_struct.layer_acf1 = LAYER_ACF1_PASA;
tli_layer_init_struct.layer_acf2 = LAYER_ACF2_PASA;

/* TLI layer frame buffer base address configuration */

tli_layer_init_struct.layer_frame_bufaddr = (uint32_t)&gBackground;
tli_layer_init_struct.layer_frame_line_length = ((440 * 2) + 3);
tli_layer_init_struct.layer_frame_buf_stride_offset = (440 * 2);
tli_layer_init_struct.layer_total_line_number = 182;
tli_layer_init(LAYER0, &tli_layer_init_struct);

```

### **函数 tli\_layer\_window\_offset\_modify**

函数tli\_layer\_window\_offset\_modify描述见下表：

**表 3-849. 函数 tli\_layer\_window\_offset\_modify**

函数名称	tli_layer_window_offset_modify
函数原形	void tli_layer_window_offset_modify(uint32_t layerx,uint16_t offset_x,uint16_t offset_y);
功能描述	重新配置窗口位置
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>layerx</b>	层基地址
<b>LAYER0</b>	0层基地址
<b>LAYER1</b>	1层基地址
<b>输入参数{in}</b>	
<b>offset_x</b>	新水平偏移

输入参数{in}	
<b>offset_y</b>	新垂直偏移
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reconfigure LAYER1 window position */

tli_layer_window_offset_modify(LAYER1, 20, 20);
```

#### 函数 **tli\_lut\_struct\_para\_init**

函数tli\_lut\_struct\_para\_init描述见下表：

表 3-850. 函数 **tli\_lut\_struct\_para\_init**

函数名称	tli_lut_struct_para_init
函数原形	void tli_lut_struct_para_init(tli_layer_lut_parameter_struct *lut_struct);
功能描述	用默认参数初始化TLI层LUT结构体，建议在定义一个tli_layer_lut_parameter_struct结构体后调用该接口实现对结构体的初始化
先决条件	-
被调用函数	-
输入参数{in}	
*lut_struct	指向tli_layer_lut_parameter_struct结构体的指针
输出参数{out}	
-	-
返回值	
-	-

例如：

```
tli_layer_lut_parameter_struct tli_lut_struct;
/* initialize the parameters of TLI layer LUT structure with the default values */
tli_lut_struct_para_init(&tli_lut_struct);
```

### 函数 tli\_lut\_init

函数tli\_lut\_init描述见下表：

**表 3-851. 函数 tli\_lut\_init**

函数名称	tli_lut_init
函数原形	void tli_lut_init(uint32_t layerx,tli_layer_lut_parameter_struct *lut_struct)
功能描述	初始化TLI层颜色查表
先决条件	-
被调用函数	-
输入参数{in}	
<b>layerx</b>	层基地址
<i>LAYER0</i>	0层基地址
<i>LAYER1</i>	1层基地址
输入参数{in}	
<b>*lut_struct</b>	指向TLI层LUT参数结构体的指针
输出参数{out}	
-	-
返回值	
-	-

例如：

```
tli_layer_lut_parameter_struct tli_lut_struct;
tli_lut_struct_para_init(&tli_lut_struct);
/* initialize TLI layer0 LUT */
tli_lut_struct.layer_table_addr = 0x20003000;
tli_lut_struct.layer_lut_channel_red = 0x20;
tli_lut_struct.layer_lut_channel_green = 0x30;
tli_lut_struct.layer_lut_channel_blue = 0xFF;
tli_lut_init(LAYER0, &tli_lut_struct);
```

### 函数 tli\_color\_key\_init

函数tli\_color\_key\_init描述见下表:

**表 3-852. 函数 tli\_color\_key\_init**

函数名称	tli_color_key_init
函数原形	void tli_color_key_init(uint32_t layerx,uint8_t redkey,uint8_t greenkey,uint8_t bluekey);
功能描述	初始化TLI层色键
先决条件	-
被调用函数	-
输入参数{in}	
<b>layerx</b>	层基地址
<i>LAYER0</i>	0层基地址
<i>LAYER1</i>	1层基地址
输入参数{in}	
<b>redkey</b>	红色键
输入参数{in}	
<b>greenkey</b>	绿色键
输入参数{in}	
<b>bluekey</b>	蓝色键
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize TLI layer0 color key */
tli_color_key_init(LAYER0, 0xAA, 0xFF, 0x00);
```

### 函数 tli\_layer\_enable

函数tli\_layer\_enable描述见下表:

表 3-853. 函数 tli\_layer\_enable

函数名称	tli_layer_enable
函数原形	void tli_layer_enable(uint32_t layerx);
功能描述	使能TLI层
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0层基地址
LAYER1	1层基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* TLI layer enable */

tli_layer_enable(LAYER0);
```

#### 函数 tli\_layer\_disable

函数tli\_layer\_disable描述见下表：

表 3-854. 函数 tli\_layer\_disable

函数名称	tli_layer_disable
函数原形	void tli_layer_disable(uint32_t layerx);
功能描述	禁能TLI层
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0层基地址

<i>LAYER1</i>	1层基址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* TLI layer disable */
```

```
tli_layer_disable(LAYER0);
```

#### 函数 **tli\_color\_key\_enable**

函数tli\_color\_key\_enable描述见下表：

表 3-855. 函数 **tli\_color\_key\_enable**

函数名称	tli_color_key_enable
函数原形	void tli_color_key_enable(uint32_t layerx);
功能描述	使能TLI层色键
先决条件	-
被调用函数	-
输入参数{in}	
<b>layerx</b>	层基址
<i>LAYER0</i>	0层基址
<i>LAYER1</i>	1层基址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TLI layer0 color keying */
```

```
tli_color_key_enable(LAYER0);
```

### **函数 tli\_color\_key\_disable**

函数tli\_color\_key\_disable描述见下表:

**表 3-856. 函数 tli\_color\_key\_disable**

函数名称	tli_color_key_disable
函数原形	void tli_color_key_disable(uint32_t layerx);
功能描述	禁能TLI层色键
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0层基地址
LAYER1	1层基地址
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable TLI layer0 color keying */
tli_color_key_disable(LAYER0);
```

### **函数 tli\_lut\_enable**

函数tli\_lut\_enable描述见下表:

**表 3-857. 函数 tli\_lut\_enable**

函数名称	tli_lut_enable
函数原形	void tli_lut_enable(uint32_t layerx);
功能描述	使能TLI层颜色查找
先决条件	-
被调用函数	-
输入参数{in}	

<b>layerx</b>	层基地址
<i>LAYER0</i>	0层基地址
<i>LAYER1</i>	1层基地址
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable TLI layer0 LUT */
tli_lut_enable(LAYER0);
```

#### 函数 **tli\_lut\_disable**

函数tli\_lut\_disable描述见下表：

表 3-858. 函数 **tli\_lut\_disable**

<b>函数名称</b>	tli_lut_disable
<b>函数原形</b>	void tli_lut_disable(uint32_t layerx);
<b>功能描述</b>	禁能TLI层颜色查找
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>layerx</b>	层基地址
<i>LAYER0</i>	0层基地址
<i>LAYER1</i>	1层基地址
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable TLI layer0 LUT */
```

```
tli_lut_disable(LAYER0);
```

### 函数 tli\_line\_mark\_set

函数tli\_line\_mark\_set描述见下表:

**表 3-859. 函数 tli\_line\_mark\_set**

函数名称	tli_line_mark_set
函数原形	void tli_line_mark_set(uint16_t line_num);
功能描述	设置行标记值
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
line_num	行编号
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* set line mark value */
tli_line_mark_set(0x20);
```

### 函数 tli\_current\_pos\_get

函数tli\_current\_pos\_get描述见下表:

**表 3-860. 函数 tli\_current\_pos\_get**

函数名称	tli_current_pos_get
函数原形	uint32_t tli_current_pos_get(void);
功能描述	获取当前像素位置
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
-	-

输出参数{out}	
-	-
返回值	
uint32_t	0x0 – 0xFFFFFFFF

例如：

```
uint32_t pos;  
  
/* get current pixel position */  
  
pos = tli_current_pos_get();
```

### 函数 tli\_interrupt\_enable

函数tli\_interrupt\_enable描述见下表：

表 3-861. 函数 tli\_interrupt\_enable

函数名称	tli_interrupt_enable
函数原形	void tli_interrupt_enable(uint32_t int_flag);
功能描述	使能TLI中断
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	TLI中断标志
TLI_INT_LM	行标记中断
TLI_INT_FE	FIFO错误中断
TLI_INT_TE	事务错误中断
TLI_INT_LCR	层配置重载中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TLI line mark interrupt */
```

```
tli_interrupt_enable(TLI_INT_LM);
```

### 函数 tli\_interrupt\_disable

函数tli\_interrupt\_disable描述见下表:

**表 3-862. 函数 tli\_interrupt\_disable**

函数名称	tli_interrupt_disable
函数原形	void tli_interrupt_disable(uint32_t int_flag);
功能描述	禁能TLI中断
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
int_flag	TLI中断标志
TLI_INT_LM	行标记中断
TLI_INT_FE	FIFO错误中断
TLI_INT_TE	事务错误中断
TLI_INT_LCR	层配置重载中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* disable TLI line mark interrupt */

tli_interrupt_disable(TLI_INT_LM);
```

### 函数 tli\_interrupt\_flag\_get

函数tli\_interrupt\_flag\_get描述见下表:

**表 3-863. 函数 tli\_interrupt\_flag\_get**

函数名称	tli_interrupt_flag_get
函数原形	FlagStatus tli_interrupt_flag_get(uint32_t int_flag);
功能描述	获取TLI中断标志

先决条件	-
被调用函数	-
输入参数{in}	
<b>int_flag</b>	TLI中断标志
<i>TLI_INT_FLAG_LM</i>	行标记中断
<i>TLI_INT_FLAG_FE</i>	FIFO错误中断
<i>TLI_INT_FLAG_TE</i>	事务错误中断
<i>TLI_INT_FLAG_LCR</i>	层配置重载中断
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如：

```
/* get TLI interrupt flag */

if(SET == tli_interrupt_flag_get(TLI_INT_FLAG_LM)){
    tli_interrupt_flag_clear(TLI_INT_FLAG_LM);
}
```

#### 函数 **tli\_interrupt\_flag\_clear**

函数tli\_interrupt\_flag\_clear描述见下表：

表 3-864. 函数 **tli\_interrupt\_flag\_clear**

函数名称	tli_interrupt_flag_clear
函数原形	void tli_interrupt_flag_clear(uint32_t int_flag)
功能描述	清除TLI中断标志
先决条件	-
被调用函数	-
输入参数{in}	
<b>int_flag</b>	TLI中断标志
<i>TLI_INT_FLAG_LM</i>	行标记中断

<i>TLI_INT_FLAG_FE</i>	FIFO错误中断
<i>TLI_INT_FLAG_TE</i>	事务错误中断
<i>TLI_INT_FLAG_LCR</i>	层配置重载中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
if(SET == tli_interrupt_flag_get(TLI_INT_FLAG_LM)){
    /* clear TLI interrupt flag */
    tli_interrupt_flag_clear(TLI_INT_FLAG_LM);
}
```

### 函数 **tli\_flag\_get**

函数tli\_flag\_get描述见下表：

表 3-865. 函数 **tli\_flag\_get**

函数名称	tli_flag_get
函数原形	FlagStatus tli_flag_get(uint32_t flag);
功能描述	从TLI_INTF寄存器或TLI_STAT寄存器获取TLI标志或状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	TLI flags
<i>TLI_FLAG_VDE</i>	当前VDE状态
<i>TLI_FLAG_HDE</i>	当前HDE状态
<i>TLI_FLAG_VS</i>	当前vs状态
<i>TLI_FLAG_HS</i>	当前hs状态
<i>TLI_FLAG_LM</i>	行标记中断标志
<i>TLI_FLAG_FE</i>	FIFO错误中断标志

<i>TLI_FLAG_TE</i>	事务错误中断标志
<i>TLI_FLAG_LCR</i>	层配置重载中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET或RESET

例如：

```
/* wait the TLI_FLAG_LM flag set */

while(RESET == tli_flag_get(TLI_FLAG_LM)){
}
```

## 3.28. TRNG

真随机数发生器能够通过连续模拟噪声生成一个 32 位的随机数值。章节 [3.28.1](#) 描述了 TRNG 的寄存器列表，章节 [3.28.2](#) 对 TRNG 库函数进行了说明。

### 3.28.1. 外设寄存器说明

TRNG 寄存器列表如下表所示：

**表 3-866. TRNG 寄存器**

寄存器名称	寄存器描述
TRNG_CTL	TRNG控制寄存器
TRNG_STAT	TRNG状态寄存器
TRNG_DATA	TRNG数据寄存器

### 3.28.2. 外设库函数说明

TRNG 库函数列表如下表所示：

**表 3-867. TRNG 库函数**

库函数名称	库函数描述
trng_deinit	复位外设TRNG
trng_enable	使能外设TRNG

库函数名称	库函数描述
trng_disable	失能外设TRNG
trng_get_true_random_data	获取真随机值
trng_interrupt_enable	使能TRNG中断
trng_interrupt_disable	失能TRNG中断
trng_flag_get	获取TRNG标志状态
trng_interrupt_flag_get	获取TRNG中断状态
trng_interrupt_flag_clear	清除TRNG中断状态

### 函数 trng\_deinit

函数trng\_deinit描述见下表:

**表 3-868 函数 trng\_deinit**

函数名称	trng_deinit
函数原形	void trng_deinit(void);
功能描述	复位外设TRNG
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 TRNG*/
trng_deinit();
```

### 函数 trng\_enable

函数trng\_enable描述见下表:

表 3-869 函数 trng\_enable

函数名称	trng_enable
函数原形	void trng_enable(void);
功能描述	使能外设TRNG
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 TRNG*/
trng_enable();
```

### 函数 trng\_disable

函数trng\_disable描述见下表：

表 3-870 函数 trng\_disable

函数名称	trng_disable
函数原形	void trng_disable(void);
功能描述	失能外设TRNG
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 TRNG*/
```

```
trng_disable();
```

### 函数 **trng\_get\_true\_random\_data**

函数trng\_get\_true\_random\_data描述见下表：

**表 3-871 函数 trng\_get\_true\_random\_data**

函数名称	trng_get_true_random_data
函数原形	uint32_t trng_get_true_random_data(void);
功能描述	获取真随机值
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
uint32_t	32位真随机值

例如：

```
/* get true random data*/
uint32_t data;
data = trng_get_true_random_data();
```

### 函数 **trng\_interrupt\_enable**

函数trng\_interrupt\_enable描述见下表：

**表 3-872 函数 trng\_interrupt\_enable**

函数名称	trng_interrupt_enable
函数原形	void trng_interrupt_enable(void);
功能描述	使能TRNG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 TRNG interrupt*/
trng_interrupt_enable();
```

### 函数 trng\_interrupt\_disable

函数trng\_interrupt\_disable描述见下表：

表 3-873 函数 trng\_interrupt\_disable

函数名称	trng_interrupt_disable
函数原形	void trng_interrupt_disable(void);
功能描述	失能TRNG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 TRNG interrupt*/
trng_interrupt_disable();
```

### 函数 trng\_flag\_get

函数trng\_flag\_get描述见下表：

**表 3-874 函数 trng\_flag\_get**

函数名称	trng_flag_get
函数原形	FlagStatus trng_flag_get(trng_flag_enum flag);
功能描述	获取TRNG标志状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	TRNG标志状态
TRNG_FLAG_DRD Y	随机数准备标志
TRNG_FLAG_CEC S	时钟错误当前标志
TRNG_FLAG_SEC S	种子错误当前标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get TRNG clock error current flag status*/
FlagStatus flag_status = RESET;
flag_status == trng_flag_get(TRNG_FLAG_CECS);
```

### 函数 trng\_interrupt\_flag\_get

函数trng\_interrupt\_flag\_get描述见下表：

**表 3-875 函数 trng\_interrupt\_flag\_get**

函数名称	trng_interrupt_flag_get
函数原形	FlagStatus trng_interrupt_flag_get(trng_int_flag_enum int_flag);

功能描述	获取TRNG中断状态
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>int_flag</b>	TRNG中断状态
<i>TRNG_INT_FLAG_CEIF</i>	时钟错误中断
<i>TRNG_INT_FLAG_SEIF</i>	种子错误中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET 或 RESET

例如：

```
/* get TRNG clock error interrupt flag*/
FlagStatus interrupt_flag = RESET;
interrupt_flag = trng_interrupt_flag_get(TRNG_INT_FLAG_CEIF);
```

#### 函数 **trng\_interrupt\_flag\_clear**

函数**trng\_interrupt\_flag\_clear**描述见下表：

**表 3-876 函数 **trng\_interrupt\_flag\_clear****

函数名称	trng_interrupt_flag_clear
函数原形	void trng_interrupt_flag_clear(trng_int_flag_enum int_flag);
功能描述	清除TRNG中断状态
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>int_flag</b>	TRNG中断状态
<i>TRNG_INT_FLAG_CEIF</i>	时钟错误中断

<i>TRNG_INT_FLAG_SEIF</i>	种子错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear TRNG clock error interrupt flag*/
trng_interrupt_flag_clear(TRNG_INT_FLAG_CEIF);
```

## 3.29. 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[3.29.1](#)描述了USART的寄存器列表，章节[3.29.2](#)对USART库函数进行说明。

### 3.29.1. 外设寄存器说明

USART寄存器列表如下表所示：

**表 3-877. USART 寄存器**

寄存器名称	寄存器描述
USART_STAT0	状态寄存器0
USART_DATA	数据寄存器
USART_BAUD	波特率寄存器
USART_CTL0	控制寄存器0
USART_CTL1	控制寄存器1
USART_CTL2	控制寄存器2
USART_GP	保护时间和预分频器寄存器
USART_CTL3	控制寄存器3
USART_RT	接收超时寄存器
USART_STAT1	状态寄存器1
USART_CHC	兼容性控制寄存器

### 3.29.2. 外设库函数说明

USART库函数列表如下表所示：

**表 3-878. USART 库函数**

库函数名称	库函数描述
uart_deinit	复位外设USART
uart_baudrate_set	配置USART波特率
uart_parity_config	配置USART奇偶校验
uart_word_length_set	配置USART字长
uart_stop_bit_set	配置USART停止位
uart_enable	使能USART
uart_disable	失能USART
uart_transmit_config	USART发送配置
uart_receive_config	USART接收配置
uart_data_first_config	配置数据传输时低位在前或高位在前
uart_invert_config	配置USART反转功能
uart_oversample_config	配置USART过采样模式
uart_sample_bit_config	配置USART采样位方法
uart_receiver_timeout_enable	使能USART接收超时
uart_receiver_timeout_disable	失能USART接收超时
uart_receiver_timeout_threshold_config	设置USART接收超时阈值
uart_data_transmit	USART发送数据功能
uart_data_receive	USART接收数据功能
uart_address_config	在地址掩码唤醒模式下配置USART地址
uart_mute_mode_enable	使能USART静默模式
uart_mute_mode_disable	失能USART静默模式
uart_mute_mode_wakeup_config	配置USART静默模式唤醒方式
uart_lin_mode_enable	使能USART LIN模式
uart_lin_mode_disable	失能USART LIN模式
uart_lin_break_detection_length_config	配置USART LIN模式中断帧长度
uart_send_break	配置USART发送断开帧
uart_halfduplex_enable	使能USART半双工模式
uart_halfduplex_disable	失能USART半双工模式
uart_synchronous_clock_enable	在USART同步通讯模式下使能CK引脚
uart_synchronous_clock_disable	在USART同步通讯模式下失能CK引脚
uart_synchronous_clock_config	配置USART同步通讯模式参数
uart_guard_time_config	在USART智能卡模式下配置保护时间值
uart_smartcard_mode_enable	使能USART智能卡模式
uart_smartcard_mode_disable	失能USART智能卡模式
uart_smartcard_mode_nack_enable	在USART智能卡模式下使能NACK
uart_smartcard_mode_nack_disable	在USART智能卡模式下失能NACK
uart_smartcard_autoretry_config	配置智能卡自动重试次数
uart_block_length_config	配置智能卡T=1的接收时块的长度

库函数名称	库函数描述
uart_irda_mode_enable	使能USART串行红外编解码功能模块
uart_irda_mode_disable	失能USART串行红外编解码功能模块
uart_prescaler_config	在USART IrDA低功耗模式下配置外设时钟分频系数
uart_irda_lowpower_config	配置USART IrDA低功耗模式
uart_hardware_flow_rts_config	配置USART RTS硬件控制流
uart_hardware_flow_cts_config	配置USART CTS硬件控制流
uart_break_frame_coherence_config	配置USART断开帧兼容模式
uart_parity_check_coherence_config	配置USART校验兼容模式
uart_hardware_flow_coherence_config	配置USART硬件流控制兼容模式
uart_dma_receive_config	配置USART DMA接收功能
uart_dma_transmit_config	配置USART DMA发送功能
uart_flag_get	获取USART状态寄存器标志位
uart_flag_clear	清除USART状态寄存器标志位
uart_interrupt_enable	使能USART中断
uart_interrupt_disable	失能USART中断
uart_interrupt_flag_get	获取USART中断标志位状态
uart_interrupt_flag_clear	清除USART中断标志位状态

### 函数 **uart\_deinit**

函数uart\_deinit描述见下表：

**表 3-879. 函数 **uart\_deinit****

函数名称	uart_deinit
函数原型	void uart_deinit(uint32_t usart_periph);
功能描述	复位外设USARTx/UARTx
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>输入参数{in}</b>	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* reset USART0 */
uart_deinit (USART0);
```

### 函数 **usart\_baudrate\_set**

函数**usart\_baudrate\_set**描述见下表：

**表 3-880. 函数 usart\_baudrate\_set**

函数名称	usart_baudrate_set
函数原型	void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);
功能描述	配置USART波特率
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
<b>usart_periph</b>	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
输入参数{in}	
<b>baudval</b>	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 3-881. 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}	
<b>usart_periph</b>	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
输入参数{in}	
<b>paritycfg</b>	配置USART奇偶校验
<i>USART_PM_NONE</i>	无校验
<i>USART_PM_ODD</i>	奇校验

<b>USART_PM EVEN</b>	偶校验
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure USART parity */

uart_parity_config(USART0, USART_PM EVEN);
```

### 函数 **uart\_word\_length\_set**

函数 **uart\_word\_length\_set** 描述见下表：

**表 3-882. 函数 **uart\_word\_length\_set****

函数名称	uart_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	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx
<b>USARTx</b>	x=0,1,2,5
<b>UARTx</b>	x=3,4,6,7
<b>输入参数{in}</b>	
<b>wlen</b>	配置USART字长
<b>USART_WL_8BIT</b>	8 bits
<b>USART_WL_9BIT</b>	9 bits
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure USART0 word length */

uart_word_length_set(USART0, USART_WL_9BIT);
```

### 函数 **uart\_stop\_bit\_set**

函数 **uart\_stop\_bit\_set** 描述见下表：

**表 3-883. 函数 **uart\_stop\_bit\_set****

函数名称	uart_stop_bit_set
------	-------------------

<b>函数原型</b>	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);
<b>功能描述</b>	配置USART停止位
<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
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>输入参数{in}</b>	
<b>stblen</b>	配置USART停止位
<i>USART_STB_1BIT</i>	1 bit
<i>USART_STB_0_5BIT</i> <i>T</i>	0.5 bit,该位对UARTx(x=3,4,6,7)无效
<i>USART_STB_2BIT</i>	2 bits
<i>USART_STB_1_5BIT</i> <i>T</i>	1.5 bits,该位对UARTx(x=3,4,6,7)无效
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-884. 函数 **usart\_enable****

<b>函数名称</b>	usart_enable
<b>函数原型</b>	void usart_enable(uint32_t usart_periph);
<b>功能描述</b>	使能USART
<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
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	

-	-
---	---

例如：

```
/* enable USART0 */

uart_enable(USART0);
```

### 函数 **uart\_disable**

函数 **uart\_disable** 描述见下表：

**表 3-885. 函数 **uart\_disable****

函数名称	uart_disable
函数原型	void usart_disable(uint32_t usart_periph);
功能描述	失能USART
先决条件	-
被调用函数	-
输入参数{in}	
<b>usart_periph</b>	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 */

uart_disable(USART0);
```

### 函数 **uart\_transmit\_config**

函数 **uart\_transmit\_config** 描述见下表：

**表 3-886. 函数 **uart\_transmit\_config****

函数名称	uart_transmit_config
函数原型	void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);
功能描述	USART发送器配置
先决条件	-
被调用函数	-
输入参数{in}	
<b>usart_periph</b>	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7

输入参数{in}	
<b>txconfig</b>	使能/失能USART发送器
<b>USART_TRANSMIT_ENABLE</b>	使能USART发送
<b>USART_TRANSMIT_DISABLE</b>	失能USART发送
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 transmitter */

uart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

### 函数 **uart\_receive\_config**

函数**uart\_receive\_config**描述见下表：

表 3-887. 函数 **uart\_receive\_config**

函数名称	uart_receive_config
函数原型	void uart_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}	
<b>usart_periph</b>	外设USARTx/UARTx
<b>USARTx</b>	x=0,1,2,5
<b>UARTx</b>	x=3,4,6,7
输入参数{in}	
<b>rxconfig</b>	使能/失能USART接收器
<b>USART_RECEIVE_ENABLE</b>	使能USART接收
<b>USART_RECEIVE_DISABLE</b>	失能USART接收
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 receiver */
```

```
uart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

### 函数 **uart\_data\_first\_config**

函数uart\_data\_first\_config描述见下表：

**表 3-888. 函数 **uart\_data\_first\_config****

函数名称	uart_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	-	
<b>输入参数{in}</b>		
usart_periph	外设USARTx	
USARTx	x=0,1,2,5	
<b>输入参数{in}</b>		
msbf	数据传输时低位在前/高位在前	
USART_MSBF_LS B	数据传输时低位在前	
USART_MSBF_MS B	数据传输时高位在前	
<b>输出参数{out}</b>		
-	-	
<b>返回值</b>		
-	-	

例如：

```
/* configure LSB of data first */
uart_data_first_config(USART0, USART_MSBF_LSB);
```

### 函数 **uart\_invert\_config**

函数uart\_invert\_config描述见下表：

**表 3-889. 函数 **uart\_invert\_config****

函数名称	uart_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	-	
<b>输入参数{in}</b>		
usart_periph	外设USARTx	
USARTx	x=0,1,2,5	
<b>输入参数{in}</b>		

<code>invertpara</code>	参考枚举 <code>usart_invert_enum</code>
<code>USART_DINV_ENA BLE</code>	数据位电平反转
<code>USART_DINV_DIS ABLE</code>	数据位电平不反转
<code>USART_TXPIN_EN ABLE</code>	TX引脚电平反转
<code>USART_TXPIN_DIS ABLE</code>	TX引脚电平不反转
<code>USART_RXPIN_EN ABLE</code>	RX引脚电平反转
<code>USART_RXPIN_DIS ABLE</code>	RX引脚电平不反转
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure USART inversion */

usart_invert_config(USART0, USART_DINV_ENABLE);
```

### 函数 `usart_oversample_config`

函数`usart_oversample_config`描述见下表：

**表 3-890. 函数 `usart_oversample_config`**

函数名称	<code>usart_oversample_config</code>
函数原型	<code>void usart_oversample_config(uint32_t usart_periph, uint32_t oversamp);</code>
功能描述	配置USART过采样模式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<code>usart_periph</code>	外设USARTx/UARTx
<code>USARTx</code>	$x=0,1,2,5$
<code>UARTx</code>	$x=3,4,6,7$
<b>输入参数{in}</b>	
<code>oversamp</code>	过采样值
<code>USART_OVSMOD_8</code>	8位
<code>USART_OVSMOD_16</code>	16位
<b>输出参数{out}</b>	

-	-
返回值	
-	-

例如：

```
/* configure USART0 oversample mode */
uart_oversample_config(USART0, USART_OVSMOD_8);
```

#### 函数 **uart\_sample\_bit\_config**

函数uart\_sample\_bit\_config描述见下表：

**表 3-891. 函数 **uart\_sample\_bit\_config****

函数名称	uart_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}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
输入参数{in}	
obsm	采样位
USART_OSB_1bit	1位
USART_OSB_3bit	3位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 sample bit */
uart_sample_bit_config(USART0, USART_OSB_1bit);
```

#### 函数 **uart\_receiver\_timeout\_enable**

函数uart\_receiver\_timeout\_enable描述见下表：

**表 3-892. 函数 **uart\_receiver\_timeout\_enable****

函数名称	uart_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	-
<b>输入参数{in}</b>	
uart_periph	外设USARTx
USARTx	x=0,1,2,5
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable receiver timeout of USART */
uart_receiver_timeout_enable(USART0);
```

#### 函数 **uart\_receiver\_timeout\_disable**

函数uart\_receiver\_timeout\_disable描述见下表：

**表 3-893. 函数 **uart\_receiver\_timeout\_disable****

函数名称	uart_receiver_timeout_disable
函数原型	void uart_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	-
<b>输入参数{in}</b>	
uart_periph	外设USARTx
USARTx	x=0,1,2,5
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable receiver timeout of USART */
uart_receiver_timeout_disable(USART0);
```

#### 函数 **uart\_receiver\_timeout\_threshold\_config**

函数uart\_receiver\_timeout\_threshold\_config描述见下表：

**表 3-894. 函数 **uart\_receiver\_timeout\_threshold\_config****

函数名称	uart_receiver_timeout_threshold_config
函数原型	void uart_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}	
uart_periph	外设USARTx
USARTx	x=0,1,2,5
输入参数{in}	
rtimeout	超时时间
0-0xFFFFFFF	超时时间值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the receiver timeout threshold of USART0 */

uart_receiver_timeout_threshold_config(USART0, 115200*3);
```

### 函数 **uart\_data\_transmit**

函数uart\_data\_transmit描述见下表：

**表 3-895. 函数 **uart\_data\_transmit****

函数名称	uart_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}	
uart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
输入参数{in}	
data	发送的数据
0-0x1FF	发送的数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 transmit data */
```

```
uart_data_transmit(USART0, 0xAA);
```

### 函数 **uart\_data\_receive**

函数uart\_data\_receive描述见下表：

**表 3-896. 函数 **uart\_data\_receive****

函数名称	uart_data_receive
函数原型	uint16_t usart_data_receive(uint32_t usart_periph);
功能描述	USART接收数据功能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
uint32_t	接收的数据 (0-0xFF)

例如：

```
/* USART0 receive data */

uint16_t temp;

temp = usart_data_receive(USART0);
```

### 函数 **uart\_address\_config**

函数uart\_address\_config描述见下表：

**表 3-897. 函数 **uart\_address\_config****

函数名称	uart_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	-
<b>输入参数{in}</b>	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
<b>输入参数{in}</b>	
addr	USART/UART地址
0-0xFF	USART/UART地址

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure address of the USART0 */

uart_address_config(USART0, 0x00);
```

### 函数 **uart\_mute\_mode\_enable**

函数uart\_mute\_mode\_enable描述见下表：

**表 3-898. 函数 **uart\_mute\_mode\_enable****

函数名称	uart_mute_mode_enable
函数原型	void usart_mute_mode_enable(uint32_t usart_periph);
功能描述	使能USART静默模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 receiver in mute mode */

uart_mute_mode_enable(USART0);
```

### 函数 **uart\_mute\_mode\_disable**

函数uart\_mute\_mode\_disable描述见下表：

**表 3-899. 函数 **uart\_mute\_mode\_disable****

函数名称	uart_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}	

<b>usart_periph</b>	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable USART0 receiver in mute mode */

uart_mute_mode_disable(USART0);
```

#### 函数 **uart\_mute\_mode\_wakeup\_config**

函数uart\_mute\_mode\_wakeup\_config描述见下表：

表 3-900. 函数 **uart\_mute\_mode\_wakeup\_config**

<b>函数名称</b>	uart_mute_mode_wakeup_config
<b>函数原型</b>	void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);
<b>功能描述</b>	配置USART静默模式唤醒方式
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>输入参数{in}</b>	
<b>wmethod</b>	两种方法用于进入或退出静默模式
<i>USART_WM_IDLE</i>	空闲线唤醒
<i>USART_WM_ADDR</i>	地址掩码唤醒
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure USART0 wakeup method in mute mode */

uart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

#### 函数 **uart\_lin\_mode\_enable**

函数uart\_lin\_mode\_enable描述见下表：

**表 3-901. 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=3,4,6,7
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 3-902. 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=3,4,6,7
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 3-903. 函数 `usart_lin_break_dection_length_config`**

函数名称	<code>usart_lin_break_dection_length_config</code>
函数原型	<code>void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lrlen);</code>
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}	
<code>usart_periph</code>	外设USARTx/UARTx
<code>USARTx</code>	$x=0,1,2,5$
<code>UARTx</code>	$x=3,4,6,7$
输入参数{in}	
<code>lrlen</code>	LIN模式中断帧长度
<code>USART_LBLEN_10B</code>	断开帧长度为10 bits
<code>USART_LBLEN_11B</code>	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_send_break`

函数`usart_send_break`描述见下表:

**表 3-904. 函数 `usart_send_break`**

函数名称	<code>usart_send_break</code>
函数原型	<code>void usart_send_break(uint32_t usart_periph);</code>
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}	
<code>usart_periph</code>	外设USARTx/UARTx
<code>USARTx</code>	$x=0,1,2,5$
<code>UARTx</code>	$x=3,4,6,7$

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 send break frame */

uart_send_break(USART0);
```

### 函数 **uart\_halfduplex\_enable**

函数uart\_halfduplex\_enable描述见下表：

表 3-905. 函数 **uart\_halfduplex\_enable**

函数名称	uart_halfduplex_enable
函数原型	void usart_halfduplex_enable(uint32_t usart_periph);
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}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 half duplex mode*/

uart_halfduplex_enable(USART0);
```

### 函数 **uart\_halfduplex\_disable**

函数uart\_halfduplex\_disable描述见下表：

表 3-906. 函数 **uart\_halfduplex\_disable**

函数名称	uart_halfduplex_disable
函数原型	void usart_halfduplex_disable(uint32_t usart_periph);
功能描述	失能USART半双工模式
先决条件	-
被调用函数	-
输入参数{in}	

<b>usart_periph</b>	外设USARTx/UARTx
<b>USARTx</b>	x=0,1,2,5
<b>UARTx</b>	x=3,4,6,7
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable USART0 half duplex mode*/
uart_halfduplex_disable(USART0);
```

### 函数 **usart\_synchronous\_clock\_enable**

函数usart\_synchronous\_clock\_enable描述见下表：

**表 3-907. 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	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1,2,5
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable USART0 CK pin in synchronous mode */
uart_synchronous_clock_enable(USART0);
```

### 函数 **usart\_synchronous\_clock\_disable**

函数usart\_synchronous\_clock\_disable描述见下表：

**表 3-908. 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	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,1,2,5
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable USART0 CK pin in synchronous mode */

uart_synchronous_clock_disable(USART0);
```

### 函数 **uart\_synchronous\_clock\_config**

函数**uart\_synchronous\_clock\_config**描述见下表：

**表 3-909. 函数 **uart\_synchronous\_clock\_config****

函数名称	<b>uart_synchronous_clock_config</b>
函数原型	<code>void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);</code>
功能描述	配置USART同步通讯模式参数
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,1,2,5
<b>输入参数{in}</b>	
<b>clen</b>	CK信号长度
<i>USART_CLEN_NO_NE</i>	8位数据帧中有7个CK脉冲，9位数据帧中有8个CK脉冲
<i>USART_CLEN_EN</i>	8位数据帧中有8个CK脉冲，9位数据帧中有9个CK脉冲
<b>输入参数{in}</b>	
<b>cph</b>	时钟相位
<i>USART_CPH_1CK</i>	在首个时钟边沿采样第一个数据
<i>USART_CPH_2CK</i>	在第二个时钟边沿采样第一个数据
<b>输入参数{in}</b>	
<b>cpl</b>	时钟极性
<i>USART_CPL_LOW</i>	CK引脚不对外发送时保持为低电平
<i>USART_CPL_HIGH</i>	CK引脚不对外发送时保持为高电平
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	

-	-
---	---

例如：

```
/* configure USART0 synchronous mode parameters */

uart_synchronous_clock_config(USART0,USART_CLEN_EN,USART_CPH_2CK,
USART_CPL_HIGH);
```

### 函数 **uart\_guard\_time\_config**

函数uart\_guard\_time\_config描述见下表：

**表 3-910. 函数 **uart\_guard\_time\_config****

函数名称	uart_guard_time_config
函数原型	void uart_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	-
<b>输入参数{in}</b>	
usart_periph	外设USARTx
USARTx	x=0,1,2,5
<b>输入参数{in}</b>	
gaut	保护时间值
0-0x000000FF	保护时间值
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure USART0 guard time value in smartcard mode */

uart_guard_time_config(USART0, 0x0000 0055);
```

### 函数 **uart\_smartcard\_mode\_enable**

函数uart\_smartcard\_mode\_enable描述见下表：

**表 3-911. 函数 **uart\_smartcard\_mode\_enable****

函数名称	uart_smartcard_mode_enable
函数原型	void uart_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	-
<b>输入参数{in}</b>	

<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1,2,5
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* USART0 smartcard mode enable */

uart_smartcard_mode_enable(USART0);
```

#### 函数 **uart\_smartcard\_mode\_disable**

函数uart\_smartcard\_mode\_disable描述见下表：

**表 3-912. 函数 **uart\_smartcard\_mode\_disable****

函数名称	uart_smartcard_mode_disable
函数原型	void uart_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	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1,2,5
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* USART0 smartcard mode disable */

uart_smartcard_mode_disable(USART0);
```

#### 函数 **uart\_smartcard\_mode\_nack\_enable**

函数uart\_smartcard\_mode\_nack\_enable描述见下表：

**表 3-913. 函数 **uart\_smartcard\_mode\_nack\_enable****

函数名称	uart_smartcard_mode_nack_enable
函数原型	void uart_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}	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1,2,5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 NACK in smartcard mode */
```

```
uart_smartcard_mode_nack_enable(USART0);
```

### 函数 **uart\_smartcard\_mode\_nack\_disable**

函数uart\_smartcard\_mode\_nack\_disable描述见下表：

**表 3-914. 函数 **uart\_smartcard\_mode\_nack\_disable****

函数名称	uart_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}	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1,2,5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 NACK in smartcard mode */
```

```
uart_smartcard_mode_nack_disable(USART0);
```

### 函数 **uart\_smartcard\_autoretry\_config**

函数uart\_smartcard\_autoretry\_config描述见下表：

**表 3-915. 函数 **uart\_smartcard\_autoretry\_config****

函数名称	uart_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

<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,1,2,5
<b>输入参数{in}</b>	
<b>scrtnum</b>	智能卡自动重试次数
<i>0-0xFFFFFFFF</i>	自动重试次数
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure smartcard auto-retry number */
uart_smartcard_autoretry_config (USART0, 0xFFFFFFFF);
```

### 函数 **uart\_block\_length\_config**

函数uart\_block\_length\_config描述见下表：

**表 3-916. 函数 **uart\_block\_length\_config****

<b>函数名称</b>	uart_block_length_config
<b>函数原型</b>	void usart_block_length_config(uint32_t usart_periph, uint32_t bl);
<b>功能描述</b>	配置智能卡T=1的接收时块的长度
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,1,2,5
<b>输入参数{in}</b>	
<b>bl</b>	块长度
<i>0-0xFFFFFFFF</i>	块长度
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure block length in Smartcard T=1 reception */
uart_block_length_config(USART0, 0xFFFFFFFF);
```

### 函数 **usart\_irda\_mode\_enable**

函数**usart\_irda\_mode\_enable**描述见下表：

**表 3-917. 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=3,4,6,7
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 3-918. 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=3,4,6,7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 IrDA mode */
```

```
uart_irda_mode_disableUSART0);
```

### 函数 **uart\_prescaler\_config**

函数**uart\_prescaler\_config**描述见下表:

**表 3-919. 函数 **uart\_prescaler\_config****

函数名称	uart_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	-
<b>输入参数{in}</b>	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
<b>输入参数{in}</b>	
psc	时钟分频系数
0-0xFF	时钟分频系数
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* configure the USART0 peripheral clock prescaler in USART IrDA low-power mode */
uart_prescaler_configUSART0, 0x00);
```

### 函数 **uart\_irda\_lowpower\_config**

函数**uart\_irda\_lowpower\_config**描述见下表:

**表 3-920. 函数 **uart\_irda\_lowpower\_config****

函数名称	uart_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	-
<b>输入参数{in}</b>	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
<b>输入参数{in}</b>	
irlp	IrDA低功耗模式或正常模式

<i>USART_IRLP_LOW</i>	低功耗模式
<i>USART_IRLP_NOR</i>	正常模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 IrDA low-power */
uart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

#### 函数 **uart\_hardware\_flow\_rts\_config**

函数uart\_hardware\_flow\_rts\_config描述见下表：

表 3-921. 函数 **uart\_hardware\_flow\_rts\_config**

函数名称	uart_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
<b>USARTx</b>	x=0,1,2,5
输入参数{in}	
<b>rtsconfig</b>	使能/失能RTS
<b>USART_RTS_ENA</b>	使能RTS
<b>BLE</b>	
<b>USART_RTS_DISA</b>	失能RTS
<b>BLE</b>	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 hardware flow control RTS */
uart_hardware_flow_cts_config(USART0, USART_RTS_ENABLE);
```

#### 函数 **uart\_hardware\_flow\_cts\_config**

函数uart\_hardware\_flow\_cts\_config描述见下表：

**表 3-922. 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	-
<b>输入参数{in}</b>	
usart_periph	外设USARTTx
USARTx	x=0,1,2,5
<b>输入参数{in}</b>	
ctsconfig	使能/失能CTS
USART_CTS_ENA BLE	使能CTS
USART_CTS_DISA BLE	失能CTS
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure USART0 hardware flow control CTS */
usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

#### 函数 usart\_break\_frame\_coherence\_config

函数usart\_break\_frame\_coherence\_config描述见下表：

**表 3-923. 函数 usart\_break\_frame\_coherence\_config**

函数名称	usart_break_frame_coherence_config
函数原型	void usart_break_frame_coherence_config(uint32_t usart_periph, uint32_t bcm);
功能描述	配置USART断开帧兼容模式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
usart_periph	外设USARTTx
USARTx	x=0,1,2,5
<b>输入参数{in}</b>	
bcm	
USART_BCM_NONE	没有检测到校验错误
USART_BCM_EN	检测到校验错误

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 break frame coherence mode */

uart_break_frame_coherence_config(USART0, USART_BCM_NONE);
```

#### 函数 **uart\_parity\_check\_coherence\_config**

函数uart\_parity\_check\_coherence\_config描述见下表：

**表 3-924. 函数 **uart\_parity\_check\_coherence\_config****

函数名称	uart_parity_check_coherence_config
函数原型	void usart_parity_check_coherence_config(uint32_t usart_periph, uint32_t pcm);
功能描述	配置USART校验兼容模式
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=3,4,6,7
输入参数{in}	
pcm	
USART_PCM_NONE	不检查奇偶校验
USART_PCM_EN	检查奇偶校验
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 parity check coherence mode */

uart_word_length_set(USART0, USART_PCM_NONE);
```

#### 函数 **uart\_hardware\_flow\_coherence\_config**

函数uart\_hardware\_flow\_coherence\_config描述见下表：

表 3-925. 函数 `uart_hardware_flow_coherence_config`

函数名称	<code>uart_hardware_flow_coherence_config</code>
函数原型	<code>void uart_hardware_flow_coherence_config(uint32_t usart_periph, uint32_t hcm);</code>
功能描述	配置USART硬件流控制兼容模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	<code>x=0,1,2,5</code>
输入参数{in}	
<code>hcm</code>	
<code>USART_HCM_NONE</code>	nRTS信号与USART_STAT0寄存器中RBNE位相同
<code>USART_HCM_EN</code>	nRTS信号在最后一个数据位被采样后被置位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 hardware flow control coherence mode */
uart_word_length_set(USART0, USART_HCM_NONE);
```

#### 函数 `uart_dma_receive_config`

函数`uart_dma_receive_config`描述见下表：

 表 3-926. 函数 `uart_dma_receive_config`

函数名称	<code>uart_dma_receive_config</code>
函数原型	<code>void uart_dma_receive_config(uint32_t usart_periph, uint32_t dmacmd);</code>
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}	
<code>usart_periph</code>	外设USARTx/UARTx
<code>USARTx</code>	<code>x=0,1,2,5</code>
<code>UARTx</code>	<code>x=3,4,6,7</code>
输入参数{in}	
<code>dmacmd</code>	使能/失能DMA接收功能
<code>USART_DENR_ENABLE</code>	使能DMA接收功能
<code>USART_DENR_DISABLE</code>	失能DMA接收功能

<i>ABLE</i>	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 DMA enable for reception */

uart_dma_receive_config(USART0, USART_DENR_ENABLE);
```

### 函数 **uart\_dma\_transmit\_config**

函数 **uart\_dma\_transmit\_config** 描述见下表：

**表 3-927. 函数 **uart\_dma\_transmit\_config****

函数名称	uart_dma_transmit_config
函数原型	void usart_dma_transmit_config(uint32_t usart_periph, uint32_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
<b>USARTx</b>	x=0,1,2,5
<b>UARTx</b>	x=3,4,6,7
输入参数{in}	
<b>dmacmd</b>	使能/失能DMA发送功能
<b>USART_DENT_EN</b>	使能DMA发送功能
<b>ABLE</b>	
<b>USART_DENT_DIS</b>	失能DMA发送功能
<b>ABLE</b>	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 DMA enable for transmission */

uart_dma_transmit_config(USART0, USART_DENT_ENABLE);
```

### 函数 **uart\_flag\_get**

函数 **uart\_flag\_get** 描述见下表：

**表 3-928. 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
<b>USARTx</b>	x=0,1,2,5
<b>UARTx</b>	x=3,4,6,7
<b>输入参数{in}</b>	
<b>flag</b>	USART标志位, 参考枚举usart_flag_enum
<b>USART_FLAG_CTS</b>	CTS变化标志
<b>USART_FLAG_LBD</b>	LIN断开检测标志
<b>USART_FLAG_TBE</b>	发送数据缓冲区空标志
<b>USART_FLAG_TC</b>	发送完成标志
<b>USART_FLAG_RB NE</b>	读数据缓冲区非空标志
<b>USART_FLAG_IDL E</b>	空闲线检测标志
<b>USART_FLAG_OR ERR</b>	溢出错误标志
<b>USART_FLAG_NE RR</b>	噪声错误标志
<b>USART_FLAG_FER R</b>	帧错误标志
<b>USART_FLAG_PE RR</b>	校验错误标志
<b>USART_FLAG_BSY</b>	忙状态标志
<b>USART_FLAG_EB</b>	块结束标志
<b>USART_FLAG_RT</b>	接收超时标志
<b>USART_FLAG_EPE RR</b>	校验错误超前检测标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	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 3-929. 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=0,1,2,5
UARTx	x=3,4,6,7
输入参数{in}	
flag	USART标志位, 参考枚举usart_flag_enum
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_RB_NE	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_EPE_RR	校验错误超前检测标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear USART0 flag */
usart_flag_clear(USART0,USART_FLAG_TC);
```

### 函数 usart\_interrupt\_enable

函数usart\_interrupt\_enable描述见下表:

**表 3-930. 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	-

<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>输入参数{in}</b>	
<b>interrupt</b>	USART中断
<i>USART_INT_PERR</i>	校验错误中断
<i>USART_INT_TBE</i>	发送缓冲区空中断
<i>USART_INT_TC</i>	发送完成中断
<i>USART_INT_RBNE</i>	读数据缓冲区非空中断和过载错误中断
<i>USART_INT_IDLE</i>	IDLE线检测中断
<i>USART_INT_LBD</i>	LIN断开信号检测中断
<i>USART_INT_ERR</i>	错误中断
<i>USART_INT_CTS</i>	CTS中断
<i>USART_INT_RT</i>	接收超时事件中断
<i>USART_INT_EB</i>	块结束事件中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable USART0 TBE interrupt */

uart_interrupt_enable(USART0, USART_INT_TBE);
```

#### 函数 **uart\_interrupt\_disable**

函数uart\_interrupt\_disable描述见下表：

表 3-931. 函数 **uart\_interrupt\_disable**

<b>函数名称</b>	uart_interrupt_disable
<b>函数原型</b>	void uart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);
<b>功能描述</b>	失能USART中断
<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
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>输入参数{in}</b>	
<b>interrupt</b>	USART中断

<b>USART_INT_PERR</b>	校验错误中断
<b>USART_INT_TBE</b>	发送缓冲区空中断
<b>USART_INT_TC</b>	发送完成中断
<b>USART_INT_RBNE</b>	读数据缓冲区非空中断和过载错误中断
<b>USART_INT_IDLE</b>	IDLE线检测中断
<b>USART_INT_LBD</b>	LIN断开信号检测中断
<b>USART_INT_ERR</b>	错误中断
<b>USART_INT_CTS</b>	CTS中断
<b>USART_INT_RT</b>	接收超时事件中断
<b>USART_INT_EB</b>	块结束事件中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable USART0 TBE interrupt */

uart_interrupt_disable(USART0, USART_INT_TBE);
```

#### 函数 **uart\_interrupt\_flag\_get**

函数 **uart\_interrupt\_flag\_get** 描述见下表：

**表 3-932. 函数 **uart\_interrupt\_flag\_get****

函数名称	<b>uart_interrupt_flag_get</b>
函数原型	FlagStatus <b>uart_interrupt_flag_get</b> (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	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx
<b>USARTx</b>	x=0,1,2,5
<b>UARTx</b>	x=3,4,6,7
<b>输入参数{in}</b>	
<b>int_flag</b>	USART中断标志，参考枚举 <b>usart_interrupt_flag_enum</b>
<b>USART_INT_FLAG_PERR</b>	校验错误中断标志
<b>USART_INT_FLAG_TBE</b>	发送缓冲区空中断标志
<b>USART_INT_FLAG_TC</b>	发送完成中断标志
<b>USART_INT_FLAG_RBNE</b>	读数据缓冲区非空中断标志

<i>_RBNE</i>	
<i>USART_INT_FLAG</i> <i>_RBNE_ORERR</i>	读数据缓冲区非空中断和溢出错误中断标志
<i>USART_INT_FLAG</i> <i>_IDLE</i>	IDLE线检测中断标志
<i>USART_INT_FLAG</i> <i>_LBD</i>	LIN断开检测中断标志
<i>USART_INT_FLAG</i> <i>_CTS</i>	CTS中断标志
<i>USART_INT_FLAG</i> <i>_ERR_ORERR</i>	过载错误中断标志
<i>USART_INT_FLAG</i> <i>_ERR_NERR</i>	噪声错误中断标志
<i>USART_INT_FLAG</i> <i>_ERR_FERR</i>	帧错误中断标志
<i>USART_INT_FLAG</i> <i>_EB</i>	块结束事件中断标志
<i>USART_INT_FLAG</i> <i>_RT</i>	超时事件中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	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 3-933. 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	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx/UARTx

少  
工  
时  
间

<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>输入参数{in}</b>	
<i>int_flag</i>	USART中断标志
<i>USART_INT_FLAG_CTS</i>	CTS变化中断标志
<i>USART_INT_FLAG_LBD</i>	LIN断开检测中断标志
<i>USART_INT_FLAG_TC</i>	发送完成中断标志
<i>USART_INT_FLAG_RBNE</i>	读数据缓冲区非空中断标志
<i>USART_INT_FLAG_EB</i>	块结束事件中断标志
<i>USART_INT_FLAG_RT</i>	超时事件中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* clear the USART0 interrupt flag */
uart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

## 3.30. 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[3.30.1](#)描述了WWDGT的寄存器列表，章节[3.30.2](#)对WWDGT库函数进行说明。

### 3.30.1. 外设寄存器说明

WWDGT寄存器列表如下表所示：

**表 3-934. 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

### 3.30.2. 外设库函数说明

WWDGT库函数列表如下表所示：

**表 3-935. 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 3-936. 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 3-937. 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 3-938. 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 3-939. 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
<i>WWDGT_CFG_PSC_DIV1</i>	WWDGT计数器时钟为 (PCLK/4096) /1
<i>WWDGT_CFG_PSC_DIV2</i>	WWDGT计数器时钟为 (PCLK/4096) /2
<i>WWDGT_CFG_PSC_DIV4</i>	WWDGT计数器时钟为 (PCLK/4096) /4
<i>WWDGT_CFG_PSC_DIV8</i>	WWDGT计数器时钟为 (PCLK/4096) /8
输出参数{out}	

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

例如：

```
/* confiure 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 3-940. 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	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-941. 函数 wwdgt\_flag\_get**

函数名称	wwdgt_flag_get
------	----------------

<b>函数原型</b>	FlagStatus wwdgt_flag_get(void);
<b>功能描述</b>	检查WWDGTR提前唤醒中断标志位是否置位
<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>	
<b>FlagStatus</b>	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 3-942. 函数 wwdgt\_flag\_clear**

<b>函数名称</b>	wwdgt_flag_clear
<b>函数原型</b>	void wwdgt_flag_clear(void);
<b>功能描述</b>	清除WWDGTR提前唤醒中断标志位状态
<b>先决条件</b>	-

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	2018 年 12 月 28 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.